Contrôle Générique de Paramètres pour les Algorithmes Evolutionnaires

Jorge Maturana

université angers

Année : 2009
N° d'ordre : 972

Contrôle Générique de Paramètres pour les Algorithmes Évolutionnaires

Thèse de doctorat

Spécialité : Informatique

École Doctorale STIM, ED 503

Présentée et soutenue publiquement

Le 24 Juin 2009

À Angers

Par **Jorge MATURANA**

Devant le jury ci-dessous :

Rapporteurs :	Christine Solnon,	Maître de Conférences, Université Lyon 1
	Ágoston E. Eiben,	Professeur, Vrije Universiteit Amsterdam
Examinateurs :	María Cristina Riff,	Professeur, Universidad Técnica Federico Santa María, Chile
	Marc Schoenauer,	Directeur de Recherche, INRIA
	Jin-Kao Hao,	Professeur, Université d'Angers
Directeur de thèse :	Frédéric Saubion,	Professeur, Université d'Angers

Laboratoire d'Etude et de Recherche en Informatique d'Angers
2, Bd Lavoisier, 49045 Angers Cedex 01

Remerciements

Tout d'abord, je veux remercier mon directeur de Thèse, Frédéric Saubion, pour son appui autant sur le plan personnel que professionnel depuis le début de mon séjour en France. Merci pour ton encadrement toujours constructif et pour m'avoir donné l'opportunité de suivre mon intuition. Je n'imagine pas de meilleur encadrant.

Je veux aussi remercier mes rapporteurs, Christine Solnon et Gusz Eiben, pour leurs commentaires qui ont beaucoup contribué à l'amélioration du manuscrit. Mes remerciements vont aussi à María Cristina Riff, Marc Schoenauer et Jin-Kao Hao, qui ont eu la gentillesse de participer à mon jury de thèse.

Une partie importante de la recherche présentée dans ce volume est issue de projets de collaboration. Au cours de ces travaux j'ai eu la satisfaction de partager des bons moments de travail avec plusieurs chercheurs qui ont enrichi mes points de vue et m'ont procuré le plaisir du contact humain : merci donc à Álvaro Fialho, Marc Schoenauer, Michèlle Sèbag et Frédéric Lardeux.

Je veux aussi remercier ceux qui ont eu le dévouement de corriger les nombreuses erreurs de langue du manuscrit et m'ont aidé à rendre le texte plus compréhensible. Merci à Carine, Massiel et Frédéric.

Les résultats de cette recherche auraient été impossibles à obtenir sans l'aide des ressources informatiques du département, et donc sans le support de l'équipe technique : merci à Eric, Frantz et Stéphane. Merci aussi aux secrétaires qui ont toujours eu du temps pour m'aider – avec un sourire – dans la préparation des déplacements et lors des démarches bureaucratiques. Merci à Catherine, Marie-Jo et Marie-Paule.

Étudier dans un pays étranger est, ainsi qu'une expérience professionnelle, une aventure personnelle dans laquelle j'ai eu l'énorme chance de découvrir tout un univers de visages. Ces remerciements ne seraient pas complets si j'oubliais mes camarades de laboratoire et des cours de langue, les chercheurs aux conférences, et les personnes connues dans d'autres cadres, qui m'ont fait le cadeau de partager un peu de leur culture et leur vision du monde. Au risque d'oublier quelqu'un, je pense à Eduardo, Lissete, Giglia, Wenceslao, Daniel, Sylvain, Adila, Julien, Thomas, Lionel, Mme. Blazec, Hazel, Bosco, Elisabeth, Anggela, Vincent, Cindy, Gonzalo, Mario, Chia-lin, Pascal, Faxin, Yili, Ludovic, Simone, Mathilde, David, Xavier, Serge, Pierre, Philippe, Russel, Gérard, Gilles, Álvaro, Marc, Zbyszek, Selmar, Gabriela, Ting, Camelia, Magali, Guillaume, Yann et Françoise.

Enfin, je veux remercier ma famille pour tout leur soutien pendant ces années, et particulièrement ma femme, Massiel, qui a toujours été là pour partager mes joies et frustrations, et m'a donné la force de continuer. Je n'y serais pas arrivé sans toi, merci de tout mon cœur.

À Massiel,
la femme de ma vie.

Table des matières

Introduction Générale

Le processus de résolution d'un problème d'optimisation comporte plusieurs étapes. Aujourd'hui, ces étapes définissent des tâches qui sont réalisées par des humains et des machines. Ce processus, présenté dans la figure 1, commence avec l'identification du problème à résoudre, issu d'une nécessité en particulier.

Afin de traiter le problème d'une façon systématique et bénéficier des outils mathématiques et informatiques disponibles, un modèle du problème est construit, dans le but d'obtenir une représentation formelle de la réalité, suffisamment précise pour que les résultats obtenus avec le modèle soient applicables à cette réalité. Un modèle d'optimisation mathématique est typiquement composé des variables de décision, qui codent souvent des choix possibles et des dimensionnements, de contraintes, qui expriment les conditions du problème, et d'un objectif, qui exprime les caractéristiques désirables de la solution à trouver.

Une fois le modèle obtenu, on doit séléctioner l'outil qui sera utilisé pour trouver une solution. Cet outil doit être capable de gérer la complexité du problème modélisé et de s'adapter à sa structure.

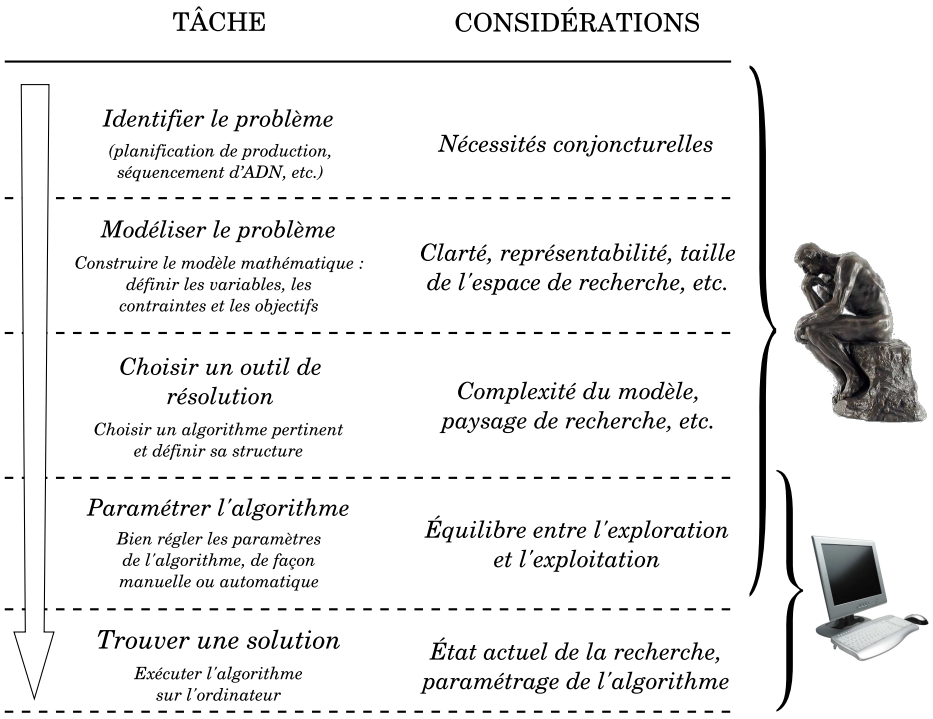

Fig. 1 – Démarche typique de résolution d'un problème d'optimisation

Même les outils les plus simples possèdent des paramètres qui doivent être réglés afin d'obtenir de bonnes performances. Ce réglage peut être fait d'une façon manuelle ou

automatique, selon le niveau de sophistication de l'approche. Finalement, le processus de recherche de solution s'effectue de manière automatique, compte tenu de la taille de l'espace de recherche (c'est-à-dire, l'ensemble des configurations possibles du problème) et du caractère répétitif de la tâche.

Actuellement, les trois premières étapes sont réalisées par un utilisateur humain, chargé d'identifier le problème, de construire le modèle et de choisir l'outil de résolution. La plupart du temps, le réglage des paramètres de l'algorithme choisi est accompli manuellement par un expert. Mais cette tâche peut aussi être faite automatiquement, ce qui a conduit à une grande quantité d'études ces dernières années.

Le réglage de paramètres est, néanmoins, souvent limité aux cas particuliers, c'est-à-dire, à des algorithmes spécifiques qui résolvent des instances particulières d'un problème. Ceci produit des solutions *ad-hoc*, difficiles à transposer à des situations différentes.

Parmi les outils de résolution on trouve les Algorithmes Évolutionnaires (AE). Les AEs ont largement été utilisés pour l'optimisation discrète et continue, balayant un large spectre d'applications. Basés initialement sur les principes de l'évolution naturelle, les AEs permettent de gérer un ensemble de configurations d'un problème, modifiées progressivement au moyen d'opérateurs de variation, afin de converger progressivement vers une solution optimale ou sous-optimale de bonne qualité. La métaphore évolutionniste amène à considérer ces configurations comme des individus formant une population qui évolue au moyen d'opérateurs génétiques, de mutation ou de croisement, selon leurs spécificités.

Ce cadre général de résolution de problèmes s'inscrit dans le contexte des méthodes dites métaheuristiques et les principales difficultés liés à la mise en oeuvre pratique de tels algorithmes reposent sur le choix d'un codage adéquat du problème ainsi que sur la définition d'opérateurs efficaces. Un fois cette architecture définie, le comportement de l'algorithme est en général déterminé par un ensemble de paramètres qui permettent d'agir principalement sur ses capacités à correctement explorer et exploiter l'espace de recherche.

Nous choisissons ici de distinguer les paramètres *structurels* de l'AE, tels que la taille de la population, des paramètres *comportementaux*, tels que les probabilités d'application des différents opérateurs. Le réglage de ces paramètres [Lobo *et al.*, 2007] s'avère alors être une tâche particulièrement ardue qui repose sur le savoir-faire, bien souvent empirique, de l'utilisateur et sur ses connaissances des caractéristiques du problème à résoudre. Dès lors, l'ajustement de paramètres s'appuie sur une série d'expériences, en général coûteuse. Une telle approche réduit considérablement l'universalité des valeurs obtenues et la transposition de ces expérimentations à d'autres problèmes. Une autre voie consiste alors à envisager un contrôle des paramètres durant l'exécution de l'AE. Ce contrôle peut être supervisé par un ensemble de connaissances acquises au préalable ou encore être abordé dans une optique plus autonome, les paramètres évoluant en fonction de l'état courant de la recherche.

Ce nouveau paradigme visant à produire des algorithmes de résolution autonomes est en pleine émergence, à l'intersection de plusieurs domaines de l'informatique, notamment en intégrant des outils issus de l'apprentissage automatique, de l'optimisation combinatoire ou encore de la programmation par contraintes [Battiti and Brunato, 2008; Battiti *et al.*, 2008; Hamadi *et al.*, 2008].

Outre le contrôle des paramètres, le choix même des composants structurels de l'AE nécessite également une expertise de la part de l'utilisateur car, si des opérateurs de variations standard existent dans la littérature (comme le croisement uniforme par exemple), l'obtention de résultats acceptables est inévitablement conditionné par la spécialisation du schéma algorithmique général et la définition d'opérateurs appropriés.

Les performances de l'AE doivent être évaluées en fonction de mesures permettant de rendre compte de l'état de la recherche en cours. Deux notions sont traditionnellement mises en avant comme les clés du succès de ces techniques de recherche heuristiques de solutions dans un espace potentiellement très vaste : la diversification et l'intensification. Alors que la diversification rend compte de l'aptitude de la méthode à explorer des zones variées de l'espace de recherche, l'intensification correspond à la propension qu'a l'algorithme à converger vers une solution dans une zone précise.

Le but de cette thèse est de développer une méthodologie qui permette d'automatiser les tâches de conception et paramétrisation des AEs. Contrairement aux méthodes existantes, nous voulons construire une abstraction des AEs, afin de proposer un contrôleur générique, capable de s'adapter à n'importe quel AE.

Cette généralité est atteinte par l'identification des aspects communs à tous les AEs. Comme cette tâche est actuellement accomplie par des experts humains, qui utilisent leur expérience dans la conception d'algorithmes et dans leur réglage, notre but est de saisir ce processus mental et de l'automatiser, afin de créer une boîte noire capable de s'adapter aux situations spécifiques, tout en gardant une simplicité d'utilisation. Cette thèse s'inscrit donc dans le cadre de l'intelligence artificielle, entendue, selon la définition de Rich et Knight, comme "l'étude des méthodes qui permettent aux ordinateurs de réaliser ce que les hommes font mieux actuellement" [Rich and Knight, 1991].

Principales Contributions

La principale contribution de cette thèse est sans doute l'abstraction des AEs, qui permet la conception d'un contrôleur générique, capable de travailler avec une grande variété de paramètres d'une façon uniforme. Ceci, grâce à la prise en compte des critères de haut niveau, inclus dans les différents contrôleurs implémentés (Contrôleur d'apprentissage Flou dans le chapitre 3, Compass et ExCoDyMAB dans le chapitre 4, et Forgeron dans le chapitre 5)

Un autre aspect intéressant est la facilité d'intégration du contrôleur. Les couches correspondant à l'algorithme contrôlé et au contrôleur sont clairement identifiées et leur communication est réduite au minimum. Ceci est particulièrement intéressant si on considère que des AEs non conçus originellement pour être contrôlés pourraient bénéficier d'un contrôleur "*plug and play*" avec un effort minimal.

Le fait de simplifier le contrôle jusqu'à un paramètre unique et facilement compréhensible par l'utilisateur facilite le conception des stratégies de contrôle. En effet, le réglage d'un paramètre unique et intuitif est bien plus simple que de gérer un ensemble de paramètres méconnus.

L'obtention d'un modèle lisible par l'utilisateur (chapitre 3) s'avère intéressant en tant qu'outil d'analyse d'algorithmes. De la même façon, la visualisation des profils des opérateurs dans les méthodes *Compass*, *DP* et *RP* (chapitres 4 et 5) peut fournir l'utilisateur d'une information précieuse sur le fonctionnement de l'AE.

L'utilisation des contrôleurs flous dans le but de modéliser le comportement de l'AE constitue une nouveauté, étant donné que ces contrôleurs avaient surtout été utilisés pour contrôler les paramètres des AEs. Une autre innovation réside dans le mécanisme de placement automatique des partitions floues dans contrôleurs Takagi-Sugeno.

Enfin, la formalisation d'une architecture qui permet de gérer en même temps la définition de l'AE et le réglage de ses paramètres (chapitre 5 et annexe A), ouvre une voie vers un éventuel standard de contrôle d'algorithmes.

Organisation

Cette thèse se décompose en deux parties. La première présente les concepts de base qui seront utilisés par la suite. Deux chapitres composent cette partie. Le chapitre 1 explique le fonctionnement des AEs et les principaux éléments qui influencent leur performance. Le chapitre 2 approfondit les enjeux de la paramétrisation des AEs et introduit les concepts qui motivent notre recherche.

La deuxième partie est consacrée aux contributions de cette thèse. Cette partie se compose de trois chapitres. Le chapitre 3 est dédié à l'étude des effets des paramètres sur le fonctionnement de l'algorithme au moyen d'un composant qui modélise cette relation pendant l'exécution de l'AE. Le chapitre 4 est consacré à la construction d'un contrôleur capable d'adapter rapidement les paramètres liés à l'application des opérateurs de manière générale. Finalement, le chapitre 5 présente une extension du contrôleur, afin de l'utiliser comme un outil de design, en choisissant automatiquement les opérateurs à inclure dans l'AE.

Des conclusions générales sont présentées ensuite, afin de résumer les aspects intéressants de notre recherche. Finalement, l'annexe A présente les éléments essentiels de l'implémentation de notre approche.

Première partie

État de l'Art

Chapitre 1

Algorithmes Évolutionnaires

Sommaire

1.1 Bref historique des Algorithmes Évolutionnaires

Les Algorithmes Évolutionnaires (AEs) sont des méthodes d'optimisation inspirées par les idées de l'évolution naturelle. C'est donc en biologie qu'il faut chercher les racines de cette discipline.

L'auteur sans doute le plus connu est le naturaliste anglais Charles Darwin (1809-1882) dont le livre "The Origin of Species" (1859) fut un exemple de vulgarisation de sujets jadis réservés aux naturalistes. Ses postulats furent largement issus de son expérience à bord d'un navire de l'armée Anglaise qui fit le tour du monde pendant 5 années, où il travailla en tant que naturaliste, observant la flore et la faune de différents endroits du monde. Cet ouvrage met l'accent sur la lutte pour la survie et les mécanismes d'adaptation des êtres vivants, qui leur permettent de générer des espèces différentes. Paradoxalement, ce livre ne parlait pas de l'évolution et ce n'est que à partir de 1900, quand Hugo de Vries et Carl Correns redécouvrent le travail de l'autrichien Gregor Mendel, que la notion d'évolution darwinienne naît. Mendel (1822-1884) réalisa une étude détaillée sur les variations dans des plantes et développa un modèle génétique proche de celui utilisé actuellement, en proposant une analyse de l'hérédité basée sur la combinaison des gènes dominants et récessifs.

Un autre nom lié au concept d'évolution est celui du naturaliste français Jean-Baptiste Lamarck (1744-1829), qui fut le premier à établir une théorie cohérente de l'évolution. Lamarck réunit les croyances de son époque sur la notion de génération spontanée et des concepts alchimiques sur les influences des quatre éléments, pour aboutir à une théorie qui mettait en jeu deux forces qui permettent l'évolution. La première force (*"Le pouvoir de la vie"*), qui pousse les individus à devenir de plus en plus complexes dans un but de perfectionnement, est tempérée par la deuxième force (*"L'influence des circonstances"*), qui oblige les individus à s'adapter au milieu naturel. Lamarck croyait que les adaptations des individus durant leur existence étaient transmises à leurs descendants, ainsi par exemple, le long cou des girafes serait le résultat de plusieurs générations d'individus essayant d'atteindre les feuilles des arbres.

Le modèle évolutif de Lamarck fut détruit par le travail du biologiste allemand August Weismann (1834-1914), qui démontra que les organismes supérieurs possèdent deux types de cellules, les *somatiques*, qui déterminent les caractéristiques de l'individu, et les *germinales*, qui passent l'information génétique aux descendants, et qu'il n'est pas possible que cette transmission d'expérience soit possible par voie génotypique. Les préceptes de l'évolution Lamarckienne restent, cependant, une source d'inspiration pour l'évolution artificielle.

L'évolution, telle qu'on la connaît aujourd'hui, est un processus où les individus doivent s'adapter à un mileu naturel qui impose une concurrence induite par des ressources limitées. Les variations entre individus sont expliquées par deux facteurs principaux : l'action des agents mutagènes, qui produisent des variations dans la chaîne d'ADN, et la combinaison de l'ADN des deux parents lors de la reproduction. L'évolution naturelle peut donc être vue comme un processus d'optimisation où les individus essayent de maximiser le profit qu'ils peuvent obtenir des conditions naturelles environnantes afin d'assurer leur descendance.

L'évolution artificielle s'inspire de ces idées pour l'optimisation mathématique. Ainsi,

les individus ne sont plus des êtres vivants, mais des solutions possibles d'un problème, les contraintes de l'environnement sont exprimées au moyen d'une fonction à satisfaire ou à maximiser, et les solutions sont modifiées avec des opérateurs qui simulent la mutation et la combinaison des individus. Les meilleures solutions ont plus de chance de se reproduire, dans un cadre d'amélioration continue.

Dans le monde de l'informatique, les AEs appartiennent à l'ensemble des *métaheuristiques*. Les métaheuristiques sont des méthodes générales pour résoudre des problèmes combinatoires qui sont difficilement traitables par des méthodes traditionnelles en raison de leur multimodalité, non-différentialité ou la taille de leurs espaces de recherche. Les EAs font aussi partie de l'intelligence artificielle (IA) en tant que méthodes inspirées par la nature.

Les premières traces de l'histoire des AEs remontent aux années 60 avec les travaux d'Ingo Rechenberg et Hans-Paul Schwefel sur les *stratégies d'évolution* [Rechenberg, 1973]. Les individus sont typiquement représentés par des vecteurs de réels, issus directement des variables du problème à résoudre. L'évolution des individus est produite en additionnant aux variables une valeur obtenue à partir d'une variable aléatoire Gaussienne.

La branche la plus populaire est probablement celle des *algorithmes génétiques*, proposés par John Holland dans les années 70 [Holland, 1975]. Ceux-ci sont caractérisés par un codage binaire pour représenter les variables du problème. Ce codage standardise les opérations de modification des individus : la *mutation* est effectuée en flippant la valeur de la variable (de 0 à 1 ou vice-versa), tandis que la combinaison des individus, appelée *croisement*, prend en compte une partie des valeurs de chaque parent pour construire le fils.

Il existe des paradigmes similaires, qui en fonction du type de problème, travaillent avec un codage spécial ou utilisent des opérateurs différents. Parmi ceux-ci on trouve la *programmation génétique*, développée par Ingo Rechenberg dans les années 60, qui a la particularité de travailler sur un codage d'arbres. Ce codage permet de représenter aisément des espaces de recherche complexes tels que les programmes informatiques. Un autre paradigme proche qui vit le jour à la même époque est la *Programmation évolutionnaire* [Fogel *et al.*, 1996] proposée par Lawrence Fogel, également dédiée à l'évolution de programmes.

Les *algorithmes mémétiques* [Moscato, 1989], proposés par Pablo Moscato en 1989, résultent quant à eux de la fusion entre les algorithmes génétiques et les méthodes d'amélioration locale de la solution. Ces algorithmes, parfois appelés *algorithmes évolutionnaires lamarckiennes*, améliorent localement les individus afin d'accélérer la convergence vers de bonnes solutions.

Un autre mécanisme de transmission d'expérience est utilisé dans les *algorithmes culturelles* [Reynolds, 1999], proposés par Robert Reynolds. Ici, additionnellement à l'information génétique des individus, on considère un *espace de croyances*, qui stocke des connaissances apportées par les meilleurs individus de la population sur différents aspects de la recherche en cours et du problème.

La communauté de l'évolutionnaire d'aujourd'hui mélange des éléments de ces paradigmes, tout en ajoutant de nouveaux aspects, ce qui produit une évolution constante de la discipline [Eiben and Smith, 2003; De Jong,]. Cela rend difuses les limites entre les différentes branches et fait naître de temps en temps de nouveaux algorithmes. On peut

donner en exemple l'*évolution différentielle* [Storn and Price, 1995], proposée par Kenneth Price et Rainer Storn en 1995, ou les *algorithmes d'estimation de distribution* [Mühlenbein and Paaß, 1996], proposés par Heinz Mülenbein et Gerhard Paaß en 1996.

1.2 Problèmes d'optimisation et de satisfaction

Avant d'expliquer le fonctionnement des AEs, nous allons voir quel type de problèmes ils peuvent résoudre. Nous en parlerons plus en détail dans la sous-section 1.4.1.

Un *problème d'optimisation* consiste à *maximiser* ou *minimiser* une fonction, en affectant les variables dont elle dépend. Plus formellement, soit E un ensemble de *solutions candidates*, et F une fonction $F : E \rightarrow \mathbb{R}$. La fonction F, appelée *fonction objectif*, affecte aux différentes solutions candidates de E une mesure dans $\mathbb{R}$. Un *problème d'optimisation* consiste à trouver la solution candidate qui maximise ou minimise (selon le but) la fonction F.

Les solutions candidates sont typiquement représentées par des vecteurs $(x_1, x_2, ..., x_n)$, et parfois soumises à contraintes. Selon les variables x_i dans les solutions candidates de E appartiennent à $\mathbb{R}$ ou à $\mathbb{Z}$, on parle de problèmes d'optimisation réels ou discrets.

Prenons par exemple le problème d'optimisation suivant :

$$minf(x) = (x-2)^2 + 1, \forall x \in \mathbb{Z}, -10 \leq x \leq 10 \tag{1.1}$$

Ici, l'ensemble d'instances a 21 valeurs possibles (les entiers de -10 à 10). La figure 1.1 montre les valeurs prises par la fonction f. On peut voir clairement que la solution au problème est trouvé pour $x = -2$.

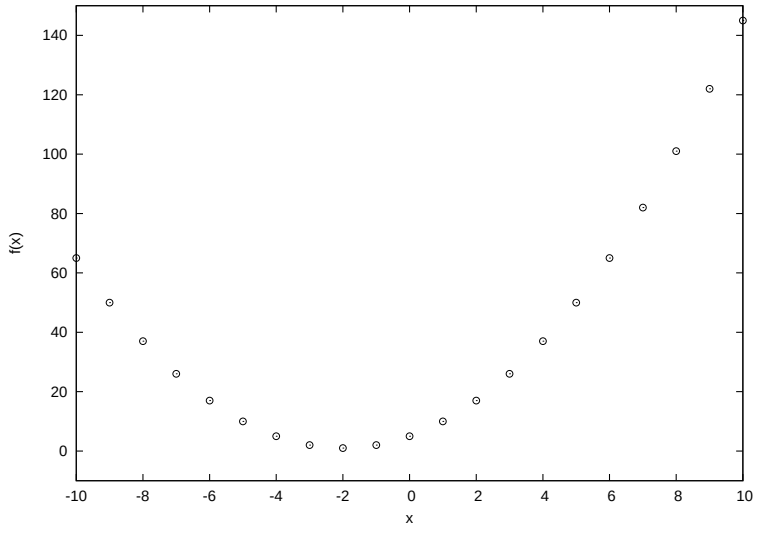

FIG. 1.1 – Solutions candidates d'un problème d'optimisation simple

On peut définir le *voisinage* d'une variable, comme un ensemble de valeurs proches de la valeur actuelle. Par exemple, on peut définir que le voisinage d'une variable discrète x_i

soit l'ensemble $\{x_i - 1, x_i, x_i + 1\}$. On appelle *optimum local* la valeur pour laquelle il n'y a aucune autre solution candidate dans son voisinage avec une valeur de fonction objectif supérieure (inférieure). Par contre, on appelle *optimum global* à la valeur pour laquelle il n'y a aucune autre solution candidate dans E avec une valeur de fonction objectif supérieure (inférieure). On peut trouver aussi des solutions candidates dont leur voisinage a la même valeur de fonction objectif, on parle alors d'un *plateau*. La figure 1.2 montre les valeurs d'une fonction objectif de $\mathbb{Z}^2 \mapsto \mathbb{R}$ pour différents valeurs des variables, ce qui définit le *paysage de recherche*.

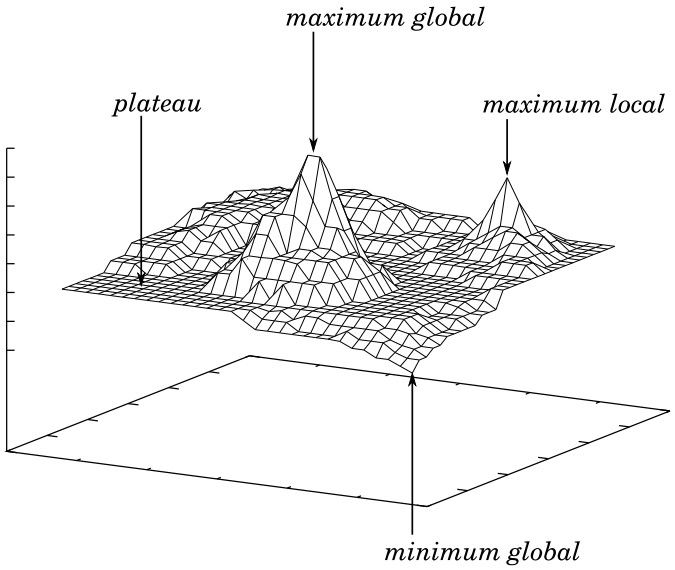

FIG. 1.2 – Paysage de recherche avec plateaux, et optimums local et global

Un autre type important de problèmes est appelé de *satisfaction de contraintes*. Ici, le but est de trouver une solution candidate qui satisfasse un ensemble de contraintes. Un problème typique est le *Boolean satisfiability problem (SAT)* [Cook, 1971], qui consiste à affecter des valeurs aux variables binaires afin de satisfaire une formule Booléenne exprimée en forme normale conjonctive (i.e., comme un ensemble de clauses formées par des variables binaires unies par des opérateurs *OR*, et unies avec d'autres clauses avec des opérateurs *AND*).

Dans la mesure où l'on peut mesurer le niveau de satisfaction de l'instance (par exemple, en considérant le nombre de clauses satisfaites dans une instance SAT), on peut traiter une problème de satisfaction de contraintes comme un problème d'optimisation, dont le but est de minimiser (jusqu'à zéro) le nombre de contraintes non satisfaites.

Un aspect intéressant est la *rugosité* de l'espace de recherche [Weinberger, 1990]. La rugosité peut être comprise intuitivement en regardant la figure 1.3, qui présente différents espaces de recherche correspondant à des problèmes de maximisation.

La rugosité est en rapport avec la corrélation des points du voisinage : s'il y a peu de variation entre les points (figure 1.3.a) la position d'un point donne une information plus valable sur son voisinage, car il n'y a pas des variations brusques entre les points. Cela

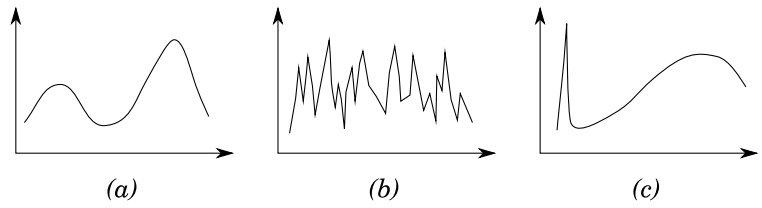

FIG. 1.3 – Différentes rugosités dans des espaces de recherche unidimensionnels

peut faciliter la recherche des maxima, par rapport à un espace de recherche plus rugueux (figure 1.3.b). Notez cependant, qu'un voisinage peu rugueux n'est pas toujours fiable, car il peut attirer la recherche vers des optimums locaux. Tel est le cas de la figure 1.3.c, où les méthodes de recherche (descente stricte ou gradient-based) seront attirées, la plupart du temps, vers le maximum local de droite.

Les problèmes vus dans cette section sont répresentatifs de l'utilisation des AEs. Il y a, cependant, d'autres problèmes qui requièrent une représentation plus complexe que celle du vecteur de valeurs à trouver. On rebondira sur ce sujet dans la section 1.3.2.

1.3 Architecture des AEs

1.3.1 Fonctionnement général

La figure 1.4 montre un schéma général de fonctionnement d'un AE. Une population d'individus qui codent des configurations possibles du problème est d'abord initialisée, puis évaluée. Les meilleurs individus de la population sont ensuite sélectionnés pour être modifiés au moyen d'opérateurs. Les individus issus de ces opérations sont alors insérés dans la population. Ce processus, allant de l'évaluation jusqu'à la réinsertion des individus modifiés, est répété jusqu'à ce qu'une condition d'arrêt soit satisfaite. Les détails du fonctionnement des AEs seront expliqués dans les sous-sections suivantes.

1.3.2 Codage

Le codage définit la représentation des variables du problème en vue de leur manipulation par l'AE. Dans le cas de variables numériques, ces variables peuvent être représentées telles quelles (i.e., par un vecteur de valeurs dans $\mathbb{R}$ ou $\mathbb{Z}$, selon le cas) [Goldberg, 1990b], ou bien par un codage binaire, qui permet de discrétiser l'espace de recherche. Par exemple, si dans un problème de n variables réelles $x_i, x_i \in \mathbb{R}, x_i \in [0,1], i = 1 \ldots n$, on choisit de coder chacune par m bits, chaque variable pourra avoir 2^m valeurs différentes dans l'intervalle $[0,1]$, et la représentation de chaque individu sera un vecteur binaire de taille $m \times n$, correspondant à la concaténation des n variables.

Ce codage a quelques inconvénients. Le premier est qu'il induit une inexactitude car le domaine des valeurs de la variable est discrétisé donc limité. Il faut aussi faire attention à la correspondance entre les valeurs binaires et réelles vis à vis des opérateurs de variation, car les différents bits ont différentes significations. Par exemple, considerons le cas

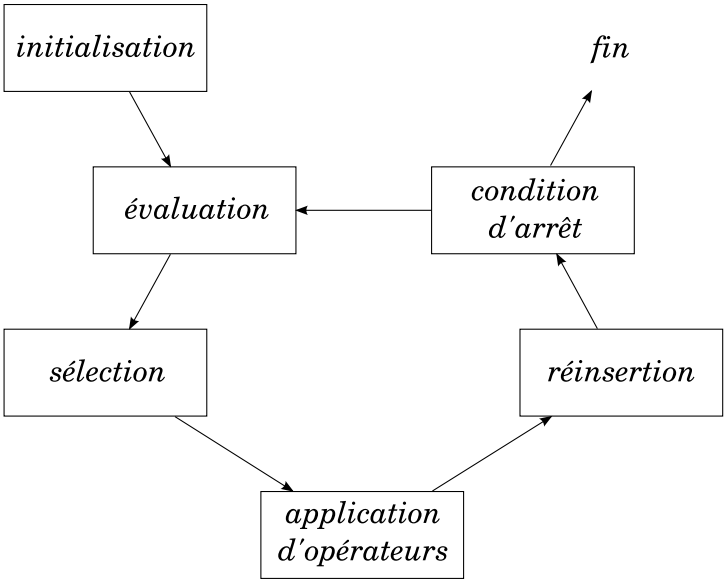

FIG. 1.4 – Fonctionnement général d'un AE

d'une variable dans $[0, 1]$ représentée par un nombre binaire de 7 bits. La chaîne 1001101, équivalente en décimal au nombre 77 et dans notre problème a la valeur $77/2^5 = 0.6015625$. Si on choisit de modifier un bit au hasard, la position de celui-ci a une grande influence sur l'effet de la modification : si on flippe le deuxième bit, pour obtenir la chaîne 1101101, on passe de 0.6015625 à 0.8515625, c'est-à-dire un écart de 0.25 sur la valeur de la variable originelle. Par contre, si on choisit de changer le sixième bit, pour obtenir la chaîne 1001111 on arrive à la valeur 0.6171875, c'est-à-dire, un écart de 0.015625 seulement. Le manque de corrélation entre les distances dans l'espace du codage et du problème est gênant car un petit changement dans le premier peut occasionner un éloignement excessif d'une valeur optimale dont on pourrait avoir été proche avant d'appliquer l'opérateur. Tout ceci induit des temps de convergence plus longs [Michalewicz, 1996].

Pour pallier ce problème de corrélation, un autre ordre sur les nombres binaires a été proposé. Dans le codage de *Gray*, les changements d'un bit produisent toujours le même changement de valeur dans le problème initial, quelle que soit la position de celui-ci. La table 1.1 compare le codage binaire traditionnel et celui de Gray pour une chaîne de 4 bits.

Un autre codage commun est celui de permutation. Ce codage est utilisé dans des problèmes où il faut établir un ordre afin de positionner ou de séquencer un ensemble de situations. Dans ce codage, la position qu'occupe chaque variable correspond à cet ordre. Par exemple, s'il y a 5 opérations à effectuer, le codage $(4, 5, 1, 3, 2)$ veut dire qu'il faut faire l'opération 4 en première place, la 5 en deuxième, et ainsi de suite. On impose que chaque opération doit se faire une seule fois. Un exemple de porblème qui utilise ce codage est celui du *voyageur de commerce*, qui consiste à trouver le plus court chemin qui relie un nombre donnée de villes, séparées par des distances connues.

decimal	bin.trad	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

decimal	bin.trad	Gray
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Tab. 1.1 – Comparaison des codages binaire traditionnel et Gray

Dans d'autres cas, les variables ne respectent pas de relation d'ordre. Imaginons par exemple qu'il faut colorer, avec quatre couleurs, les pays sur une mappemonde, de telle sorte que deux pays voisins ne soient pas de la même couleur. Le codage utilisé dans ce cas correspond à un vecteur de taille égale au nombre de pays sur la carte, où chaque variable est affectée d'un nombre correspondant à la couleur utilisée.

Parfois on utilise des structures de données plus complexes afin de réduire l'espace de recherche. Par exemple, [Maturana and Riff, 2007] traite un problème de scheduling de générateurs électriques sur un horizon de 24 heures, discrétisé heure par heure. L'énergie demandée est connue à l'avance et elle doit être produite par 10 à 100 générateurs différents, qui fournissent chacun différentes quantités d'énergie. Tous les générateurs ont des contraintes de temps minimales d'utilisation et d'arrêt. Le problème consiste à produire l'énergie nécessaire au coût minimal. Puisqu'il existe juste deux états dans lesquels le générateur peut se trouver (allumé/éteint) et que l'horizon de temps est discrétisé en 24 heures, la représentation la plus intuitive des variables du problème est une matrice bidimensionnelle de binaires avec 24 colonnes pour représenter le temps, et 10-100 lignes pour représenter les machines. Néanmoins, l'espace de recherche code plusieurs planifications invalides, dûes aux contraintes de temps minimales. Les auteurs proposent d'utiliser un codage basé sur des paires $(i_k, t_k) \in \mathbb{N}$, où i_k est l'heure de démarrage du générateur k, et t_k correspond à la somme des temps d'utilisation et d'arrêt. Grâce à l'exclusion des individus invalides, l'espace de recherche est réduit d'un ordre de magnitude (de N^2 à N) par rapport au nombre de générateurs.

Une autre question qui se pose à l'heure de choisir le codage est de savoir si on permettra ou non l'inclusion des individus ne satisfaisant pas les contraintes. D'un côté, leur exclusion diminue la taille de l'espace de recherche, mais cette discrimination peut occasionner la division de l'espace de recherche en plusieurs zones déconnectées, rendant plus difficile son exploration. D'ailleurs, certains problèmes de satisfaction consistent précisément à trouver un individu valide, donc la question ne se pose même pas. Dans la pratique, l'inclusion de ces individus invalides n'est pas rare.

Un cas standard de codage utilisant des structures de données complexes est la programmation génétique [Koza, 1992], dont le but est de représenter des programmes qui évoluent. On a donc besoin de travailler avec une représentation capable de coder des individus de taille variable et structurés selon les règles de syntaxe des programmes. Cela

est possible grâce à l'utilisation des arbres. Par exemple, pour coder le terme $x=(a+b)/c$ on utilisera l'arbre **(=,x,(/,(+,a,b),c))** (en notation préfixée). Comme la taille de l'arbre n'est pas limitée, il faut mettre en place des mécanismes pour empêcher une croissance démesurée des individus, phénomène connu sous le nom de *bloating*.

Les différents choix de codage induisent des espaces de recherche ayant des caractéristiques différentes pour le même problème, et influençant le design des opérateurs qui modifieront les individus.

1.3.3 Opérateurs

Afin de progresser dans la recherche de solutions, il est nécessaire de produire des variations dans les configurations candidates. Ceci est réalisé au moyen des opérateurs. En général, on peut trouver deux classes principales : les opérateurs de mutation et ceux de recombinaison. Les opérateurs de mutation sont tous ceux qui prennent une solution candidate et la modifient, notamment pour explorer son voisinage. Les opérateurs de recombinaison, par contre, prennent deux (ou plusieurs) candidats et obtiennent un nouvel individu (fils), qui contient idéalement le meilleur des parents. Les opérateurs sont souvent appliqués selon une *probabilité d'application*.

Comme mentionné précédemment, le codage détermine le fonctionnement des opérateurs. Ceux-ci doivent être capables d'atteindre toutes les valeurs possibles dans l'espace de recherche, sinon la recherche sera limitée à une partie qui peut ne pas avoir de solutions intéressantes.

Dans les AEs à codage binaire, notamment dans les algorithmes génétiques, les principaux opérateurs sont la mutation et le croisement. Dans la plupart des cas, la mutation est faite simplement par le "flip" de la valeur de la variable (i.e., de 0 a 1 ou vice-versa). Cette opération est effectuée sur un bit du codage, et selon une probabilité plutôt basse. La mutation permet d'explorer des zones non connues de l'espace de recherche. De l'autre coté, le croisement consiste à combiner l'information de deux individus, en prenant une partie des valeurs d'un des parents, et complétant avec l'autre. Il existe plusieurs façons de combiner les individus, soit en choisissant un point pour croiser les parents, soit en en choisissant plusieurs. On peut aussi réaliser un croisement uniforme où la valeur de chaque bit du fils est issue d'un des parents avec une probabilité uniforme. La figure 1.5 montre quelques formes de croisement pour les codages binaires. Dans (a) apparaît le croisement en 1 point, où deux fils sont obtenus par l'échange des deux parties ainsi définies par la coupe. (b) montre une extension de cette idée, cette fois en coupant à plusieurs endroits. On peut aussi n'obtenir qu'un fils en prenant au hasard (avec une probabilité uniforme) chaque bit à partir des deux parents (c), voire depuis plusieurs (d).

Pour les algorithmes à codage réel, il existe aussi plusieurs choix. Dans les stratégies d'évolution par exemple, la variation est faite en additionnant aux variables une valeur obtenue à partir d'une variable aléatoire respectant une loi de distribution $N(0, \sigma)$. La figure 1.6 montre quelques choix pour implémenter la recombinaison dans un espace de recherche bidimensionnel. Une première option consiste à affecter au fils la valeur de la variable d'un des deux parents, avec une probabilité uniforme (figure 1.6.a). On peut aussi créer un nouveau fils comme une combinaison linéaire entre les deux parents au moyen

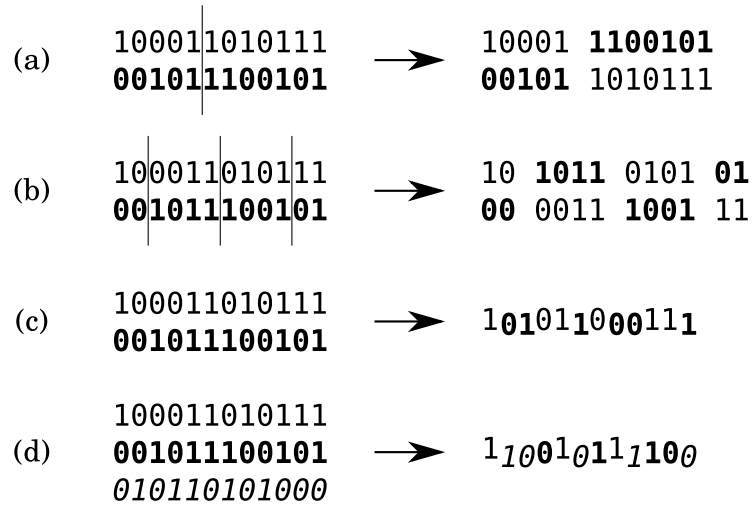

FIG. 1.5 – Croisements de codage binaire : (a) en 1 point, (b) en n-points, (c) uniforme, (d) uniforme à partir de n parents

d'un poids $w \in [0, 1]$ (figure 1.6.b). Une autre option est de construire le fils à partir des poids w_i pour chaque variable (figure 1.6.c). Finalement, plus de deux parents peuvent être utilisés, comme le montre la figure 1.6.d.

Pour les algorithmes à codage par permutation, le choix de mutation est, dans la plupart des cas, l'échange des variables entre deux positions, connu sous le nom de *swap*. Pour la recombinaison, il existe quelques options visant à garder, dans la mesure du possible, soit la position des variables, telles que le *Partially mapped crossover* (ou PMX)[Goldberg and Lingle, 1985] et le *Cycle crossover* (CX)[Oliver *et al.*, 1987], soit l'ordre relatif entre les variables, comme dans le *Order crossover* (OX)[Davis, 1985].

Il faut remarquer que tous les opérateurs vus jusqu'ici ne prennent pas en compte le résultat produit par son application. Bien qu'en général ils permettent de bien explorer l'espace de recherche dans l'espoir de trouver de bonnes solutions, on pourrait envisager d'inclure un composant "intelligent", afin d'accélérer la convergence vers de bons individus. C'est ici que l'idée de l'évolution lamarckienne fait son apparition. Ainsi, les EAs peuvent être *hybridés* en ajoutant des opérateurs spécialisés qui prennent en compte soit l'état de la recherche, soit le domaine du problème. Par exemple, une mutation intelligente peut décider de ne pas modifier une variable quand il est évident qu'elle est proche d'une bonne valeur, ou une recombinaison intelligente pourrait empêcher de croiser des individus incompatibles dont le fils est destiné à l'échec. On trouve ici des opérateurs dits de *réparation*, chargés de restaurer des individus devenus invalides après une opération.

Les algorithmes mémétiques ont pris la voie de l'hybridation en intégrant des algorithmes de recherche locale. Dans plusieurs domaines, notamment celui des CSP difficiles, des AEs simples tels que les algorithmes génétiques ne peuvent pas trouver de solution sans l'aide de la recherche locale.

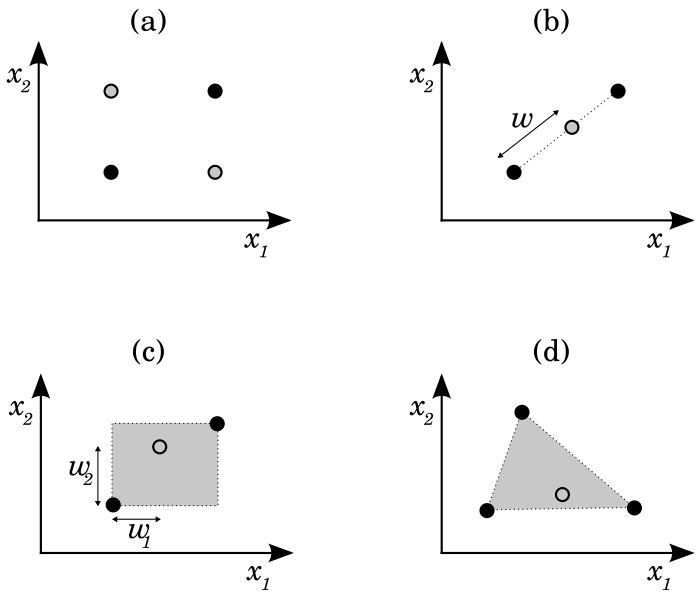

FIG. 1.6 – Croisements de codage réel : les points noirs représentent les parents et les gris le fils. La zone grise montre où peut se trouver la descendance. (a) uniforme, (b) arithmétique avec 1 poids, (c) arithmétique avec n poids, (d) arithmétique avec n poids et trois parents

1.3.4 Évaluation

Afin d'évaluer la qualité des individus, il faut définir une mesure, appelée *fonction d'évaluation* qui établit un ordre entre individus. Cette fonction est en rapport avec la fonction objectif du problème à résoudre. Dans plusieurs cas, la fonction d'évaluation est simplement la fonction objectif, néanmoins il est aussi possible d'incorporer d'autres composantes en rapport avec le processus d'évolution. Par exemple, pour des problèmes de satisfaction de contraintes, on peut incorporer une pénalisation sur les contraintes non satisfaites, d'une façon similaire à la *relaxation lagrangienne*. On peut aussi pénaliser des individus trop proches, afin d'empêcher une concentration trop grande des individus sur une zone particulière de l'espace de recherche.

Dans certains problèmes, notamment des problèmes combinatoires, il est possible d'avoir plusieurs configurations avec la même valeur de fonction objectif. Cela génère des *plateaux*, où l'algorithme est incapable de savoir quelle est la meilleure direction à prendre pour améliorer. Dans ce cas, il est bénéfique d'incorporer d'autres éléments pour établir des différences parmi ces configurations et "marquer le chemin" vers les optimums du problème [Rodríguez-Tello, 2007].

Il existe des cas où la qualité de l'individu ne peut pas être mesurée de façon directe. Imaginons, par exemple, qu'on essaie d'optimiser la planification des matériaux et des processus dans une usine, afin de maximiser sa performance. Pour des problèmes de taille réelle, l'évaluation est tellement complexe qu'il est préférable de remplacer la fonction

d'évaluation par une série de simulations. On utilise les termes biologiques de *génotype* et *phénotype* pour faire la différence entre l'individu lui-même, c'est-à-dire son codage, et l'expression de son codage en tant que solution. En biologie, le génotype se réfère à la constitution génétique d'un organisme, alors que le phénotype est une caractéristique observable, produit de l'expression du génotype *et* de l'influence des facteurs entourants.

On a parlé jusqu'ici uniquement des problèmes avec un seul objectif. En effet, lorsqu'on n'a qu'un seul critère à optimiser, la différence entre la qualité des individus peut être nettement établie. Il y a pourtant des cas où on est intéressé à optimiser plusieurs critères à la fois. C'est ici qu'apparaît la notion d'*optimum de Pareto* [Pareto, 1896]. Une solution est Pareto-optimale s'il n'existe aucune autre solution qui soit meilleure sur tous les critères. Cela peut être mieux compris en regardant la figure 1.7 : ici, on a deux critères à maximiser, et plusieurs solutions marquées avec des points. Pour tous les points, sauf ceux noircis, il en existe un autre qui est meilleur sur les deux critères. Pour les points noirs, par contre, il n'existe aucun autre placé à droite et en haut, donc ils sont Pareto-optimaux. L'ensemble des points noirs, appelés *non-dominés*, constitue la *frontière de Pareto*. On dit qu'un point a un *rang* égal au nombre de points qui le dominent. Les points de la frontière de Pareto ont donc un rang égal à zéro. Notez que, car les deux critères sont d'importance égale, on ne peut pas établir d'ordre entre les points de la frontière de Pareto. Tous ces points sont donc également valables, et si on ne cherche qu'une solution, il faudra choisir parmi eux avec un critère supplémentaire.

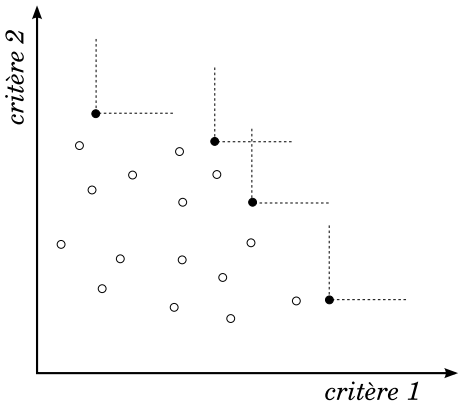

Fig. 1.7 – Frontière de Pareto

1.3.5 Population

En général, on peut classifier les métaheuristiques selon qu'elles travaillent avec un ou plusieurs individus. Dans le premier groupe, on trouve la *recherche tabue* [Glover, 1989; Glover, 1990], le *recuit simmulé* [Aarts and Korst, 1989] et quelques variantes des *stratégies d'évolution* [Rechenberg, 1973]. Dans le deuxième, les *AEs* en général et la *Swarm intelligence* [Blum and Merkle, 2008].

Dans les AEs, le fait de travailler avec plusieurs individus à la fois donne l'avantage de pouvoir explorer d'une façon plus étendue l'espace de recherche. Cependant, pour que cela

se passe ainsi, il faut veiller à l'utilisation correcte de la population. Un concept central est la *diversité* de la population. En effet, si tous les individus sont identiques, on ne fait que multiplier une recherche individuelle par la taille de la population, perdant du temps de calcul.

La diversité peut être maintenue par différents moyens. On peut choisir d'encourager l'application des opérateurs qui diversifient, ou d'utiliser des opérateurs qui interdisent de recombiner des individus différents, favorisant la *spéciation* (i.e., la formation de différentes espèces, ou groupes d'individus avec des caractéristiques différentes du reste), ce qui est connu sous le nom de *niching* [Mahfoud, 1995]. À la limite, on peut attendre que la population converge dans un optimum et la réinitialiser avec des individus générés au hasard afin de rétablir la diversité perdue.

Dans la plupart des cas, l'encouragement de la diversité implique une perte de qualité. En effet, le fait d'imposer que les individus soient différents souvent éloigne les individus des zones de bonne qualité. À l'exception du traitement de problèmes *multimodaux* (i.e., ceux qui ont plusieurs optimums de qualité), la diversité et la qualité sont des objectifs contradictoires. On peut concevoir cette situation en tant qu'un problème multiobjectif, et utiliser les concepts de Pareto.

Bien que dans la plupart des cas la population est générée au hasard, il est parfois bénéfique d'utiliser des individus de bonne qualité, issus d'un processus de recherche spécifique, particulièrement quand il existe une méthode connue capable de produire une population diversifiée avec un certain niveau plus rapidement que l'AE ne le ferait.

Un autre aspect important est la taille de la population. Quelques auteurs soutiennent qu'une grande taille est plus importante au début de la recherche qu'à la fin [De Jong, 2007], car tout au début on traite typiquement un espace complexe, plein de "pièges" alors que vers la fin les individus sont concentrés dans des zones à topologie plus simple. En tout cas, la taille doit être en concordance avec les objectifs de la recherche (veut-on trouver juste une solution ou plusieurs ?) et le niveau de multimodalité de l'espace de recherche du problème (un paysage rugueux requiert plus d'individus qu'un paysage lisse).

Il est possible aussi de grouper les individus dans des populations semi-indépendantes, ce qui peut aider à maintenir la diversité et améliorer la performance [Cantú-Paz, 1997; Wineberg and Chen, 2004; Tsutsui *et al.*, 1997]. Divers niveaux de groupement sont possibles, selon la taille des sous-populations choisies.

1.3.6 Sélection et Réinsertion

Il existe deux moments où des individus sont choisis dans chaque cycle de l'AE. Le premier est avant d'appliquer les opérateurs et le deuxième est pour décider si les individus générés seront ou non inclus dans la population.

En général, on peut distinguer deux approches génériques de sélection, dénommées *plus* et *comma*. La différence entre les deux réside dans l'ensemble où sont choisis les survivants : s'ils sont choisis parmi les individus générés et leurs parents, on parle d'une réinsertion *plus*, et s'ils sont issus uniquement des individus générés, on parle d'une réinsertion *comma*.

Parmi les sélections *plus*, on trouve le schéma *steady state*, où typiquement un individu est généré et doit concurrencer ses parents au moment d'être inséré dans la population. De

cette manière un bon individu peut rester longtemps dans la recherche, et par conséquent les nouveaux individus doivent concurrencer une population contenant plusieurs individus "mûrs" (on dit que la *pression de sélection* est élevée). Cela induit une recherche plus stable, mais aussi une exploration plus restreinte.

Parmi les sélections *comma*, on trouve le schéma des *générations*, qui consiste soit à générer tous les individus afin de créer une population secondaire, soit de créer une population secondaire et appliquer les opérateurs sur elle. De cette façon, tous les parents meurent au terme d'un cycle de l'AE, ce qui accorde une plus grande liberté d'exploration à la recherche, car la concurrence est établie uniquement parmi des individus générés, ce qui, en moyenne, est moins exigeant que concurrencer des individus mûrs. Éventuellement, la qualité de la population peut baisser, ou le meilleur individu peut être perdu à cause des opérations aléatoires.

Une façon de protéger la rémanence des bons individus, notamment quand une sélection comma est utilisée, est l'*élitisme*, pratique qui consiste à assurer la survivance du meilleur individu tout au long de la recherche.

La sélection des individus qui seront générés répond au critère que seuls les meilleurs individus d'une population ont accès à la reproduction, comme cela se passe dans la nature. Elle peut être implémentée de différentes façons, parmi lesquelles on peut trouver :

Roulette : l'individu est choisi avec une probabilité proportionnelle à sa valeur de fonction d'évaluation [Holland, 1975]. C'est la façon la plus intuitive de choisir un individu, mais le problème se pose de choisir presque toujours le même individu s'il en existe un bien meilleur que les autres.

Basée en rangement : les individus sont mis en ordre selon leur évaluation, puis affectées selon une échelle (par exemple, dans une population de 100 individus, affecter la valeur 100 au premier, 99 au deuxième, etc.). Ensuite on effectue une sélection par roulette sur ces nouvelles valeurs [Baker, 1985]. Cela aide à pallier le problème de la roulette, en linéarisant les mesures selon lesquelles on fait la sélection.

Tournoi : un sous-ensemble d'individus est choisi au hasard, et le meilleur d'entre eux est sélectionné [Miller and Goldberg, 1995]. Ici, on a plus de chance de choisir des individus de façon plus équitable, en favorisant toujours les meilleurs.

Les mécanismes pour décider de l'insertion ou non d'un individu généré dans la population consiste typiquement à comparer s'il est meilleur que le moins bon des individus actuellement dans la population.

1.3.7 Évolution

En général, le processus de recherche de solution d'un problème peut être vu comme un problème bi-objectif : d'une part on veut progresser vers les meilleures zones de l'espace de recherche, et de l'autre, on veut parcourir la plupart de l'espace de recherche, afin de minimiser le risque de passer à côté d'une zone intéressante.

Ces deux objectifs sont cependant opposés dans un temps fini (ce qui est le cas), donc il faut chercher un compromis entre les deux. Si le premier objectif, appelé *exploitation*, est prépondérant par rapport au deuxième (appelé *exploration*), on risque de converger

prématurément vers un optimum local. Par contre, si l'exploration prédomine sur l'exploitation, on risque de "flâner" dans l'espace de recherche sans jamais trouver de solution.

L'équilibre entre l'exploration et l'exploitation est donc essentiel. Cet équilibre dépend notamment des opérateurs et de la sélection, et en dernier lieu, des paramètres qui contrôlent ces opérations. On discutera plus largement de ce sujet dans la sous-section 2.1.1.

Il faut noter que les termes *exploration* et *exploitation* sont souvent utilisés d'une façon intuitive pour exprimer les deux tendances, la plupart de temps contradictoires, et quelques fois associés à des opérateurs ou parties de l'AE. Une étude approfondie de la bibliographie à ce sujet montre que les deux forces ne sont pas toujours contradictoires et elles ne sont pas forcement associées à une partie ou une autre de l'AE [Eiben and Schippers, 1998]. Dans ce travail, on considère ces concepts en tant que forces normalement contradictoires non associées à des parties spécifiques de l'algorithme.

1.3.8 Paramètres

Toutes les décisions liées au fonctionnement de l'AE peuvent être paramétrées. En effet, le codage des individus, la taille de population, les opérateurs que seront utilisés et la probabilité de chacun d'eux, la façon de faire l'évaluation, la condition d'arrêt, etc. peuvent être réglés par des paramètres.

On distingue ici les paramètres *structurels* des paramètres *comportementaux*. Les premiers sont en rapport avec le design de l'AE, par exemple un paramètre qui détermine si un opérateur est présent ou pas dans l'algorithme, ou un autre qui choisit le type de sélection que l'on utilisera. Les paramètres comportementaux, par contre, sont ceux qui règlent le fonctionnement des éléments présents dans l'AE, typiquement les probabilités d'application de tel ou tel opérateur. En général, le réglage des paramètres structurels est plutôt vu comme un choix de design de l'algorithme, par rapport aux paramètres comportementaux, qui guident l'exécution de l'AE.

Les paramètres peuvent être réglés de différentes façons. La plus simple consiste à trouver, au moyen d'expériences préalables, de bonnes valeurs. Il existe aussi des schémas plus sophistiqués, qui consistent à modifier les valeurs des paramètres durant l'exécution de la recherche. Cette modification peut être basée soit sur des règles pré-définies, soit sur l'état actuel de la recherche.

Cette thèse se centre sur l'étude du contrôle de paramètres. Le chapitre 2 présente en détail les types et outils pour régler les paramètres des AEs.

1.4 Domaines d'application

Les AEs ont été appliqués dans pratiquement tous les domaines où l'on retrouve des problèmes d'optimisation [Michalewicz and Fogel, 1998; Poli *et al.*, 2008; Haupt and Haupt, 1998; Coley, 1999; Gen and Cheng, 1997; Clark *et al.*, 2000; Bentley, 1999; Baeck *et al.*, 1997]. Étant donné leur généralité, les AEs sont plutôt un cadre algorithmique pour la résolution des problèmes.

On peut trouver des applications dans des domaines d'optimisation variés comme la chaîne logistique et le commerce [Borisovsky *et al.*, 2009; Jawahar and Balaji, 2009; Soak *et al.*, 2008; Hurley *et al.*, 1998], le transport [Mendoza *et al.*, 2009; Liu *et al.*, 2008; Chung *et al.*, 2009], la production [Shao *et al.*, 2009; Zhou *et al.*, 2009; Alexouda, 2005; Rom and Slotnick, 2009; Elaoud *et al.*, 2007; Levitin, 2006], la production d'énergie [Zio *et al.*, 2009; Wu, 1999], l'investissement [Soleimani *et al.*, 2009], l'hydrologie [Elferchichi *et al.*, 2009; Chen and Chang, 2009; Loonen *et al.*, 2006], la biologie cellulaire [Mohebbi *et al.*, 2008; Kim *et al.*, 2007], la génétique [Moscato *et al.*, 2007; Kim *et al.*, 2008], la médecine [Setzkorn *et al.*, 2007; Smith and Timmis, 2008], la pharmacologie [Terfloth and Gasteiger, 2001], la recherche documentaire [Silva *et al.*, 2009] ou les jeux [Berghman *et al.*, 2009].

1.4.1 Problèmes de satisfaction de contraintes et d'optimisation sous contraintes

Nous nous plaçons ici dans le contexte de la résolution de problèmes d'optimisation combinatoires classiquement définis par un ensemble de variables $Var = \{x_1, \cdots, x_n\}$, auxquelles sont associés des domaines de valeurs possibles $D = \{D_1, \cdots, D_n\}$, D_i étant l'ensemble des valeurs que peut prendre la variable x_i. L'espace de recherche S correspond au produit cartésien des domaines, $S = \Pi_{i=1}^{n} D_i$. Nous considérons un ensemble de contraintes C et une solution réalisable est alors une affectation de valeurs aux variables satisfaisant toutes les contraintes de C et respectant les domaines initiaux de D. Nous considérons également une fonction objective $f : X \rightarrow \mathbb{R}$. Une solution optimale est une solution réalisable maximisant ou minimisant, selon le cas, la fonction f.

Dans l'approche classique de la complexité [Papadimitriou, 1994], les problèmes sont regroupés en classes de complexité en fonction des ressources mises en œuvre par les algorithmes pour les résoudre. Les mesures de complexité sont obtenues en donnant une borne pour le temps de calcul ou la quantité de mémoire utilisé : Cette borne doit être valable pour toutes les instances possibles du problème considéré : on parle de complexité dans le pire des cas.

Les deux principales classes de complexité sont la classe des problèmes décidables en temps polynomial sur une machine de Turing déterministe (*classe P*) et la classe des problèmes Turing non déterministe (*classe NP*). Parmi les problèmes de la classe *NP*, certains problèmes sont plus difficiles que d'autres : ce sont les problèmes *NP-complets*. Plus formellement un problème est dit *NP-complet* s'il est dans la classe NP et si tout problème *NP* se réduit polynomialement à lui. Si un problème n'est pas dans la classe NP mais que tout problème *NP* se réduit polynomialement à lui, on dit qu'il est un problème *NP-difficile*. La question de savoir si les classes P et NP sont effectivement distinctes ($P \neq NP$) est l'une des conjectures majeures de la théorie de la complexité.

Parmi les problèmes sous contraintes nous allons étudier le problème de satisfaction (SAT), et le problème d'affectation quadratique (QAP).

Nous avons déjà mentionné le problème SAT. Plus formellement, une instance du problème SAT est définie par un ensemble des variables Booléennes $\mathcal{X} = \{x_1, ..., x_n\}$ et une formule Booléenne $\mathcal{F}$: $\{0,1\}^n \rightarrow \{0,1\}$. La formule est dite satisfiable s'il existe une affectation v: $\mathcal{X} \rightarrow \{0,1\}^n$ qui satisfasse $\mathcal{F}$, ou insatisfiable sinon. Les instances sont

souvent formulées comme conjonctions de clauses où tous les clauses doivent être satisfaites pour résoudre l'instance.

SAT est peut-être le problème combinatoire plus connu, car il a été le premier problème à être prouvé *NP-complet* [Cook, 1971] et donc il a été amplement utilisé pour coder et résoudre des problèmes issus de divers domaines.

Un autre problème combinatoire bien connu est le problème d'affectation quadratique (QAP), qui a été largement étudié au cours des cinquante dernières années. Il peut s'exprimer de la forme suivante. Considérons deux matrices $A = (a_{ij})_{n\times n}$, $B = (b_{kl})_{n\times n}$, et une fonction de mapping Π. Le but est de trouver la permutation $p_i = (\pi(1), \pi(2), \ldots, \pi(n))$ qui minimise :

$$f(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)}$$

Ce problème a été formulé par Koopmans et Beckmann [Koopmans and Beckmann, 1957] pour résoudre un problème de affectation d'équipements, dans lequel un ensemble de n équipements avec des flux physiques entre eux (matrice A) doivent être affectés dans n endroits séparés par des distances connues (matrice B). Le but est de minimiser le $flux \times distance$ de tout le système. Ce problème a été prouvé *NP-difficile* [Sahni and González, 1976], et parfois consideré du plus difficile, du au fait que le problème du voyageur de commerce (TSP) en est un cas particulier. D'autre part, il présente une forte importance pratique due à sa capacité à modéliser de nombreuses situations issues du domaine industriel.

1.5 Conclusion du chapitre

Les AEs sont des algorithmes de recherche inspirés des principes de l'évolution naturelle. Depuis leurs débuts dans les années 60 ils sont un sujet d'étude important dans le cadre de l'intelligence artificielle. Plusieurs branches existent, différenciées notamment par le codage qu'ils utilisent.

La généralité de leur formulation a permis de résoudre des problèmes dans de nombreux domaines d'application, là où les outils traditionnels de l'optimisation mathématique sont inefficaces.

L'équilibre correct entre exploration et exploitation est une des clés de la bonne performance des AEs. Cela implique un bon réglage des paramètres qui régissent les différents choix du design et du fonctionnement de l'algorithme. Ce travail a pour but l'étude de l'influence des paramètres des AEs sur leur performance, ainsi que les façons de bien contrôler ces paramètres de manière générique.

Chapitre 2

Paramétrage des Algorithmes Évolutionnaires

Sommaire

2.1 L'importance du contrôle

2.1.1 L'équilibre entre l'exploration et l'exploitation

Nous avons déjà introduit les concepts d'*exploration* et d'*exploitation* (EvE), et nous avons souligné l'importance de maintenir un équilibre entre ces deux forces afin d'éviter de tomber dans une *convergence prématurée*, ou de "flâner" sans jamais atteindre une solution de bonne qualité (voir section 1.3.7).

En réfléchissant sur l'importance de maintenir cet équilibre, on réalise que la situation est en fait plus complexe qu'il n'y paraît. Bien qu'il faille éviter les extrêmes de l'exploration et l'exploitation, il y a des circonstances où un peu plus d'exploration ou exploitation s'avère bénéfique. Des algorithmes de recherche tels que le *recuit simulé* [Aarts and Korst, 1989], sont construits de telle façon que l'exploration laisse place à l'exploitation au fur et à mesure que la recherche avance. Ce principe est aussi valable pour d'autres algorithmes de recherche, y compris les AEs. L'idée est de parcourir légèrement la majeure partie de l'espace de recherche au début, afin d'identifier les zones les plus prometteuses, pour se concentrer exclusivement sur celles-ci vers la fin de l'exécution.

Néanmoins, il existe d'autres nuances à prendre en compte pour régler l'EvE. La plupart de temps il n'est pas possible de savoir si on est en train d'atteindre un optimum global quand les individus convergent vers une zone particulière. Si, par contre, l'optimum n'est que local, il est souhaitable que l'AE mette en place des moyens pour en sortir et progresser vers une autre zone.

Un autre problème est de choisir une mesure pour déterminer quand l'équilibre d'EvE penche trop d'un côté ou de l'autre. Des problèmes différents requièrent différents niveaux d'EvE pour être convenablement résolus.

2.1.2 Types de paramètres

Le réglage des paramètres joue un rôle prépondérant dans l'établissement d'un niveau correcte d'EvE [Lobo *et al.*, 2007]. On peut distinguer plusieurs catégories de paramètres, en fonction du composant de l'AE avec lequel ils sont en rapport. Chacun d'eux peut affecter l'EvE et, plus généralement, la performance de l'AE à sa façon. Dans la suite, nous présentons une liste des paramètres que l'on peut trouver dans un AE.

Codage Les paramètres sont alors par exemple le nombre de bits pour répresenter une variable réelle dans un codage binaire, ce qui a un effet sur la taille de l'espace de recherche, mais aussi sur la précision de la solution.

Taille de la population La taille de la population a une grande influence sur la capacité de l'AE à explorer l'espace de recherche. Une petite population aura plus de mal à parcourir un grand espace de recherche et convergera plus facilement vers des optimums locaux qu'une grande population (bien sûr, cela dépend des opérateurs mis en place). D'un autre côté, une grande population est plus coûteuse à maintenir (plus d'évaluations, une convergence plus lente, etc.).

Fonction d'évaluation Bien que la fonction d'évaluation découle de la fonction objectif du problème à résoudre, il n'est pas rare d'inclure d'autres composants afin de

faciliter la découverte de solutions de bonne qualité. Pour les problèmes de satisfaction de contraintes, ou SAT (voir section 1.4.1) on peut inclure des poids (donc paramètres) dans les clauses (contraintes) afin de donner plus d'importance à celles qui semblent les plus dures à satisfaire. Dans le cadre de l'évaluation, on peut aussi envisager d'encourager les individus diférents en augmentant leur évaluation par un facteur réglé par un paramètre (voir niching en section 1.3.5).

Sélection et Remplacement la sélection et le remplacement sont intimement liés à la fonction d'évaluation utilisée, car c'est souvent sur cette façon d'évaluer que les individus sont choisis. Selon que le réglage des poids ou des facteurs mentionnés dans le point précédent sont faits dans l'évaluation ou la sélection (i.e., si les poids et facteurs sont inclus dans l'évaluation de l'individu ou bien considérés uniquement dans la sélection), on peut dire que ces paramètres sont en rapport avec l'évaluation ou la sélection. En outre, on peut trouver des paramètres pour régler la pression de sélection, ou pour faire une différence plus ou moins importante entre individus de qualités différentes.

Opérateurs Les paramètres associés à la probabilité d'application des opérateurs ont des effets différents selon l'opérateur qui s'applique. Parmi les différents types d'opérateurs on trouve :

Mutation Cet opérateur est typiquement dédié à l'exploration. La mutation est vue ici comme la modification des individus d'une façon aléatoire. Trop peu de mutation peut empêcher les individus d'explorer de nouvelles zones de l'espace de recherche, tandis que trop de mutation peut occasionner une destruction de l'information utile contenue dans les individus.

Croisement Le croisement consiste à mélanger l'information provenant de deux individus, appelés *parents*, pour former de nouveaux individus, appelés *fils*. Au début de la recherche, quand les individus sont plutôt différents, le croisement a un rôle d'exploration, car la combinaison des individus produit des individus placés aux endroits non encore visités pendant la recherche. À la fin, par contre, les parents sont similaires, ce qui produit des fils similaires, accordant au croisement un rôle plutôt d'exploitation. Cette dualité de rôle rend difficile le réglage du croisement, mais son effet est adourci dû le caractère auto-adaptatif de cet opérateur [De Jong, 2007].

Opérateurs spécialisés Outre les opérateurs déjà mentionnés, il y a une quantité innombrable d'opérateurs qui peuvent être intégrés à un AE, notamment dans le cas des algorithmes mémétiques, où des opérateurs de recherche locale sont ajoutés afin d'améliorer les individus. On parle en général d'opérateurs *asexués*, quand il n'y a qu'un parent, et *sexués* losqu'il y en a plus d'un. Dans beaucoup de cas, la création d'opérateurs *ad-hoc* est nécessaire car un AE standard ne peut pas résoudre les problèmes de façon optimale. Ajouter ces opérateurs signifie, en même temps, l'insertion des nouveaux paramètres qui règlent leurs probabilités d'application. L'effet sur l'EvE dépend, naturellement, des particularités de chaque opérateur.

Populations multiples Dans le cas où on utilise un AE parallèle (i.e., avec plus d'une population), des nouveaux paramètres apparaissent, tels que le nombre de populations ou le taux de migration des individus entre elles.

Sélection de composants Par sélection de composants on entend les choix de conception qui permettent d'opter pour une façon de réaliser une opération (sélection, codification, opérateurs, etc.) par rapport à une autre. Bien qu'il n'existe pas de consensus pour accepter les choix de conception en tant que paramètres, on peut les considérer comme tels dans la mesure où ils peuvent se coder et se régler. Pour mieux expliquer ce raisonnement, considérons un AE dans lequel on a deux choix de croisements. En pratique, il n'y a aucune différence entre considérer un paramètre de conception qui choisit entre les deux, ou bien avoir deux paramètres de probabilité d'application et fixer l'un à la valeur 0 et l'autre à la valeur 1. Ces choix de conception sont partout présents dans les AEs : la façon dont on code les solutions, les opérateurs à inclure, la méthode de sélection, la topologie des populations parallèles, etc.

Paramètres spéciaux Finalement, on trouve des paramètres qui ne sont pas en rapport direct avec l'AE sinon plutôt avec l'implémentation des opérateurs spécialisés. Leur réglage peut avoir une forte influence sur l'EvE, selon le rôle que les opérateurs correspondants jouent dans l'algorithme.

Il existe des paramètres dont le contrôle est plus évident. Typiquement, les probabilités d'application des opérateurs ont été les premiers paramètres à être contrôlés. Ces paramètres, ici appelés *comportementaux*, sont souvent réglés en modifiant une variable, mais laissant le reste de la structure sans modification.

Dans l'autre extrême se trouvent les paramètres ici appelés *structurels*, qui affectent l'AE d'une façon plus radicale, en utilisant des méthodes alternatives qui peuvent modifier les algorithmes jusqu'à les rendre méconnaissables. Considérons par exemple un AE avec représentation réelle, et opérateurs de mutation et de croisement, où il existent des paramètres qui règlent la taille de la population, le replacement et les opérateurs à appliquer. Si on détermine que la taille de la population est égale à 1, le croisement est appliqué avec une probabilité 0 (c'est à dire jamais), et le replacement de type plus (voir section 1.3.6) en n'acceptant que les fils d'une qualité supérieure à celle de leurs parents, l'algorithme résultant sera plus ressemblant à une méthode de type *descente stricte* qu'à un AE.

Il est difficile de dire quand un algorithme se transforme à cause de la modification de ses paramètres. D'ailleurs, l'étude de ce phénomène n'est probablement pas importante. Néanmoins, cela nous permet de réaliser que, dans une perspective vaste, le champ d'action du réglage de paramètres dépasse le cadre des AEs : le réglage des paramètres consiste en fait à trouver la meilleure option dans l'espace de recherche des algorithmes.

2.1.3 Difficulté lors de la paramétrisation des AEs

Chaque exécution d'un AE peut être vue comme la conjugaison de quatre éléments : l'instance du problème à résoudre, l'AE (y compris codage, opérateurs, paramétrage, etc.),

la population initiale, et les aléas (notamment ceux apportées par les générateurs pseudo aléatoires souvent utilisés).

Tout cela rend chaque exécution d'un AE unique et donc, dans une certaine mesure, imprévisible, notamment si on veut considérer le contrôle d'une façon générale, comme c'est le cas ici. Le réglage et le contrôle de paramètres sont difficiles à faire, au moins pour les raisons suivantes :

- Un bon réglage dépend du problème traité : un paramétrage efficace pour un problème n'est pas nécessairement optimal pour un autre.
- Puisque l'état courant de la recherche (i.e., le placement des individus sur l'espace de recherche et l'état des variables de la méthode) varie au cours du temps, les valeurs optimales des paramètres changent. Si on cherche une paramétrisation optimale à chaque instant, il ne faut donc pas trouver une valeur, mais plusieurs.
- Quand les paramètres sont liés à des opérateurs inconnus, l'effet de leurs applications sur les individus est également imprévisible.
- Il existe une interaction complexe entre les paramètres : il peut arriver que deux opérateurs ne marchent pas bien indépendamment, mais qu'ensemble ils produisent de bons résultats, ou l'inverse.

Afin de surmonter ces difficultés, il faut concevoir l'AE en tant qu'entité abstraite, n'en retenant que les aspects les plus généraux. On discutera plus amplement cette idée dans la section 2.4.

2.2 Taxonomie du régage de paramètres

Selon différents critères, on peut classifier les techniques de réglage de paramètres de façons différentes. Les classifications les plus courantes sont compilées dans un article publié en 1999 [Eiben *et al.*, 1999], puis étendu en 2007 [Eiben *et al.*, 2007]. Ici quatre classifications sont présentées.

- Selon le paramètre réglé
- Selon la manière dont le réglage est effectué
- Selon les critères pris en compte
- Selon la portée du réglage

Ces classifications sont détaillées dans les sous-sections suivantes.

2.2.1 Selon le paramètre réglé

Cette première classification est en rapport avec le type de paramètre qui est réglé. Ici "type" signifie la partie de l'AE avec laquelle le paramètre est en rapport. Parmi ces paramètres on peut trouver ceux qui définissent les aspects suivants :

- Le codage des individus
- La fonction d'évaluation
- L'application des opérateurs et leur fonctionnement
- La sélection et le replacement
- La population

Cette classification est donc en lien avec la liste de types de paramètres présentée dans la sous-section 2.1.2.

2.2.2 Selon la manière dont le réglage est effectué

Cette classification est de loin la plus citée dans les articles sur le contrôle de paramètres. La figure 2.1 présente la classification issue de [Eiben *et al.*, 1999], qui divise le réglage en deux ensembles principaux. On dit que les paramètres sont *ajustés* s'ils sont fixés avant l'exécution de l'algorithme, et restent inchangés pendant la recherche. Si, par contre, les valeurs des paramètres sont modifiées durant l'exécution, on parle de *contrôle de paramètres*.

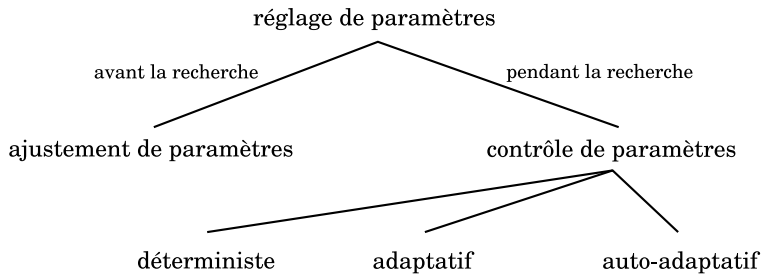

FIG. 2.1 – Taxonomie selon la manière dont le réglage est effectué

Le contrôle peut être encore divisé en trois sous-ensembles. S'il est effectué selon une règle déterministe qui ne prend pas en compte l'état de la recherche, on parle de contrôle déterministe. Ce type de contrôle est typiquement réalisé en fonction du nombre de cycles (générations) écoulés, par exemple, au moyen d'un processus qui rend de plus en plus difficile la réinsertion des individus avec une qualité insuffisante, au fur et à mesure que la recherche avance. Ceci est l'approche la plus simple du contrôle de paramètres, et reste approximative dans le sens où une synchronisation correcte est difficile à trouver.

Le contrôle *adaptatif* prend en compte l'état courant de la recherche pour modifier la valeur des paramètres. Ce contrôle est fait typiquement au moyen de règles empiriques. Un exemple classique est la *"1/5 success rule"* de Rechenberg [Rechenberg, 1973], où l'on attend qu'en moyenne une application sur cinq de l'opérateur de mutation soit bénéfique. De cette façon, si le rapport est supérieur à 1/5, la diversité est augmentée, et s'il est inférieur, la diversité est réduite. Le contrôle adaptatif est un des schémas les plus utilisés, car il permet d'intégrer facilement les connaissances du concepteur sur le fonctionnement de l'algorithme.

Finalement, il existe une dernière façon d'effectuer le contrôle, qui donne à l'algorithme une liberté plus grande de s'auto-contrôler. Dans le *contrôle auto-adaptatif*, les paramètres sont inclus dans le codage de l'individu, pour qu'ils s'adaptent pendant l'évolution. L'idée sous-jacente de cette approche est que les bons individus sont, en partie, le produit des bons paramètres, et donc qu'il faut les récompenser en encourageant leur survie. Cette approche est cependant plus délicate à implémenter, car il peut arriver que les paramètres modifient la manière dont la fonction d'évaluation prend en compte les individus, et que

les meilleures solutions du problème ne soient pas toujours récompensées [Eiben *et al.*, 2000].

2.2.3 Selon les critères pris en compte

En fonction de l'information prise en compte pour régler les paramètres, on peut distinguer deux possibilités générales. La première, dénommée *évidence absolue*, compare une mesure représentative l'état de l'algorithme à une mesure fixe. Par exemple, on pourrait comparer la mesure de diversité [Ursem, 2002] ou la performance d'un opérateur [Rechenberg, 1973] avec un seuil, et modifier quelques paramètres si cette quantité est dépassée.

D'autre part, on pourrait décider la modification de la valeur d'un paramètre en accord avec la *comparaison* entre composants de l'AE. Considérons par exemple, un AE avec plusieurs opérateurs, dans lequel on peut mesurer la qualité générée par chaque opérateur afin d'encourager l'application des meilleurs [Davis, 1989]. Dans ce cas, il faut absolument établir une base de comparaison entre composants pour distinguer quel opérateur a la meilleure performance.

2.2.4 Selon la portée du réglage

Un dernier critère pour classifier le réglage de paramètres est la portée du réglage. Ici, Angeline [Angeline, 1995] reconnaît trois niveaux : *population*, *individu* et *composant* (sous-individu). Cette classification est proche de celle qui considère le type de paramètres, car la portée du réglage dépend en fait du paramètre réglé.

Les réglages au niveau de population sont effectués lorsque les paramètres ont un effet global sur la population, par exemple, pour modifier la probabilité d'application d'une mutation, ou bien la pression de sélection. Ces paramètres, communs à toute la population, ont donc un effet global.

Les réglages au niveau de l'individu sont souvent en rapport avec la position qu'a l'individu dans l'espace de recherche. Par exemple, on pourrait définir un opérateur de mutation en rapport avec la rugosité du paysage de recherche : si l'individu se trouve dans une zone relativement "douce", on peut permettre que la mutation soit plus radicale, tandis que pour des zones plus rugueuses, on souhaite progresser plus doucement, afin de minimiser le risque de rater une bonne valeur dans le voisinage.

Finalement, on pourrait envisager des paramètres ayant une action locale à l'intérieur de l'individu. Considérons par exemple, dans un cadre auto-adaptatif, un individu qui code, en plus des variables du problème, des probabilités de mutation pour chacune des variables codées, afin de ne pas trop modifier les variables reconnues comme bonnes, et concentrer l'exploration sur le reste des variables.

On pourrait ajouter dans cette classification un niveau supplémentaire, si on considère une voie de recherche relativement nouvelle : les *hyperheuristiques* [Burke *et al.*, 2003], définies comme "heuristiques pour choisir des heuristiques". Dans cette approche, le réglage ne porte pas sur les paramètres *à l'intérieur* d'un algorithme, mais plutôt sur l'algorithme –parmi plusieurs– qu'il faut appliquer pour résoudre un certain problème.

Une approche similaire peut être observé dans les solveurs de type portfolio, comme *SATzilla* [Xu *et al.*, 2008].

2.3 Outils pour le contrôle

2.3.1 Apprentissage

Comme ous avons vu dans la sous-section 2.2.2, le contrôle peut être effectué de trois manières. On oubliera pour l'instant la plus simple d'entre elles (déterministe) pour se concentrer sur les deux autres. Le contrôle auto-adaptatif a l'avantage d'être plus flexible que le contrôle adaptatif, car il n'y a aucune règle établie pour guider le réglage des paramètres, donc le réglage *évolue* vraiment au cours de l'exécution. Néanmoins, l'inclusion des paramètres dans le codage de l'individu augmente la taille de l'espace de recherche, et ne prend pas en compte les connaissances des experts. D'autre part, le contrôle adaptatif est comparativement plus simpliste et n'est pas capable de s'adapter aux circonstances particulières que rencontre l'algorithme.

Afin de pallier les défauts du contrôle adaptatif, une composante d'*apprentissage* peut être incluse dans le mécanisme de contrôle. Cela apporte au contrôle adaptatif une capacité d'adaptation, sans perdre la possibilité d'inclure des connaissances expertes (idéalement exprimées d'une façon générale) ni augmenter la taille de l'espace de recherche.

[Lis, 1996] propose un schéma de contrôle de paramètres basé sur la comparaison de performances dans plusieurs populations indépendantes. Chaque population a initialement une valeur différente pour les paramètres, et ils sont modifiés de telle façon qu'il y a toujours des populations avec des valeurs supérieures et inférieures à l'optimum actuel, afin de réaliser une exploration de l'espace des paramètres.

L'utilisation des techniques d'apprentissage a été appliquée principalement lors du contrôle des probabilités d'application d'opérateurs, où les plus performants sont récompensés en augmentant cette probabilité [Julstrom, 1995; Davis, 1989]. [Lobo and Goldberg, 1997] présente une approche similaire pour des algorithmes génétiques hybrides, en considérant les opérateurs comme des bandits manchots qui donnent une récompense chaque fois qu'ils sont utilisés. Dans [Igel and Kreutz, 2001], un schéma similaire est appliqué sur un problème de topologies de réseaux neuronaux, où les récompenses sont basées sur la fonction d'évaluation. [Kee *et al.*, 2001] propose deux méthodes qui incluent une phase d'apprentissage pendant laquelle différentes combinaisons des paramètres sont essayées ; le comportement de l'AE est codé dans des tables ou des règles, qui seront ensuite utilisées pour contrôler l'algorithme. [Wong *et al.*, 2003] propose un algorithme divisé en périodes d'apprentissage et de contrôle des paramètres, en ajustant leurs valeurs centrales et limites. Dans [Eiben *et al.*, 2004], la taille de la population est modifiée selon plusieurs critères basés sur l'amélioration de la qualité historique.

[Thierens, 2005; Thierens, 2007] présente un algorithme qui choisit, parmi un ensemble d'opérateurs, celui qui sera appliqué, selon la probabilité de chacun. Cette probabilité est adaptée en fonction du succès des applications passées. Dans le but d'accélérer la vitesse d'aprenstissage du contrôleur, une méthode de *pursuit* est proposée, et comparée avec la *probabilité proportionnelle* utilisée lors de travaux antérieurs.

Dans [Eiben *et al.*, 2006], une approche basée sur la méthode REVAC [Nannen and Eiben, 2007] distingue trois niveaux : la couche d'application (problème à résoudre), celle de l'algorithme (correspondant à l'AE), et celle du contrôle. Trois paramètres sont contrôlés, et l'apprentissage est fait en mélangeant deux algorithmes d'*apprenstissage par renforcement* : Q-learning et SARSA.

[Whitacre *et al.*, 2006] contrôle les paramètres sur la base de la performance observée, au moyen de quelques mesures statistiques (telles que la fonction d'évaluation, si les fils survivent ou pas, le rang de la solution dans la population, etc.). Il considère ces mesures de deux façons : soit en prenant la moyenne, soit en prenant les valeurs extrêmes (*outliers*). Il conclue que le contrôle marche mieux avec les outliers, car le cas contraire ne bénéficie qu'aux bons opérateurs à court terme.

[Petrovic and Epstein, 2006] propose une méthode pour choisir des heuristiques afin de résoudre un problème de satisfaction de contraintes. L'algorithme apprend une combinaison d'heuristiques qui donne les meilleurs résultats.

Dans la plupart des cas, les récompenses accordées aux opérateurs sont basées sur l'amélioration de la fonction d'évaluation. Dans nos travaux, nous avons décidé d'inclure aussi d'autres considérations pour évaluer la performance, telles que la capacité à maintenir la diversité dans la population et le temps d'exécution des opérateurs. Nous verrons cela plus en détail dans la section 4.

2.3.2 Apprentissage flou

Une autre manière d'implémenter l'apprentissage dans le contrôle des paramètres est d'utiliser de contrôlleurs flous. La logique floue (LF) [Kulkarni, 2001; Piegat, 2001] est une extension de la logique Booléenne. Dans la logique Booléenne, une formule est *vraie* ou *fausse*, tandis que la LF admet un nombre infini de nuances de vérité exprimées par une fonction d'*appartenance*, qui est dans l'intervalle $[0, 1]$, où 0 et 1 signifient respectivement faux et vrai. La LF peut mieux exprimer des concepts imprécis tels que "froid", "loin" ou "rapide".

Une des applications les plus utiles de la LF sont les *Contrôleurs Flous* (CF) [Kulkarni, 2001; Piegat, 2001; Passino and Yurkovich, 1998]. Les CFs permettent d'inférer des réponses à partir de règles telles que *"SI vélocité_de_la_voiture est HAUTE ET état_de_la_route est SEC, ALORS le risque_d'accident est MOYEN"*. La figure 2.2 montre la structure générale d'un CF simple, connu sous le nom de *Mamdani*. Le premier pas consiste à transformer une entrée exprimée par une quantité réelle vers son expression floue correspondante (valeur réelle entre 0 et 1). Puis, un moteur d'inférence obtient la sortie floue à l'aide d'un ensemble de règles floues. Finalement, la sortie floue est traduite en utilisant un défuzzificateur.

Les CFs ont été utilisés pour contrôler des AEs [Cordon *et al.*, 2004], typiquement en prenant des statistiques exprimant l'état de la recherche comme entrées du CF et les valeurs des paramètres contrôlés comme sorties. Par exemple, [Herrera and Lozano, 1996] présente un AE contrôlé par CF (FAGA) où les entrées sont la diversité phénotypique et génotypique, et les sorties sont la probabilité de mutation ainsi qu'un paramètre pour contrôler la pression de sélection. [Lee and Takagi, 1993] propose un FAGA dans lequel

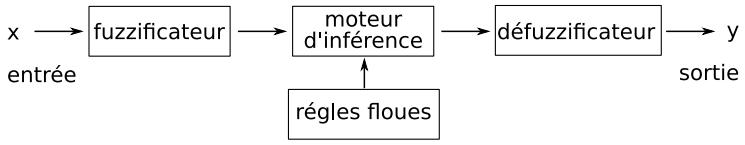

FIG. 2.2 – Schéma général d'un Contrôleur Flou (CF)

les entrées pour le CF sont trois mesures en rapport avec la fonction d'évaluation, et les sorties sont les probabilités de mutation, de croisement et un paramètre pour ajuster la taille de la population. [Liu and Lampinen, 2005] présente un algorithme d'évolution différentielle contrôlé par un CF. [Subbu and Bonissone, 2003] présente une application industrielle (design des circuits imprimés) au moyen d'un AE contrôlé par un CF qui règle les probabilités de mutation et croisement et où les entrées sont deux mesures de diversité.

Finalement, [Herrera and Lozano, 2003] présente une étude bibliographique des FA-GAs, ainsi qu'une classification selon les deux critères suivants :

Selon la manière dont les règles sont obtenues : ici on distingue trois méthodes. La première utilise la connaissance des *experts* (humains), la deuxième est *offline*, c'est-à-dire générée automatiquement à partir d'un ensemble de problèmes standards, et la troisième est *online*, i.e., automatiquement générée à partir du problème qui est en train d'être résolu.

Selon la portée du réglage : ce critère est analogue à la classification présentée en 2.2.4, où l'on distingue deux niveaux : *population* et *individu*.

Il existe plusieurs variations du CF "standard" décrit plus haut. Un type particulier de CF utilisé dans cette thèse est le *Takagi-Sugeno* [Takagi and Sugeno, 1985]. Dans ce contrôleur, le moteur d'inférence a comme sortie un nombre réel à la place d'une variable floue. Cette valeur est exprimée en fonction des valeurs d'entrée, la déffuzification n'étant plus nécessaire. Ce contrôleur fournit des contrôles plus fins, mais plus difficilement lisibles par un utilisateur.

Puisque les CFs sont des approximateurs universels des fonctions continues [Buckley, 1993], ils peuvent aussi être utilisés comme outils pour modéliser une fonction à partir de données expérimentales [Chiu, 1997; Piegat, 2001]. Une approche originale concernant l'utilisation des CFs pour cette tâche a été proposée par Wang et Mendel [Wang and Mendel, 1992]. Dans ce travail, une fonction $y = f(x_1, x_2)$ est modélisée par un contrôleur de type Mamdani à partir des données expérimentales sous la forme de triplets (x_1, x_2, y), en divisant l'espace défini par les variables d'entrée grâce à une grille, puis en trouvant la valeur caractéristique de y pour chaque boîte. De nombreuses méthodes sont basées sur cette approche.

Il existe de nombreux études visant à l'obtention des CFs de bonne qualité. [Riid and Rüstern, 2004] présente une technique heuristique pour positioner les partitions floues dans le domaine des variables, à partir de la mesure de l'erreur dans la sortie du CF. [Nozaki *et al.*, 1997] propose une méthode pour obtenir rapidement des règles floues pour un CF Takagi-Sugeno en minimisant la somme des carrés des exemples. [Setnes *et al.*, 1998] propose une méthode pour obtenir des CFs exacts sans perdre de

vue leur interprétabilité, en obtenant un CF Takagi-Sugeno qui est ensuite réduit vers un CF de type Mamdani. Les études présentées dans [Costa-Branco and Dente, 2001; Costa-Branco and Dente, 1999] abordent des aspects peu traités dans les FAGAs, tels que l'effet du bruit sur la modélisation, la taille de l'ensemble des échantillons d'exemples, la capacité de généralisation des CFs et le traitement des règles pour lesquelles il n'y a pas de données expérimentales.

L'utilisation des CFs en tant qu'outils de modélisation permet un nouvel usage de la LF au service des AEs. Dans notre travail, nous avons utilisé des CFs, pas réellement en tant que contrôleurs, mais comme un outil d'apprentissage. Nous présentons les détails dans la section 3.2.2

2.3.3 Autres approches

Il existe d'autres approches pour régler les paramètres des AEs. Cette sous-section en recense quelques unes.

[Ursem, 2002] propose une idée simple pour contrôler un AE en fonction de la diversité de la population. L'algorithme oscille entre deux phases, l'une d'exploration, qui encourage la mutation, et l'autre d'exploitation, qui encourage la sélection et la recombination. Le changement entre phases se fait lorsque des seuils de diversité sont atteints. Cela permet une alternance entre les deux phases.

[Nannen and Eiben, 2007] présente une méthode appelée *REVAC*, qui permet, grâce à plusieurs exécutions de l'AE, de ne pas uniquement régler les paramètres, mais de connaître aussi l'importance de chacun.

[Yuan and Gallagher, 2007] présente des techniques d'ajustement basées sur la comparaison de plusieurs exécutions du même algorithme avec différents paramètres. Des tests statistiques sont faits avec les résultats partiels afin d'éliminer les configurations les moins bonnes, en économisant ainsi du temps de calcul. Une approche similaire peut être trouvée dans [Birattari, 2004].

Quelques travaux cherchent à éliminer tous les paramètres des AEs. Outre les premiers efforts tendant à trouver des valeurs optimales universelles pour des algorithmes génétiques [De Jong, 1975], on peut trouver le *parameter-less GA* [Harik and Lobo, 1999] dans lequel les paramètres correspondant aux taux de sélection et probabilité de croisement ainsi que la taille de la population sont éliminés. Les deux premiers par ajustement et le troisième en commençant par une petite population, laquelle est dupliquée et réinitialisée chaque fois qu'elle converge, jusqu'à la limite de la mémoire de l'ordinateur. La probabilité de mutation est simplement ignorée.

[Lima and Lobo, 2004] propose un cadre composé de deux algorithmes différents, qui s'exécutent alternativement pour profiter des avantages de chacun d'eux. On utilise alors une combinaison de *ECGA* (Extended Compact Genetic Algorithm), qui se focalise sur la recherche des distributions et *ILS* (Iterated Local Search), une méthode de recherche locale qui perturbe l'individu, puis effectue une recherche locale et finalement décide si le nouvel individu remplace l'ancien selon un critère d'acceptation.

2.4 Généralité du contrôle

Malgré leur versatilité, les AEs restent encore du domaine des spécialistes. Les AEs ont, en fait, beaucoup de paramètres à régler, ce qui les rend difficile pour l'utilisateur lambda. D'ailleurs, la littérature sur ce sujet se focalise sur l'étude des paramètres spécifiques qui appartiennent à un AE particulier, qui résout un problème particulier. À l'exception des stratégies d'évolution [Beyer and Schwefel, 2002], lorsque les AEs sont appliqués aux problèmes du monde réel, l'ajustement des paramètres (le contrôle est quasi inexistant) est fait au moyen de méthodes rudimentaires, souvent par des séries d'essais et erreurs très coûteuses en temps de calcul. On manque en fait de critères de haut niveau pour concevoir des stratégies de contrôle. Il serait donc intéressant de proposer des méthodes de réglage utilisables par des non-spécialistes du domaine.

Dans ce contexte, il est important de rappeler le principe du *No free lunch* (NFL). Ce principe, introduit par Wolpert et Macready [Wolpert and Macready, 1997], déclare qu'il n'existe pas de méthode qui soit globalement plus performante sur un ensemble varié de problèmes, par rapport à d'autres. Ceci est valable pour tous les algorithmes, ce qui nous laisse uniquement deux alternatives au moment de résoudre des problèmes variés : soit on maintient un ensemble d'algorithmes pour chaque cas, soit on maintient un algorithme capable de se métamorphoser en plusieurs, comme expliqué dans la section 2.1.2.

Cette thèse propose de créer une abstraction des AEs afin de rendre le contrôle plus simple et intuitif. L'importance d'une telle abstraction est liée à un facteur humain : une bonne abstraction est beaucoup plus facile à saisir qu'un ensemble de variables de bas niveau entremêlées de façon complexe. Imaginons qu'à la place du volant et des pédales d'une voiture, il nous faille manipuler plusieurs valves, roues et engrenages différentiels pour la conduire, comme c'était le cas dans les premières voitures. Les AEs actuels sont semblables aux voitures lors de leur apparition : une machine complexe qui ne peut être manipulée que par des spécialistes.

2.4.1 Caractéristiques d'un contrôleur générique

Quelles sont donc les caractéristiques souhaitables d'un contrôleur générique ? Parmi toutes les réponses à cette question, nous avons choisi de considérer les aspects suivants :

- Il doit fournir une abstraction des paramètres de bas niveau : il faut laisser l'utilisateur se concentrer sur le *guidage de la recherche* plutôt que sur le *réglage des paramètres*, souvent difficiles à appréhender. Cette abstraction doit être applicable à une grande variété d'algorithmes.
- Il doit être capable de travailler avec des paramètres inconnus, afin de ne pas interdire à l'utilisateur d'ajouter de nouveaux composants. Une méthode qui travaille uniquement avec des paramètres standards est naturellement limitée. Le contrôleur doit fournir à l'usager des informations sur l'effet que les nouveaux paramètres produisent sur la recherche.
- Il doit être facile à intégrer dans la plupart des AEs. Le but d'un utilisateur lambda n'est pas de créer une approche sophistiquée pour régler les paramètres, mais bien de résoudre le problème. Le contrôle doit donc être disponible avec un minimum

d'effort.
– Il doit être le plus autonome possible, afin que l'utilisateur puisse se concentrer sur sa tâche principale, et l'aider à gagner de temps.

2.4.2 Approche pour construire un contrôleur générique

Afin de rendre le contrôleur autonome, un composant d'apprentissage devient alors indispensable, pour appréhender les particularités de l'algorithme et de bien adapter les paramètres. Cet apprentissage et ce contrôle doivent être exprimés en termes communs à tous les AEs, afin de s'abstraire des différences de chacun.

Notre approche pour réaliser cette abstraction consiste à se concentrer sur les deux aspects présentés dans la sous-section 1.3.7, c'est-à-dire, essayer de maximiser l'exploitation et l'exploration, en considérant tout AE comme un problème bi-objectif où les deux forces sont nécessaires. Afin de présenter ce cadre, nous avons divisé les aspects étudiés en trois parties, détaillées dans les chapitres 3 à 5 et évoquées rapidement ici.
– La première partie (chapitre 3) est dédiée à la connaissance des effets des paramètres sur le fonctionnement de l'algorithme au moyen d'un composant qui modèlise cette relation pendant l'exécution de l'AE.
– La deuxième partie (chapitre 4) est dédiée à la construction d'un contrôleur capable d'adapter rapidement les paramètres liés à l'application des opérateurs de manière générale.
– La troisième partie (chapitre 5) est dédiée à l'extension du contrôleur ainsi créé, afin de l'utiliser comme un outil de design, en choisissant automatiquement les opérateurs à inclure dans l'AE.

2.4.3 Paramètres et méta-paramètres

Dans cette étude, on distingue différents niveaux qui séparent le problème, l'AE et le contrôleur. La figure 2.3 montre ces couches et les liens entre elles.

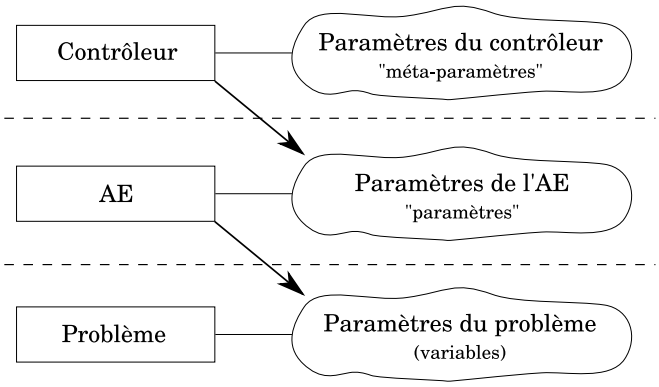

Fig. 2.3 – Différents niveaux de paramétrisation

On trouve d'abord, dans la couche inférieure, le problème à résoudre. Ce problème

consiste à trouver une solution (quasi-)optimale pour l'affectation des variables. On peut considérer les variables du problème comme les paramètres qu'il faut déterminer. Dans la deuxième couche se trouve l'AE, qui agit sur les paramètres du problème et possède, à son tour, des paramètres à régler pour fonctionner correctement (probabilités d'application d'opérateurs, taille de population, etc.). Finalement, dans la couche supérieure, on trouve le contrôleur, dont le but est analogue à celui de l'AE vis à vis du problème, c'est-à-dire, trouver les bons paramètres de l'AE afin qu'il ait une performance (quasi-)optimale. À son tour, le contrôleur possède des paramètres qu'il faut régler pour qu'il marche bien. Dans ce travail, on appellera *paramètres* ceux de l'AE et *méta-paramètres* ceux du contrôleur.

Notons que l'AE, au moins dans sa forme la plus basique, crée une abstraction du problème pour travailler. Quel que soit le problème, l'AE connaît uniquement le codage et une fonction objectif. Peu importe si le problème consiste à affecter des moyens de transport dans une chaîne logistique ou à reconstruire une chaîne d'ADN, l'AE travaille toujours selon les mêmes principes, dictés à partir de son modèle d'évolution artificielle. Un des principaux objectifs de cette thèse est de créer une abstraction de l'AE pour le contrôleur, dans le but d'obtenir un contrôle générique, applicable sur une grande variété d'algorithmes de recherche.

En regardant la figure 2.3 on pourrait se demander quel est l'intérêt d'ajouter des couches supplémentaires, étant donné qu'il y aura toujours des paramètres à régler dans la nouvelle couche. On pourrait se demander aussi quelle est la limite de couches à ajouter. On ne doit pas chercher les réponses à ces questions dans la spéculation pure, mais dans l'utilité pratique des couches supplémentaires. Nous pensons qu'une nouvelle couche d'abstraction (et donc de paramètres) est bénéfique dans les deux cas suivants :

- Quand la recherche est moins sensible aux paramètres de la nouvelle couche que à ceux de la couche inférieure. Considérons, par exemple, le cas de la probabilité de mutation : de petits changements de valeur peuvent avoir des effets drastiques sur la performance de l'AE. Il y a donc un intérêt à utiliser un contrôleur qui fournisse un comportement plus stable, même en incluant des paramètres additionnels.
- Quand les paramètres de la nouvelle couche fournissent une abstraction plus facilement compréhensible des paramètres de la couche inférieure. Au fur et à mesure qu'on monte dans les niveaux des couches, on se rapproche de l'utilisateur humain (qui d'ailleurs sera toujours la dernière "couche" du système), on a donc intérêt à rendre les paramètres de plus en plus appréhendables. Ceci est le cas dans notre approche : il apparaît plus simple pour l'usager de penser à demander plus ou moins d'exploration que d'avoir à modifier une douzaine de paramètres qu'il ne comprend pas toujours très bien.

2.5 Conclusion du chapitre

Ce chapitre a présenté les aspects inhérents au réglage de paramètres, où l'équilibre entre l'exploration et l'exploitation apparaît comme un objectif primordial. Nous avons rappelé les classifications et les différentes perspectives possibles et nous avons présenté les lignes générales des méthodes de contrôle des paramètres. Nous avons souligné les

motivations principales de ce travail, à savoir, la nécessité de créer une abstraction afin de rendre le contrôle plus intuitif, et la prise en compte des aspects autres que la seule amélioration de la fonction d'évaluation.

Avec ce chapitre se termine la première partie de cette thèse, dédiée à l'état de l'art. La partie suivante présentera le travail effectué dans le but d'implémenter le contrôleur générique décrit dans la section 2.4.1, selon l'organisation présentée dans la section 2.4.2.

Deuxième partie

Contrôle Autonome de Paramètres

Chapitre 3

Modélisation pour le contrôle

Sommaire

3.1 Introduction

Ce chapitre présente une méthode permettant d'encapsuler les paramètres d'un AE afin de créer une abstraction qui simplifie le contrôle et la compréhension du fonctionnement interne d'un algorithme. Un modèle issu de la logique floue est utilisé pour apprendre les effets des paramètres sur la recherche. Ainsi, une stratégie de haut niveau peut être définie afin de modifier les valeurs des paramètres et atteindre un niveau prévu d'équilibre entre exploration et exploitation tout le long la recherche. Cette méthode a été présentée dans [Maturana and Saubion, 2007a; Maturana and Saubion, 2007b; Maturana and Saubion, 2008b].

Une idée intuitive de notre approche est résumée sur la figure 3.1. Le contrôleur affecte des valeurs aux paramètres de l'AE. L'AE exécute une génération avec cette configuration et informe le contrôleur sur la diversité et la qualité ainsi produite. Le contrôleur utilise cette information pour modéliser l'influence des paramètres sur la performance de l'AE, et retient les meilleures configurations –pour chaque niveau de diversité et qualité– afin de les utiliser plus tard.

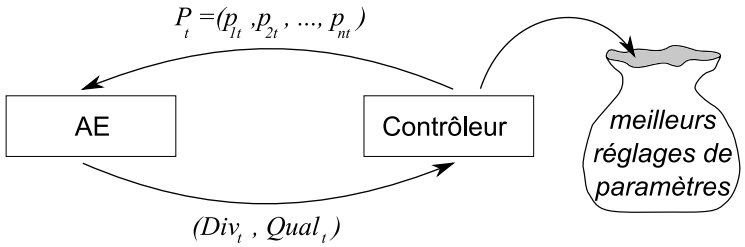

Fig. 3.1 – Schéma général de l'interaction entre le contrôleur et l'AE

Cette méthode simplifie le contrôle grâce à l'abstraction de plusieurs paramètres. Cette abstraction permet de contrôler l'AE avec un paramètre unique qui est en rapport avec un concept de haut niveau : l'équilibre entre l'exploration et l'exploitation (EvE). De cette façon, cet équilibre peut être commandé au moyen d'une stratégie, ce qui est plus facilement compréhensible par l'utilisateur.

Dans ce chapitre, notre motivation principale est de généraliser le processus de contrôle en permettant à l'utilisateur de penser en termes plus généraux et faciliter ainsi la conception de méthodes de contrôle autonomes (voir section 3.3).

Notre méthode se compose de deux phases :

1. *L'apprentissage* est dédié à comprendre comment les paramètres influent sur la recherche et à modéliser ce comportement. Des exemples sont générés pour plusieurs combinaisons de paramètres et un modèle utilisant la logique floue est utilisé pour stocker la connaissance acquise.
2. *Le contrôle* utilise cette connaissance pour guider la recherche. Le modèle construit dans la phase précédente est mis en ouvre, conjointement avec une stratégie de recherche, afin d'ajuster les valeurs des paramètres durant la recherche, en fonction des niveaux d'exploration et d'exploitation requises.

Dans ce contexte, trois aspects importants doivent être considérés : comment collecter les exemples qui seront utilisés pour l'apprentissage, comment construire le modèle et comment concevoir la stratégie permettant de guider la recherche.

Le chapitre est organisé comme suit. La section 3.2 présente notre méthode, en discutant les différents aspects mentionnés. La section 3.3 détaille les conditions expérimentales. La section 3.4 discute les résultats, alors que la section 3.5 propose des conclusions et suggestions pour les travaux futurs.

3.2 Abstraction de paramètres

Cette section présente la méthode que nous avons développée pour créer une abstraction du contrôle des paramètres. Cette méthode inclut une collecte initiale des exemples, qui sera expliquée dans la sous-section 3.2.1. Ensuite, les exemples ainsi rassemblés sont utilisés pour construire un modèle (sous-section 3.2.2). Finalement, le modèle est utilisé pour contrôler le déroulement de la recherche en modifiant la valeur d'un paramètre unique, guidé par une stratégie globale de contrôle (sous-section 3.2.3). Afin de fournir une véritable abstraction du point de vue de l'utilisateur, toute la méthode lui est présentée comme une boîte noire, ce qui est expliqué dans la sous-section 3.2.4. En outre, l'utilisateur peut se servir des données collectées pendant l'apprentissage, afin de mieux connaître l'AE. Cet aspect est présenté dans la sous-section 3.2.5.

La méthode se compose de deux phases. La première est dédiée à la compréhension des effets des paramètres sur le comportement de l'algorithme. Nous proposons d'évaluer la performance suivant deux mesures : la diversité et la qualité de la population. Dans ce chapitre, nous utilisons la valeur moyenne de la fonction d'évaluation en tant que mesure de qualité et une mesure de dissimilarité en tant que mesure de diversité. La diversité dépend du codage utilisé (voir la section 3.3.2 pour plus de détails sur la mesure de diversité utilisée), alors que la fonction d'évaluation moyenne pourrait être replacée par d'autres mesures de qualité, comme par exemple la valeur de qualité du meilleur individu ou une mesure de tendance centrale.

La diversité a été choisie comme variable de contrôle car elle est directement en rapport avec l'équilibre de l'exploration et l'exploitation (EvE). D'une part, une faible diversité signifie que les individus de la population sont concentrés plus ou moins dans la même zone de l'espace de recherche, soulignant une phase d'exploitation. D'autre part, une diversité élevée indique que les individus sont répartis dans l'espace de recherche, révélant une phase d'exploration. Bien que cela soit vrai dans la plupart des cas, il faut remarquer qu'il existe des exceptions, par exemple un AE qui résout un problème multimodal où les optimums sont dispersés, en utilisant une population de faible taille, peut avoir une diversité élevée et réaliser un maximum d'exploitation. Même si on ne peut savoir si cette situation est commune ou non, nous considérons que la diversité semble être un bon compromis entre une véritable expression d'EvE et une facilité de conception et implémentation.

Une fois que le modèle a été construit, il s'agit de trouver les meilleures combinaisons de paramètres pour chaque niveau de diversité. Ici, deux caractéristiques sont souhaitables : une diversité élevée, afin d'éviter de stagner dans les optimums locaux, et une qualité

élevée. Puisque ces objectifs sont conflictuels, on identifie les combinaisons de paramètres qui produisent les combinaisons de diversité et qualité sur la frontière Pareto (voir section 1.3.4).

Néanmoins, l'obtention du front Pareto ne fait que réduire le nombre de configurations. Un autre critère est nécessaire afin de choisir la configuration qui sera utilisée à chaque moment du processus de recherche, c'est à dire, le niveau de diversité qui sera requis pour l'AE. En effet, la phase d'apprentissage ne fait que construire une abstraction, donc une stratégie doit être définie pour guider la deuxième phase.

La figure 3.2 montre les états principaux du contrôleur, appelé par l'AE à chaque génération. Les états 1 à 5 correspondent à la première phase d'apprentissage, alors que l'état 6 correspond à celle de contrôle.

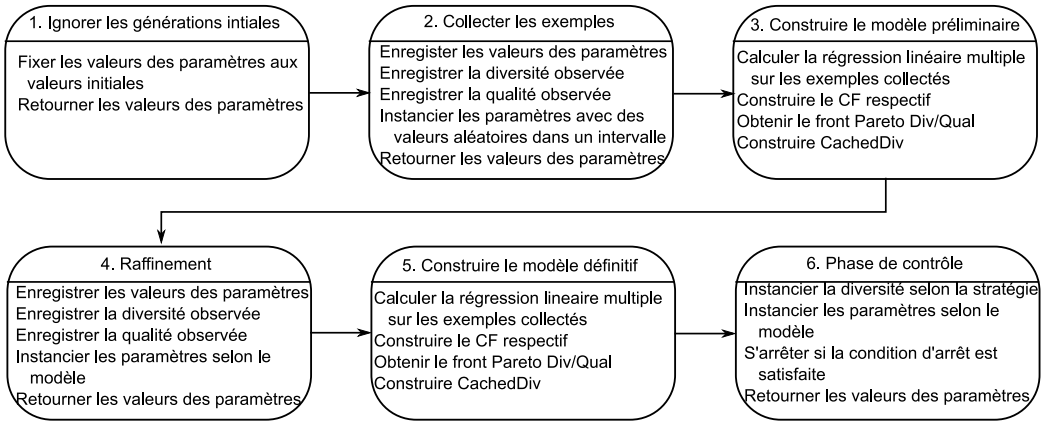

Fig. 3.2 – Principaux états du contrôleur

3.2.1 Apprentissage

Afin de collecter les exemples, l'espace des paramètres est divisé en réalisant des partitions floues pour chaque paramètre. Les intersections des partitions floues sont appelées *secteurs d'influence*. Un de ces secteurs est représenté par un carré aux coins ronds dans la figure 3.3.a. Cette division est ensuite sous-divisée afin d'obtenir une *grille d'entraînement* plus fine (Les motivations de cette démarche seront expliquées plus loin). Ce facteur de division est appelé *finesse*.

La phase d'apprentissage est divisée en cinq sous-phases :

1. *Ignorer les générations initiales* : Afin d'écarter une diversité élevée et une qualité basse, caractéristiques de la population initiale générée aléatoirement, un nombre de générations initiales est ignoré.
2. *Collecte d'exemples* : Les valeurs des paramètres, de la diversité et de la qualité, nécessaires pour la future construction du modèle, sont enregistrées.
3. *Construction du modèle préliminaire* : Les exemples rassemblés préalablement sont utilisés pour construire les contrôleurs flous (CF) de diversité et de qualité.

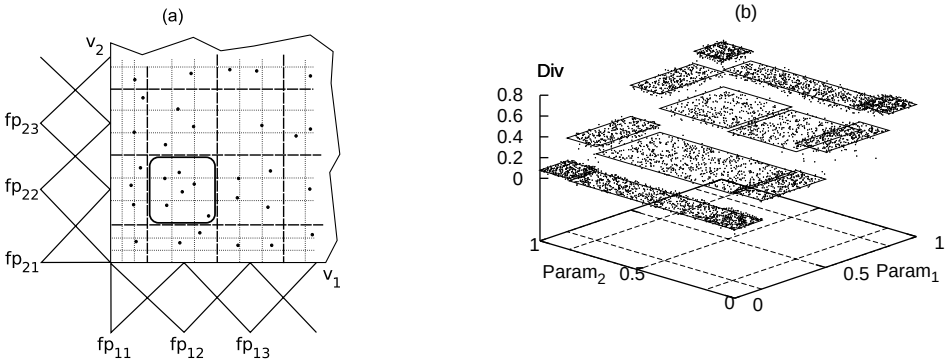

FIG. 3.3 – (a) secteur d'influence (fp_{12},fp_{22}) pour deux dimensions (deux paramètres) et une grille d'entraînement de finesse 3. (b) Formation de plateaux (signalés par des carrées) dans une grille d'entraînement de 4x4

4. *Raffinement* : De nouveaux exemples sont collectés, en se focalisant dans les zones les plus prometteuses de l'espace de recherche des paramètres, afin d'ajuster le modèle.
5. *Construction du modèle définitif* : Tous les exemples sont pris en compte pour construire le modèle définitif, qui sera utilisé pendant la phase de contrôle.

Trois problèmes principaux apparaissent dans cette phase générale d'apprentissage : la *dimensionalité*, l'*inertie* et le *bruit*. La *dimensionalité* est liée au fait que la quantité d'exemples à générer dépend exponentiellement de la quantité de paramètres contrôlés. L'*inertie* est en rapport avec la résistance aux changements des valeurs de diversité et de qualité entre les générations consécutives. Nous comprenons le *bruit* comme la variation à court terme produite par les aléas liés à l'utilisation des opérateurs, ce qui introduit un degré d'inexactitude dans le modèle.

Un symptôme d'inertie peut être observé dans la figure 3.3.b. Ici, un espace de paramètres bidimensionnel est divisé par une grille de 4×4, correspondant à 4 partitions floues par paramètre (la partition est indiquée sur la base du graphique), et l'axe z correspond à la diversité. Tous les exemples à l'intérieur d'un secteur d'influence ont été générés avant de passer au suivant. Bien que la surface soit conçue pour être continue, l'inertie de la diversité aplatit les données à l'intérieur des secteurs d'influence. Puisqu'une partition floue de 4×4 est largement suffisante pour modéliser la surface, il ne s'agit pas d'augmenter le nombre de partitions, mais plutôt de diviser chaque secteur d'influence en parcelles plus petites, d'où la nécessité d'utiliser ce que nous avons appelé *finesse*.

Dans le but d'éviter des changements brusques dans les valeur des paramètres, nous avons défini un parcours spécial de visite, appelé *smooth*, qui passe toujours d'une boîte d'une grille n-dimensionnel à une autre voisine. La figure 3.4 montre des exemples pour 2 et 3 paramètres, par rapport au parcours classique de *boucle imbriquée*.

Afin d'exclure les mesures de diversité et de qualité qui ne sont pas en rapport avec les paramètres, mais avec la population initiale (générée aléatoirement), le contrôleur ignore un nombre de générations au début du processus de recherche. Pour considérer l'effet à long terme de certains opérateurs (par exemple la mutation, dont les effets bénéfiques

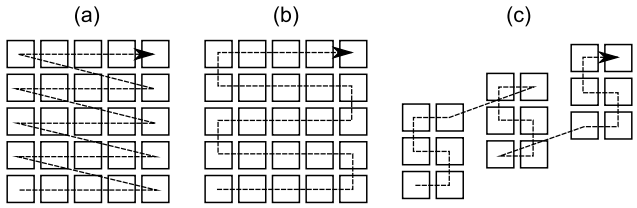

FIG. 3.4 – Parcours de visite : (a) boucle imbriquée, (b) smooth dans 2D, (c) smooth dans 3D

ne sont pas perçus instantanément), la mesure de qualité est corrigée en affectant une moyenne décroissant exponentiellement entre sa propre valeur et celles des générations suivantes comme fait dans [Kee *et al.*, 2001].

Une fois que tous les exemples ont été collectés, le contrôleur obtient le modèle des fonctions $Div(\overrightarrow{P})$ et $Qual(\overrightarrow{P})$, qui expriment la relation du vecteur de paramètres $\overrightarrow{P}$ avec la diversité et la qualité, au moyen des CF (voir sous-section suivante).

Avant de passer à la phase de contrôle, une deuxième collecte d'exemples est réalisée, cette fois en se focalisant sur les zones proches de celles les plus prometteuses trouvées après la première collecte d'exemples. Ceci est fait en prenant les valeurs des paramètres correspondant aux points sur le front Pareto de diversité/qualité, avec une erreur distribuée normalement. Finalement, tous les exemples sont pris en compte pour construire le modèle définitif.

3.2.2 Construction du modèle

Dans les CF de Takagi-Sugeno, la sortie est exprimée en fonction des variables d'entrée. Si les CF sont utilisés dans le but de modéliser, il faut inférer cette fonction à partir des exemples. La figure 3.5 montre un exemple de modélisation floue. Supposons qu'il faille modéliser une fonction bruitée $f(x)$ avec des paires (x, y), obtenues expérimentalement (points sur la figure). La première chose à faire est de diviser le domaine, afin de modéliser la fonction $f(x)$ par intervalles. Dans chaque partition, un polynôme est ajusté, par exemple, par minimisation de la somme des carrés des erreurs. La figure 3.5.a montre 4 partitions floues et les polynômes de degré 1 (lignes), qui représentent chaque partition. Pour obtenir le modèle complet de $f(x)$, les polynômes sont combinés en accord avec les valeurs de la fonction d'appartenance, montrée en bas de la figure 3.5.b. La ligne discontinue correspond au modèle final.

Dans ce travail, la régression est faite en considérant tous les m paramètres contrôlés, c'est à dire, l'hyperplan $\beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m = 0$ est calculé pour chaque secteur d'influence, puis utilisé pour construire la fonction toute entière.

La figure 3.6 montre un exemple des modèles obtenus. Ici deux paramètres, *mut* et *rep* sont pris en compte. Ces paramètres contrôlent la probabilité d'application de deux opérateurs, respectivement *mutation* et *réparation*. Les surfaces représentent la diversité et la valeur moyenne de la fonction d'évaluation des individus de la population. Notons que, pendant la phase de contrôle, ce dont nous avons besoin correspond justement au

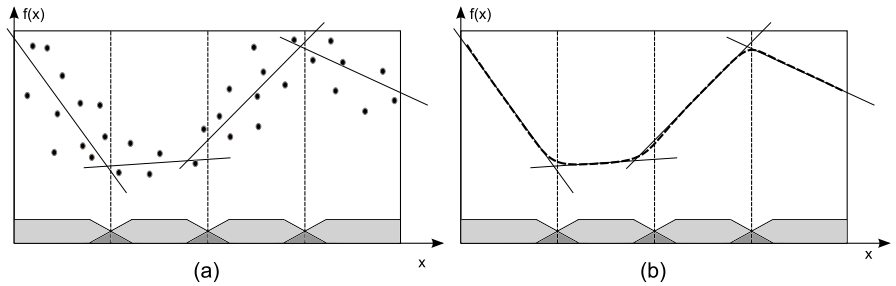

FIG. 3.5 – Modélisation floue : (a) le domaine est divisé et des régressions linéaires sont exécutées avec les points de chaque partition, (b) La fonction est composée à partir des polynômes.

contraire, c'est-à-dire, fixer les paramètres qui produiront un certain niveau de diversité. Afin de construire l'"inverse" de cette fonction non-bijective, on ne s'intéresse qu'aux valeurs qui maximisent l'espérance de qualité de la population pour les différentes valeurs de la diversité.

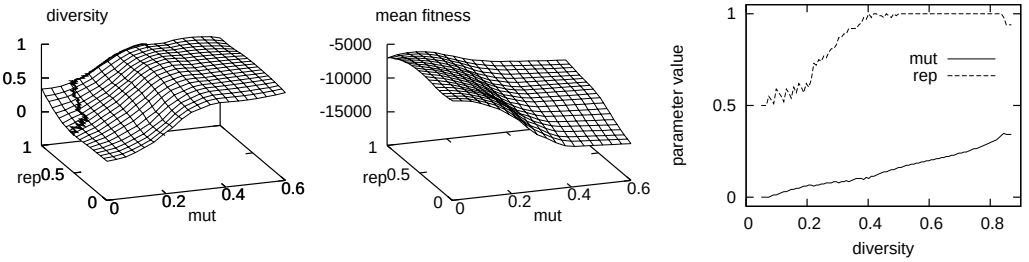

FIG. 3.6 – Information obtenue pendant la phase d'*apprentissage* : Les surfaces correspondants à la diversité et à la qualité moyenne pour deux paramètres, et la trace correspondante de la table *CachedDiv*

La ligne épaisse sur la surface de diversité montre les valeurs des paramètres qui produisent, pour toutes les valeurs de diversité que l'AE est capable d'engendrer, le niveau le plus haut de qualité, par rapport au modèle de qualité. Cette ligne est stockée sous la forme d'une table (montrée à droite), appelé *CachedDiv*. Cette table donne la configuration de paramètres optimale pour chaque valeur de diversité (sur l'axe x). Une fois que cette table est créée, le contrôleur l'utilise comme seul guide de réglage des paramètres (sans considérer la stratégie de contrôle, qui sera détaillée dans la sous-section suivante).

3.2.3 Phase de contrôle et conception des stratégies de contrôle

L'avantage de considérer séparément les phases d'apprentissage et de contrôle est que nous pouvons nous abstraire des détails de l'algorithme une fois le modèle obtenu. Notons que dans cette deuxième phase, le contrôle est totalement abstrait, car le seul aspect à

considérer est l'équilibre de l'EvE. Ceci est réalisé en contrôlant la valeur de la diversité.

Le principal défi dans cette phase est de trouver une stratégie pour régler la diversité tout au long de la recherche, dans le but d'éviter de stagner dans des optimums locaux et, en même temps, de bien exploiter l'espace de recherche et d'accélérer globalement la recherche. Plusieurs questions se posent ici : Doit-on laisser la diversité dans un niveau intermédiaire ? Doit-on la faire bouger alternativement entre exploration et exploitation ? Doit-elle diminuer lentement pour passer de l'exploration vers l'exploitation comme dans le recuit simulé ?

Si une diversité excessive est autorisée, on perdra du temps de calcul à cause d'une exploration dispersée. Si, par contre, la diversité n'est pas suffisante, on risque de converger prématurément, en perdant des informations potentiellement utiles. Ces deux arguments sont en faveur d'un niveau *"correct"* de diversité, situé quelque part entre les valeurs minimale et maximale. Certes, des problèmes différents ont des valeurs différentes de diversité "correcte". Par conséquent, le contrôleur doit être capable de la trouver. Une approche consiste à considérer la variation de la valeur de la fonction d'évaluation du meilleur individu de la population pendant les dernières générations ; si la même valeur est souvent répétée, il est probable que la population soit de nouveau attirée vers le même secteur de l'espace de recherche.

Une approche différente est de commencer avec une période d'exploration, puis de basculer vers une autre d'exploitation. Ceci pourrait aider à identifier d'abord les zones prometteuses pour y concentrer ensuite la recherche. Cependant, si l'espace de recherche est trop grand ou rugueux, si les opérateurs ne sont pas suffisamment bons, ou si le problème est "trompeur" (*deceptive*), la population pourrait ne pas être capable d'identifier les meilleures zones. Dans ce cas, une stratégie possible pourrait être d'exécuter cette opération plusieurs fois. En effet, pendant les expériences préliminaires, nous avons remarqué que certains problèmes sont très facilement résolus tout simplement en effectuant un zigzag entre des valeurs extrêmes de diversité.

Lorsque la diversité est augmentée pour échapper aux optima locaux il faut considérer plusieurs aspects. Le premier est le niveau jusqu'auquel la diversité doit s'augmenter. Si ce niveau est trop bas, les individus peuvent rester dans la même zone de l'espace de recherche et converger vers le même optimum une fois que la diversité descend encore. Si cette valeur est trop haute, on risque de perdre des informations importantes. Le deuxième aspect est le temps qu'il faut attendre avant de laisser la diversité descendre encore. Trop ou trop peu de générations peuvent produire les mêmes effets que ceux mentionnés avant. Ici on considère une période d'*"oubli"*, qui consiste à augmenter la diversité jusqu'à sa valeur maximale pendant un certain nombre de générations. Puisque les paramètres sont automatiquement réglés pour que l'AE obtienne la plus haute qualité possible, la perte de qualité est en quelque sorte limitée.

Nous avons aussi expérimenté de *petites oscillations* de la diversité autour d'un niveau central, dans le but de faire des cycles locaux d'exploration et exploitation, et de stabiliser la diversité observée par rapport à celle commandée.

Afin de comparer les différentes stratégies, les considérations précédentes ont été incluses dans les quatre stratégies suivantes :

– **MX (Mixed) :** Intègre le critère d'exploration-puis-exploitation, oubli et petite

oscillation. Une série de niveaux décroissants de diversité est commandée à l'AE, avec une petite oscillation autour du niveau nominal. Un nombre de générations est exécuté dans chaque niveau, et élargi si une amélioration historique est trouvée. Une fois que la diversité atteint son niveau le plus bas, elle est augmentée vers son niveau maximal, afin d'échapper aux optima locaux. Ce schéma est répété à plusieurs reprises.

- **CD (Correct Diversity) :** Essaie le critère de la diversité "correcte"'. Toutes les dix générations, les valeurs de qualité du meilleur individu pendant les g dernières générations sont considérées. Si plus de $\frac{g}{2}$ de ces valeurs sont répétées, la diversité est augmentée, et s'il y a moins de $\frac{g}{8}$ doublons, la diversité est diminuée.
- **ZZ (ZigZag) :** Implémente l'oscillation large autour d'un niveau central de diversité. Cette valeur est donnée par la moyenne des diversités commandées qu'ont produit les 5 dernières améliorations historiques. L'oscillation s'agrandit à mesure que les générations avancent, jusqu'à atteindre les limites de diversité possibles. Si une amélioration historique est faite, l'amplitude de l'oscillation est réinitialisée à zéro, pour commencer à croître de nouveau.
- **FX (Fixed) :** Cette stratégie laisse la diversité à une valeur fixe. Elle est utilisée plutôt comme base de comparaison.

Différents niveaux d'autonomie peuvent être identifiés au sein de ces stratégies de recherche. FX n'a pas du tout d'autonomie : elle n'est pas capable de s'adapter face aux changements du processus de recherche. ZZ et MX sont plus réactives aux changements : la stratégie est modifiée en fonction des événements observés. Finalement, la stratégie la plus autonome et la plus réactive est CD, qui surveille constamment l'état de l'AE pour régler la valeur de diversité.

3.2.4 Point de vue de l'utilisateur

La figure 3.7 montre comment la boucle principale de l'AE se met en lien avec le contrôleur. Il y a principalement deux appels aux méthodes du contrôleur, le premier pour demander de nouvelles valeurs pour les paramètres, et le deuxième pour informer le contrôleur des effets de la configuration des paramètres utilisée.

Du point de vue de l'utilisateur, l'implémentation est simple. Il sufit simplement d'inclure deux appels au contrôleur, et deux nouvelles méthodes (marquées en gris sur la figure). La première méthode doit instancier les valeurs données par le contrôleur pour les paramètres de l'AE, tandis que la deuxième doit fournir les mesures de qualité et diversité au contrôleur.

3.2.5 Outil d'analyse

La phase d'apprentissage produit deux résultats qui peuvent s'avérer utiles dans l'étude du fonctionnement interne de l'AE. Considérons les graphiques de la figure 3.6, qui montrent les surfaces du CF correspondant à la diversité et à la qualité, et le graphique de *CachedDiv*. En analysant la surface de diversité, l'effet important de la mutation se fait sentir : la diversité augmente conjointement avec *mut* jusqu'à atteindre son niveau

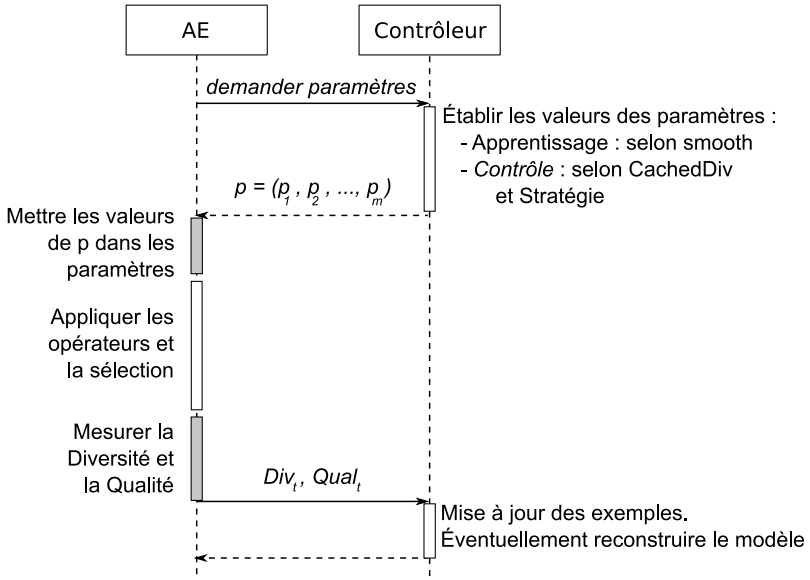

FIG. 3.7 – Interaction entre l'AE et le contrôleur

maximal aux environs de $mut = 0.3$. D'autre part, *rep*, qui a une faible influence sur la diversité, a un effet important sur la valeur moyenne de qualité. Effectivement, quand les deux valeurs sont grandes ($mut > 0.3$ et $rep > 0.5$), un effet conjoint peut être observé : au fur et à mesure que la mutation fait augmenter la diversité, l'effet secondaire de la chute de qualité est réparé par l'autre opérateur, empêchant une dégradation excessive des individus.

Les CF de diversité et qualité, et notamment le tableau CachedDiv peuvent être utilisés pour obtenir des informations utiles sur l'AE contrôlé. Une courbe plutôt plate dans CachedDiv indique un paramètre avec un effet faible ou nul sur l'équilibre EvE, et sa valeur peut donc être fixée à celle qui apparaît dans le tableau. Si plusieurs paramètres ont un comportement similaire, ils pourraient être manipulés ensemble. Si deux paramètres sont complémentaires (par exemple, si leur somme est presque identique pour toutes les valeurs de diversité), on pourrait remplacer l'un d'eux par une règle permettant d'obtenir sa valeur à partir de l'autre.

Bien que cet exemple puisse sembler limité, l'utilisation de cette approche peut devenir un outil précieux lorsqu'on travaille avec trois ou plus de paramètres, où l'effort pour appréhender les valeurs des paramètres afin d'obtenir des niveaux de diversité donnés devient insurmontable.

3.3 Cadre expérimental

Puisque notre méthode utilise l'apprentissage et le contrôle séparément, la validation doit être double. D'abord, on veut vérifier que le tableau CachedDiv inclut les combinaisons de paramètres qui produisent vraiment les diversités requises et des populations

de bonne qualité. Ensuite, on souhaite trouver la stratégie de contrôle qui produit les meilleurs résultats pour plusieurs problèmes. Les paramètres contrôlés correspondent aux probabilités d'application des opérateurs décrits dans la section 3.3.1.

Le contrôleur travaille sur un AE qui résout plusieurs instances du QAP (voir section1.4.1). Le but ici n'est pas d'être compétitif avec des algorithmes spécialisés dans la résolution de ce problème, mais de comparer la performance du contrôleur en utilisant différentes stratégies de contrôle.

Un ensemble de 38 instances de taille moyenne extraite de la librairie *QAPLIB* [Burkard *et al.*, 1997], incluant des instances appartenant à toutes les familles, a été sélectionné pour tester notre approche.

3.3.1 Algorithme Évolutionnaire

Les individus sont codés comme des permutations. La taille de la population est fixée à 100 individus et trois opérateurs sont appliqués : la mutation standard, qui échange deux variables aléatoirement, le Croisement Cyclique (CX) [Oliver *et al.*, 1987], et un opérateur spécialisé, (*remake*), qui efface aléatoirement quatre variables, et essaie les 4! reconstructions possibles pour choisir la meilleure. La sélection est faite par roulette (voir section 1.3.6). 15 exécutions de 10.000 générations (sans compter les générations dédiées à l'apprentissage) ont été réalisées pour chaque instance et chaque stratégie. On a choisi de calculer un grand nombre de générations afin d'avoir des résultats fiables sur la convergence du processus de la recherche.

3.3.2 Mesure de diversité

En utilisant le codage par permutation, on mesure la diversité comme le complément de la similarité entre individus. On définit la similarité de la population par la formule suivante :

$$sim = \sum_{i=1}^{t} \sum_{j=1}^{t} \sum_{k=1}^{n} c(i,j,k) \quad i \neq j \tag{3.1}$$

Où n est le nombre de variables du problème, t est la taille de la population et la fonction c est définie par :

$$c(i,j,k) = \begin{cases} 1 & \text{si la variable } k \text{ a la même valeur dans les individus } i \text{ et } j \\ 0 & \text{sinon} \end{cases}$$

Afin de normaliser la similarité (et donc la diversité) il faut connaître ses bornes inférieure et supérieure. La borne supérieure est trouvée une fois que tous les individus sont égaux, et elle est donnée par l'expression suivante :

$$l_s = t \cdot (t-1) \cdot n$$

Calculer la borne inférieure est moins évident. Si $t \leq n$ (figure 3.8.a), la limite inférieure est égale à 0, pour une population dont les valeurs des variables sont décalées d'une place chaque fois. En général, la borne inférieure est donnée par la formule suivante :

$$l_i = \begin{cases} 2 \times n \times A \times B & \text{if } A < 2 \\ 2 \times n \times A \times B + \frac{A!}{(A-2)!} \times n^2 & \text{if } A \geq 2 \end{cases}$$

Où A et B correspondent au quotient et au reste de la division de t par n (A=t *div* n, B=t *mod* n), respectivement. A correspond au nombre de blocs de similarité minimale, indiqués en gris sur la figure 3.8. B est le nombre d'individus qui n'appartiennent pas à un bloc complet. La similarité minimale entre deux blocs est de $2n^2$ unités, et chaque individu hors-bloc contribue individuellement avec $2n$ pour chaque bloc existant (figure 3.8.b). L'expression $\frac{A!}{(A-2)!} \times n^2$ correspond à la similarité apportée par la permutation de tous les blocs existants (figure 3.8.c), alors que $2 \times n \times A \times B$ exprime la similarité apportée par les individus hors-bloc.

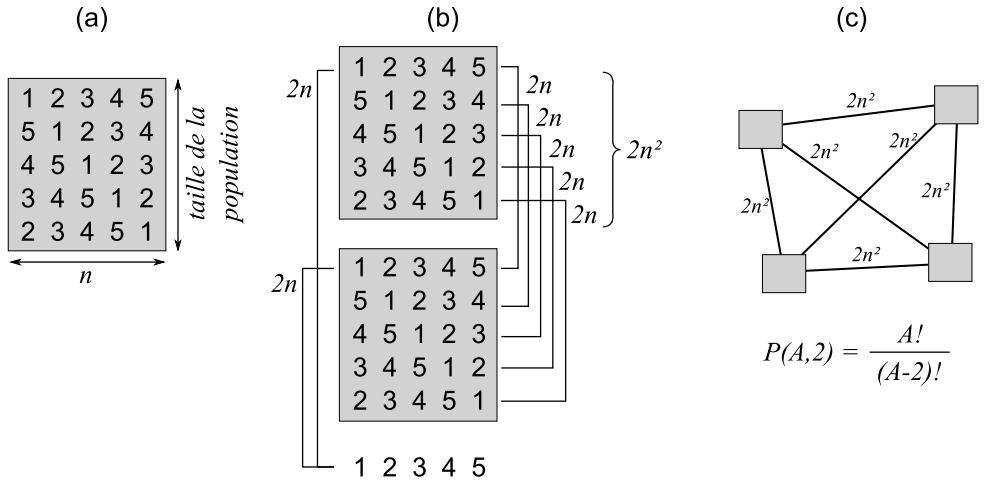

FIG. 3.8 – Borne inférieure de similarité pour un codage par permutation

Une fois calculées les bornes inférieures et supérieures de similarité, il est possible de normaliser la similarité, et ce qui nous intéresse particulièrement, normaliser la diversité de la population, donnée par l'expression :

$$div = \frac{l_s - sim}{l_s - l_i}$$

Calcul de la diversité

Les bornes inférieures et supérieures sont calculées une fois pour toutes, tandis que la similarité est calculée pour chaque génération. La complexité de calcul de l'équation 3.1 est en $O(nt^2)$, où n est le nombre de variables et t la taille de la population. Cependant, cette complexité peut être réduite si certaines structures de données sont créées et si on décompose le calcul en deux étapes :

1. Créer une matrice $A = (a_{ij})_{n \times n}$, qui stockera le nombre d'apparitions de chaque variable i en la position j. Visiter toutes les variables des individus pour remplir la matrice A (ordre $O(np)$).
2. Obtenir la similarité comme $\sum_{j=1}^{n} \sum_{i=1}^{n} a_{ij} \times (a_{ij} - 1)$ (ordre $O(n^2)$).

Ceci permet de diminuer la complexité de calcul de $O(nt^2)$ en $O(nt + n^2)$[1]

3.3.3 Méta-paramètres de la phase d'apprentissage

Pendant la phase d'apprentissage, 2.000 générations ont été ignorées au début des sous-phases de *Collecte d'exemples* et de *Raffinement*. Les domaines des paramètres ont été divisés en 4 partitions floues et sous-divisés avec une *finesse* égale à 3. Cinq exemples ont été pris à l'intérieur de chaque boîte de la grille d'entraînement. Durant la sous-phase de *Raffinement*, la diversit décroît linéairement pendant 800 générations, puis augmente pendant les 800 suivantes. Dans le but d'exclure l'influence de la modélisation dans la comparaison des stratégies, 15 exécutions préliminaires ont été effectuées pour chaque instance, afin de choisir un tableau CachedDiv unique. On a choisi le tableau qui présentait la plus petite différence entre les diversités observées et commandées pendant les exécutions préliminaires.

3.3.4 Stratégies de contrôle

Pendant la phase de contrôle, six stratégies (voir section 3.2.3) ont été comparées, trois dynamiques (MX, CD, ZZ) et trois statiques (FX).

Les niveaux intermédiaires de diversité dans MX sont 0.7, 0.6, 0.5, 0.4, 0.3 et 0.2 (dans l'intervalle de diversité $[0, 1]$). Chaque niveau est maintenu pendant 300 générations. La petite oscillation est de $\pm 10\%$, et la période d'oubli est de 200 générations.

Pour CD, $g = 100$, et les augmentations et les réductions de diversité sont de 0.003 et 0.001, respectivement.

Trois stratégies statiques ont été définies par les valeurs de diversité 0.4, 0.5 et 0.6. Elles sont appelées FX.4, FX.5 et FX.6, respectivement.

3.4 Résultats et discussion

La table 5.1 présente la moyenne de la différence en pourcentage par rapport au meilleur résultat connu et à l'écart type (entre parenthèses) pour les différentes stratégies (colonnes) et instances (lignes). Par exemple, la valeur 0.07(0.05) signifie que le résultat moyen des 15 exécutions est à 0.07% en dessous de la meilleure solution connue pour cette instance, et que l'écart type est de 0.05% par rapport à cette valeur. La valeur dans la colonne de droite montre la meilleure solution publiée sur le site web de QAPLIB (juin 2008). Le nombre moyen de solutions optimales (sur 15 exécutions) apparaît dans le bas de la table. Le nombre d'instances sur lesquelles chaque méthode a battu les autres est aussi présenté

[1] Les valeurs de n pour les instances de QAPLIB vont de 12 à 256, et car t est 100, les économies de temps sont réelles.

Tab. 3.1 – Moyenne de l'erreur percentuelle et écart type, par rapport au meilleur résultat connu

Instance	MX	CD	ZZ	FX.4	FX.5	FX.6	Meilleur
bur26a	0.07(0.05)	0.04(0.05)	0.09(0.03)	0.12(0.06)	0.06(0.04)	0.08(0.03)	5426670
bur26b	0.05(0.06)	0.07(0.08)	0.05(0.06)	0.21(0.11)	0.07(0.08)	0.03(0.04)	3817852
bur26g	0(0)	0(0.01)	0.01(0)	0.02(0.1)	0(0)	0.01(0)	10117172
bur26h	0(0)	0.04(0.15)	0(0)	0.17(0.28)	0(0)	0(0)	7098658
chr12a	0(0)	0(0)	0(0)	1.01(2.13)	0.27(1.03)	1.52(2.6)	9552
chr18b	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	1534
chr20c	5.92(4.78)	10.05(6.83)	2.26(2.93)	14.05(8.47)	12.18(11.73)	11.16(5.67)	14142
chr25a	12.8(3.81)	9.58(5.08)	24.81(4.97)	9.95(6.26)	10.22(5.29)	13.96(5.69)	3796
els19	0.24(0.37)	1.1(2.48)	0(0)	1.44(4.52)	3.22(6.31)	0.44(0.5)	17212548
esc32a	2.3(0.76)	5.38(2.3)	12.3(3.07)	1.53(1.53)	3.07(0.76)	6.15(1.53)	130
esc32b	3.57(4.76)	8.92(4.16)	13.09(1.78)	4.76(4.76)	2.97(4.16)	4.16(4.76)	168
esc64a	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	116
had12	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	1652
had20	0(0)	0(0)	0(0)	0.27(0.26)	0.23(0.26)	0.21(0.18)	6922
kra30a	1.93(0.6)	2.01(0.75)	3.26(0.62)	1.86(0.85)	1.58(1.03)	2.75(0.46)	88900
lipa20a	0.35(0.57)	0.19(0.32)	0.32(0.57)	0.13(0.32)	0.27(0.54)	0.27(0.48)	3683
lipa40b	5.79(8.47)	6.94(8.8)	17.24(3.98)	4.68(8.04)	2.37(5.88)	7.59(8.88)	476581
lipa60a	1.12(0.05)	0.97(0.04)	1.33(0.03)	1.21(0.04)	1.25(0.02)	1.3(0.02)	107218
lipa60b	19.1(5.29)	19.25(0.24)	21.77(0.2)	20.8(0.19)	21.21(0.23)	21.59(0.16)	2520135
nug15	0(0)	0(0)	0(0)	0.08(0)	0(0)	0(0)	1150
nug20	0.11(0.11)	0.03(0.07)	0.15(0.23)	0.07(0.03)	0.03(0.03)	0.07(0.07)	2570
nug30	0.86(0.42)	0.52(0.34)	2.23(0.4)	0.52(0.29)	1.3(0.57)	1.91(0.35)	6124
rou20	0.56(0.21)	0.45(0.24)	0.61(0.46)	0.33(0.33)	0.41(0.26)	0.47(0.31)	725522
scr20	0.31(0.38)	0.08(0.16)	0.13(0.23)	0.18(0.31)	0.1(0.22)	0.05(0.1)	110030
sko42	1.07(0.39)	0.94(0.3)	3.34(0.38)	0.93(0.35)	1.61(0.3)	2.44(0.27)	15812
sko64	1.43(0.24)	1.14(0.36)	5.02(0.43)	2.46(0.34)	3.49(0.28)	4.22(0.3)	48498
ste36a	2.28(1.07)	1.86(1.02)	7.78(1.55)	2.33(1.17)	2.62(0.82)	3.94(0.85)	9526
ste36b	2.25(1.84)	3.08(3.31)	7.07(1.6)	3.57(3.47)	3.19(2.71)	3.05(1.77)	15852
ste36c	1.97(0.92)	1.47(1.02)	3.47(0.84)	2.19(1.24)	1.8(1.16)	2.8(1.09)	8239110
tai20a	1.01(0.4)	0.89(0.22)	1.32(0.34)	0.84(0.25)	0.88(0.37)	0.8(0.49)	703482
tai20b	0.08(0.18)	0.3(0.22)	0.17(0.21)	0.21(0.23)	0.3(0.22)	0.17(0.21)	122455319
tai40a	3.52(0.38)	2.91(0.41)	5.26(0.35)	4.08(0.36)	4.65(0.36)	5.14(0.36)	3139370
tai40b	1.6(1.26)	2.11(1.93)	2.7(1.28)	2.04(1.81)	2.03(1.49)	1.83(1.24)	637250948
tai60a	5.68(0.46)	3.64(0.26)	7.17(0.31)	6.3(0.34)	6.72(0.16)	7.13(0.24)	7205962
tai60b	1.54(0.82)	1.35(0.74)	3.58(0.77)	5.7(3.4)	1.66(1.93)	1.91(0.83)	608215054
tai64c	0.1(0.11)	0.03(0.03)	0.1(0.11)	0.3(0.2)	0.28(0.19)	0.16(0.14)	1855928
tho40	1.55(0.56)	1.45(0.58)	4.75(0.66)	1.39(0.33)	2.68(0.37)	3.71(0.44)	240516
wil50	0.43(0.11)	0.36(0.18)	1.86(0.16)	0.86(0.21)	0.69(0.1)	1.22(0.11)	48816
opt.moy	4.82	4.6	4.29	4.5	4.61	3.87	
bat MX	–	8	2	2	1	1	
bat CD	3	–	1	2	2	1	
bat ZZ	21	21	–	18	21	16	
bat FX.4	12	11	6	–	6	5	
bat FX.5	12	12	4	9	–	14	
bat FX.6	16	17	4	12	1	–	

en bas de la table. La comparaison entre méthodes a été effectuée en utilisant le test T Student avec un niveau de confiance de 95%.

L'efficacité des stratégies peut être appréciée en regardant le nombre moyen de fois où l'algorithme a atteint l'optimum. En utilisant ces critères, la meilleure stratégie semble être MX, suivi par FX.5, CD, FX.4, ZZ et FX.6. Néanmoins, si on s'intéresse à des stratégies qui puissent être appliquées lors de situations différentes, on observera plutôt le nombre de fois où une stratégie a obtenu de meilleures valeurs que les autres. CD a battu les autres méthodes 69 fois, suivi par MX, FX.4, FX.6, FX.5 et ZZ. CD et MX semblent être les stratégies les plus génériques, et elles pourraient fonctionner correctement sur la plupart des problèmes. ZZ est la seule stratégie capable de résoudre l'instance *els19* à chaque fois, mais elle obtient de piètres résultats sur le reste. Ceci est un bon exemple du principe *No free lunch* : d'un côté il y a des algorithmes spécialisés pour des problèmes spécifiques, et de l'autre des algorithmes "généralistes" qui sont moyennement efficaces sur plusieurs problèmes mais n'excellent sur aucun.

Une question est alors de savoir si la variation de la diversité est nécessaire. Considérons la stratégie statique la moins battue, FX.4, et comparons-la avec CD et MX. FX.4 bat les autres dans 43 instances, alors que CD le fait 69 et 64 fois, respectivement. D'un autre point de vue, CD et MX sont battus à 9 et 14 occasions, alors que FX.4 l'est dans 40. Les résultats suggèrent que le contrôle de paramètres, s'il est effectué de façon soigneuse, a des avantages clairs par rapport aux réglages fixes, notamment quand le réglage adéquat est inconnu.

Afin de comprendre comment les stratégies CD, MX et ZZ fonctionnent, on développera l'étude de trois instances représentatives. Nous sommes intéressés par l'exactitude du modèle, i.e., si la diversité observée suit la diversité commandée, et comment les modifications dans la diversité commandée aident à améliorer la qualité.

Considérons CD sur l'instance *tai64c* (figure 3.9). La diversité observée reste dans un intervalle $\pm 10\%$ par rapport à la diversité commandée. La diversité a été plutôt bien modélisée, au moins dans l'intervalle $[0.6, 0.9]$, où elle est utilisée. Au début de l'exécution, la diversité commandée est réduite afin de "concentrer" la population initiale. Vers la génération 1800, la diversité insuffisante produit une stagnation de la population, identifiée par une courbe plate dans les données correspondant à la qualité du meilleur individu. Cette situation est précocement détectée par le contrôleur et la diversité est augmentée jusqu'à un niveau plutôt stable, proche de 0.8, dans un compromis entre l'exploration et l'exploitation.

Considérons maintenant MX pour résoudre *ste36b* (figure 3.10). La diversité observée reste plus proche de la commandée que dans CD. Cela est peut être dû à l'oscillation de la diversité commandée : le fait que plusieurs niveaux soient commandés dans un temps court diminue la probabilité qu'une diversité mal modelisée soit commandée, obtenant un bon comportement de la diversité observée. Les cycles multiples de l'exploration-puis-exploitation de MX s'avèrent utiles pour s'échapper des optimums locaux. Notons que la population commence à converger autour de la génération 4.700, d'où elle ne pourrait probablement pas s'échapper ensuite. L'augmentation de la diversité permet à la population d'*oublier* cet optimum pour en trouver plus tard un autre meilleur.

La figure 3.11 illustre le comportement de ZZ sur l'instance *els19*. Dans ce cas, le modèle

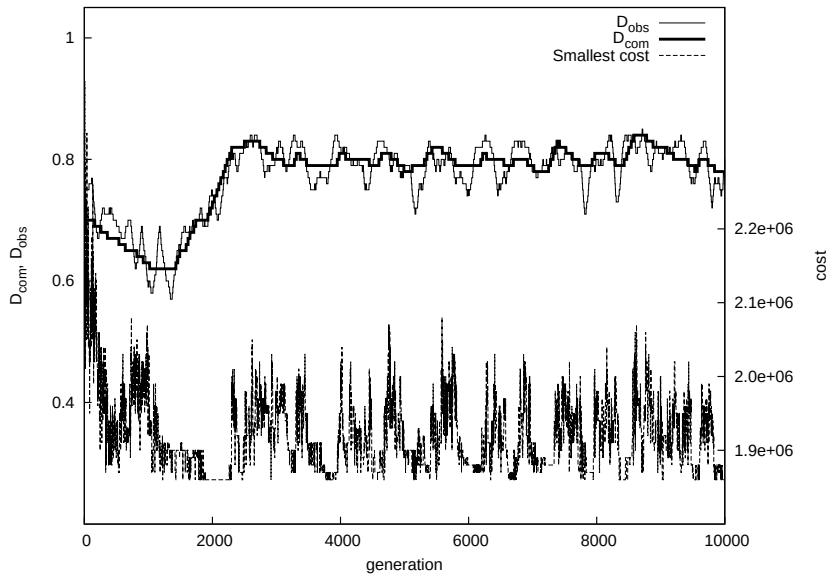

FIG. 3.9 – Graphique de la diversité commandée (D_{com}), moyenne glissante (100) de la diversité observée (D_{obs}) et évaluation (coût) du meilleur individu (en bas) dans *tai64c*, en utilisant CD

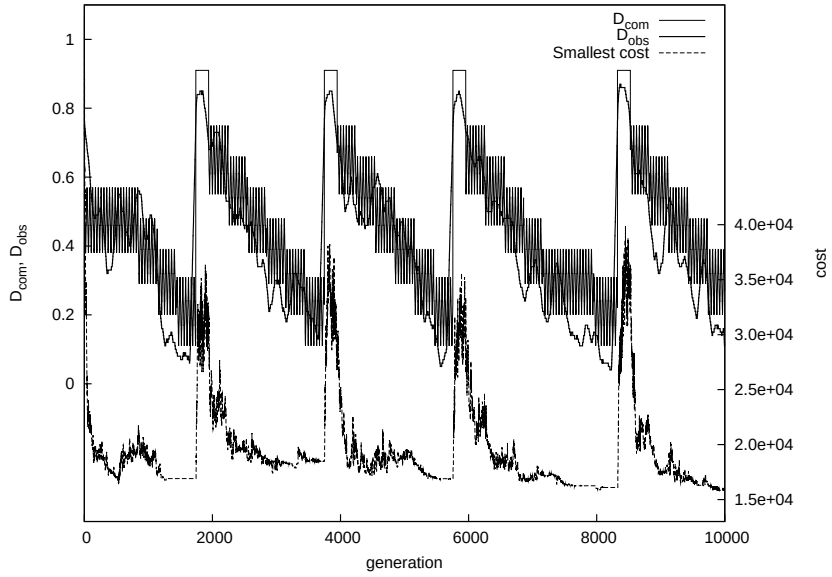

FIG. 3.10 – Graphique de la diversité commandée (D_{com}), moyenne glissante (100) de la diversité observée (D_{obs}) et évaluation (coût) du meilleur individu (en bas) dans *ste36b*, en utilisant MX

tend à produire des diversités inférieures à celles commandées, mais ceci ne semble pas

être important, puisque ZZ a résolu cette instance dans chaque exécution. Les meilleures améliorations ont été produites dans les générations 650, 1930 et 8267 (où l'oscillation a été remise à zéro). Dans tous les cas les améliorations ont été trouvées lorsque la diversité était proche de 0.7. Il semble donc simplement utile de conserver une bonne diversité. En fait, les exécutions avec une diversité fixée à 0.7 (non montrées ici) ont produit presque les mêmes résultats.

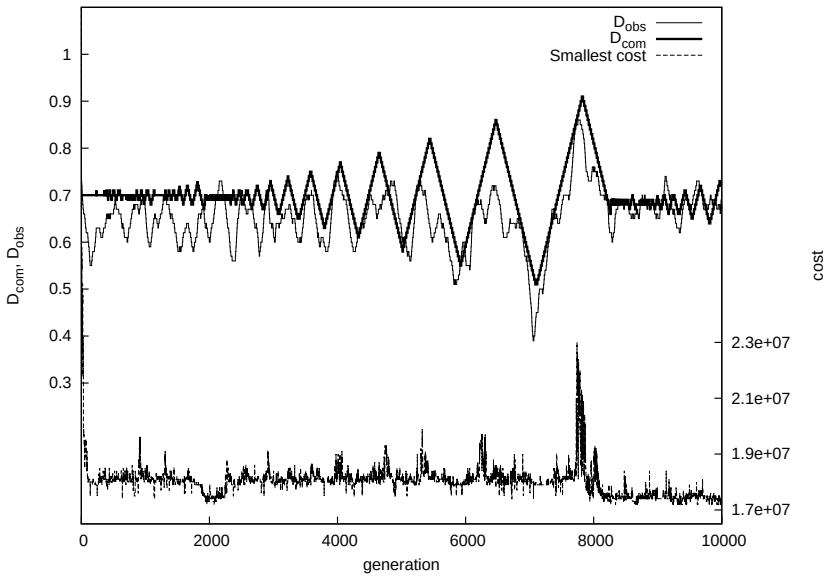

FIG. 3.11 – Graphique de la diversité commandée (D_{com}), moyenne glissante (100) de la diversité observée (D_{obs}) et évaluation (coût) du meilleur individu (en bas) dans *els19*, en utilisant MX

La question la plus étonnante est peut-être de savoir pourquoi ni CD ni MX n'ont pas atteint cette valeur. La figure 3.12 montre les graphiques d'*els19* résolu par CD et MX.

CD réduit systématiquement la diversité en-dessous de 0.7, provoquant une stagnation dans des optima locaux. Ceci est probablement dû au schéma de sélection utilisé : *els19* a de grandes valeurs de coût ($\sim 10^7$), donc la sélection par roulette ne peut pas toujours choisir la meilleure valeur d'une génération à la prochaine, puisque même des différences de qualité importantes ne produisent que de petites différences dans les probabilités correspondantes. Ceci donne l'illusion que la population est dispersée, raison pour laquelle CD diminue la valeur de diversité. D'autre part, le problème avec MX est que la plupart du temps la diversité est au-dessus ou en-dessous de 0.7, donc la population n'a pas le temps de trouver l'optimum global. Notons que lorsque la diversité reste plus longtemps à cette valeur, des valeurs de coût inférieures sont produites (générations 2300 et 8400 dans la figure, en employant MX).

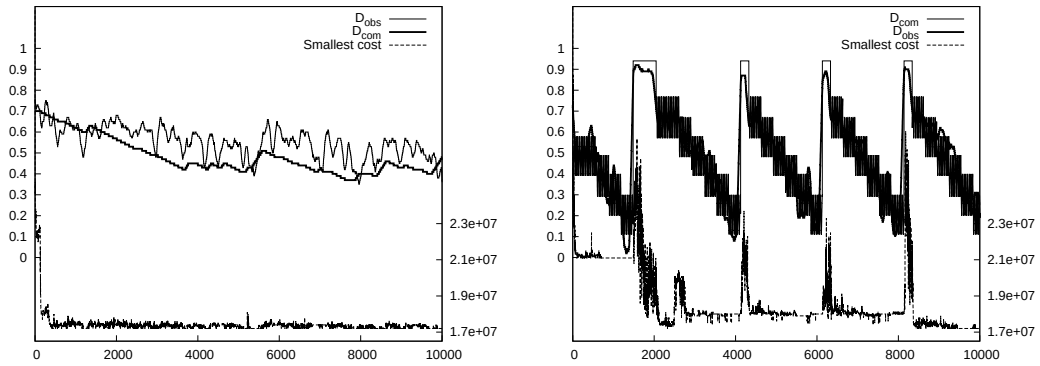

FIG. 3.12 – Graphique de la diversité commandée (D_{com}), moyenne glissante (100) de la diversité observée (D_{obs}) et évaluation (coût) du meilleur individu (en bas) dans *els19*, en utilisant CD (à gauche) et MC (à droite)

3.4.1 Temps d'apprentissage

L'inconvénient principal de cette approche est le temps consacré à la phase d'apprentissage, étant donné le rapport exponentiel entre le nombre de paramètres contrôlés et le nombre d'exemples à collecter. Ce phénomène est connu sous le nom de "malédiction de la dimensionalité". Le nombre de générations exigées par cette phase de l'étude est donné par l'expression suivante :

$$G_i + e \times \prod_{j=1}^{d} (p_j \times f) + G_r \tag{3.2}$$

où G_i et G_r sont le nombre de générations dans les sous-phases correspondant à l'oubli des exemples et celle de raffinement, p_j est le nombre de partitions floues du paramètre j, f est la finesse, et e est le nombre d'exemples pris dans chaque boîte de la grille d'entraînement. La table 3.2 montre le nombre et le pourcentage de générations consacrées à chaque phase, utilisant les méta-paramètres décrits ci-dessus.

TAB. 3.2 – Distribution de générations par phase et sous-phase

Phase	Sous-phase	Générations	Pourcentage
Apprentissage	d'oubli	2.000	9%
	Collection d'exemples	8.640	39%
	Raffinement	1.600	7%
Contrôle		10.000	45%

Les sous-sections suivantes abordent quelques extensions et modifications possibles pour améliorer la méthodologie, en essayant de réduire particulièrement le temps d'apprentissage. Ces extensions ont été implémentées sans obtenir des améliorations en termes

de qualité de solution.

3.4.2 Arbre d'apprentissage

L'idée principale ici est d'identifier rapidement les combinaisons inutiles afin d'élaguer l'espace de recherche des paramètres. La figure 3.13 présente un espace de recherche composé de 2 paramètres. La ligne foncée représente le CachedDiv idéal que nous essayons de trouver. L'idée est de diviser le domaine de chaque paramètre en un certain nombre d'intervalles (2 dans l'exemple), et prélever un échantillon de diversité et de qualité dans chaque secteur (points sur la figure). Les valeurs moyennes donnent une approximation pour comparer les secteurs et pour décider ceux qui doivent être explorés plus en détail. La comparaison est effectuée sur ces deux critères, et seuls les secteurs qui sont Pareto-dominants sont "dépliés" (B et C, dans la figure). Le processus est répété à plusieurs reprises, en ouvrant des sous-secteurs avec différents compromis d'EvE. À la fin, seules les zones grises devraient être fortement explorées. Pour explorer cette approche, nous avons développé une méthode capable d'explorer un espace de recherche n-dimensionnel, en divisant chaque dimension en s intervalles à chaque fois.

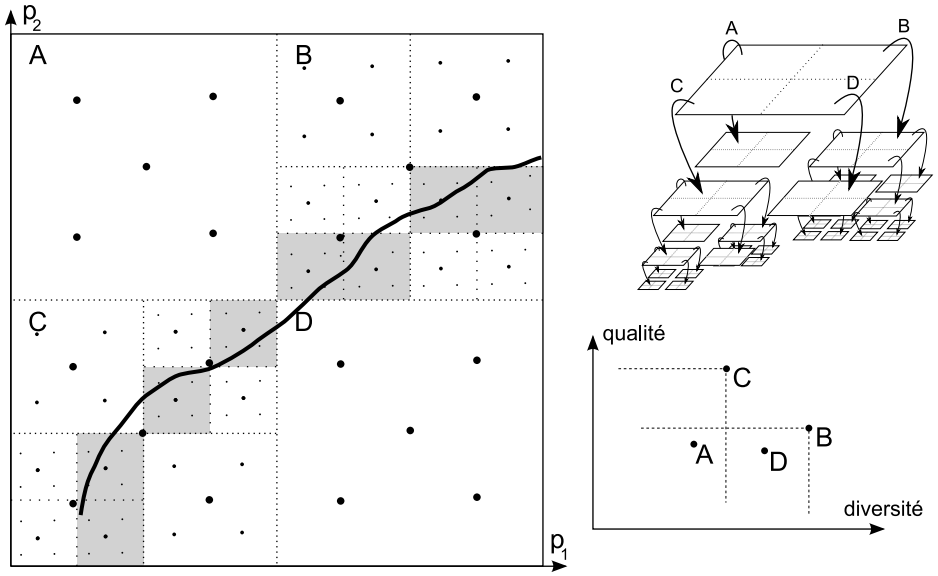

Fig. 3.13 – Fonctionnement prévu de l'arbre d'apprentissage. L'espace de recherche des paramètres est exploré selon sa Pareto-dominance. Dans le premier niveau les zones B et C sont ouvertes parce qu'elles sont Pareto-dominantes par rapport à A et D

Le problème avec cette approche est la vitesse de l'AE pour trouver une solution : indépendamment des méta-paramètres utilisés, la diversité et la qualité augmentent rapidement, ainsi les exemples dans un petit secteur dominent le reste des points, évitant l'exploration d'autres segments de CachedDiv. Dans la pratique, cette méthode pourrait s'avérer utile pour explorer les espaces qui n'évoluent pas (ou bien qui le font lentement). Ce problème pourrait être surmonté en comparant plusieurs exécutions parallèles des al-

gorithmes, une dans chaque secteur déplié, en utilisant une méthode de "Racing" comme celle proposée par [Yuan and Gallagher, 2007]. Ces méthodes réalisent des tests statistiques pendant l'exécution afin d'identifier et éliminer prématurement les configurations moins bonnes dans le but d'économiser du temps de calcul. Pour plus de détails, voir section 4.6.2.

3.4.3 Partitionnement flou automatique

Le placement des partitions floues a une influence cruciale sur la qualité du modèle. Considérons la fonction modélisée sur la figure 3.14 avec deux schémas de partition différentes. La ligne foncée de la figure 3.14.a représente la fonction à modéliser, et la ligne discontinue le modèle, obtenu par régression linéaire dans chaque partition floue, étant ces partitions placées à intervalles réguliers (montrés en bas). La figure 3.14.b illustre la modélisation de la même fonction, cette fois avec les partitions floues placées d'une manière plus pertinente. Notons que des améliorations importantes du modèle peuvent être obtenues par des partitions légèrement déplacées. L'implémentation d'un contrôleur capable de placer automatiquement ses partitions possède deux avantages : d'une part, l'arrangement automatique des partitions floues élimine la nécessité d'avoir un méta-paramètre pour le nombre de partitions floues (p_j dans l'équation 3.2) ; d'autre part, une meilleure division peut réduire le nombre de partitions utilisées, et de ce fait diminuer le nombre de générations durant la sous-phase de collection d'exemples.

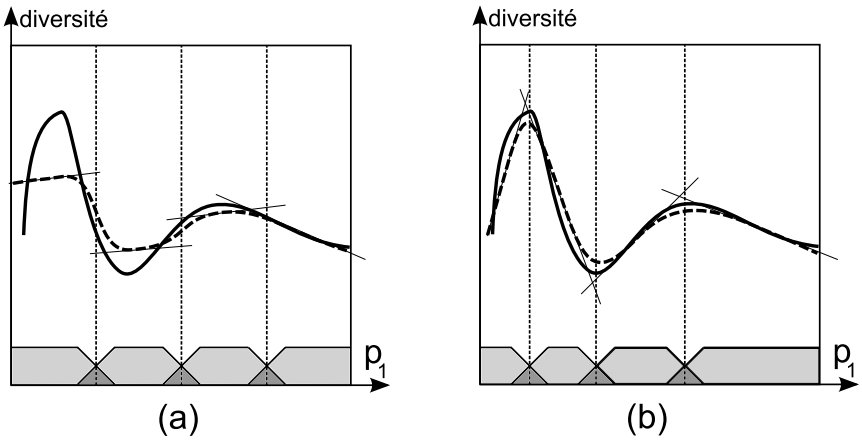

Fig. 3.14 – Effet du placement des partitions floues dans le modélage d'une fonction

Plusieurs méthodes autonomes de gestion ont été proposées pour définir la position des partitions floues dans des CF de type Mamdani [Riid and Rüstern, 2004; Piegat, 2001]. Nous avons expérimenté une méthode inspirée par ces travaux, apportant les modifications nécessaires pour l'employer avec des contrôleurs de type Takagi-Sugeno.

Ce processus est illustré par la figure 3.15. Les lignes foncées représentent la fonction à modéliser, les lignes discontinues représentent le modèle et les lignes fines représentent l'approximation linéaire de chaque partition floue. Au bas de chaque graphique, l'erreur du modèle et l'arrangement des partitions est indiqué.

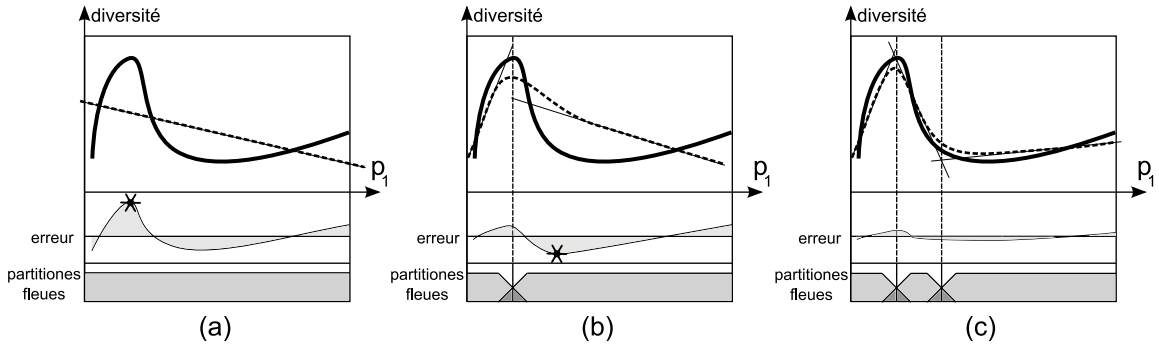

FIG. 3.15 – CF d'organisation autonome. Exemple de placement des partitions floues, guidé par l'erreur du modèle

Au début (a), une partition simple, qui couvre tout le domaine des paramètres, modélise la fonction comme une fonction linéaire. Le premier point de coupe est défini en observant la fonction erreur : le point maximal sur le plus grand secteur d'erreur est choisi pour être coupé (identifié par un astérisque). La fonction est encore modélisée (b), cette fois en considérant la coupe. Un nouveau point de découpage est défini et le modèle est reconstruit (c).

Afin d'empêcher le surapprentissage du modèle, les données rassemblées sont divisées en deux ensembles, le premier, de taille $\frac{2}{3}$ des exemples, est employé pour l'entraînement, et le reste pour contrôler la généralité du résultat obtenu. Des coupes sont placées dans la zone plus élevée d'erreur, en considérant tous les paramètres. Si un nouveau modèle (après avoir fait une coupe) est moins précis que le précédent, la dernière coupe est supprimée du modèle et mise dans une liste taboue, jusqu'à ce qu'une autre coupe réussie soit trouvée. Le processus continue jusqu'à ce que l'erreur du modèle ait atteint une fraction de l'erreur du premier modèle ou jusqu'à ce qu'il ne soit plus possible de placer de nouvelles partitions à cause du surapprenstissage.

Comparé aux méthodes existantes, nous avons modifié un détail petit mais crucial. Étant donné que les conséquentes des règles (sorties) dans les contrôleurs de type Mamdani représentent les niveaux fixes de la variable de sortie, les points d'erreur maximaux correspondent typiquement au centre d'une règle floue. Par contre, dans les CF de type Takagi-Sugeno, les points d'erreur maximaux correspondent aux *bords* des partitions floues, et non à leurs centres.

Cette méthode élimine la nécessité de définir un certain nombre de partitions floues et d'établir manuellement les domaines des paramètres (notez que dans la figure 3.6 le domaine de l'opérateur de mutation est limité à $[0, 0.6]$ afin de ne pas "gaspiller" de partitions floues dans le plateau qui s'étend au-dessus de 0.4).

Dans notre problème, l'effet des paramètres sur la diversité et la qualité présente souvent une forme simple, donc la précision n'a pas été beaucoup améliorée. Cependant, cette approche présente une avancée intéressante pour notre contrôleur, puisqu'elle nous permet de nous débarrasser de quelques paramètres, en fournissant un algorithme plus autonome.

3.4.4 Intervalle de confiance pour CachedDiv

Au fur et à mesure que la recherche avance, les individus de l'AE parcourent différentes zones de l'espace de recherche. La forme du paysage de recherche peut varier dans les différentes zones de l'espace de recherche, et donc l'effet des paramètres peut changer lors des différentes étapes de la recherche. Afin de mettre à jour le modèle, nous avons essayé de mélanger la sous-phase de raffinement et la phase de contrôle. L'idée est de raffiner constamment le modèle, en particulier pour les valeurs des paramètres qui produisent les niveaux de diversité les plus demandés. Ceci pourrait avoir l'avantage annexe de réduire le nombre de générations nécessitées par la phase d'apprentissage initial ; un apprentissage approfondi serait superflu : un premier modèle approximatif suffirait pour démarrer, et le CachedDiv correct serait construit en cours d'exécution.

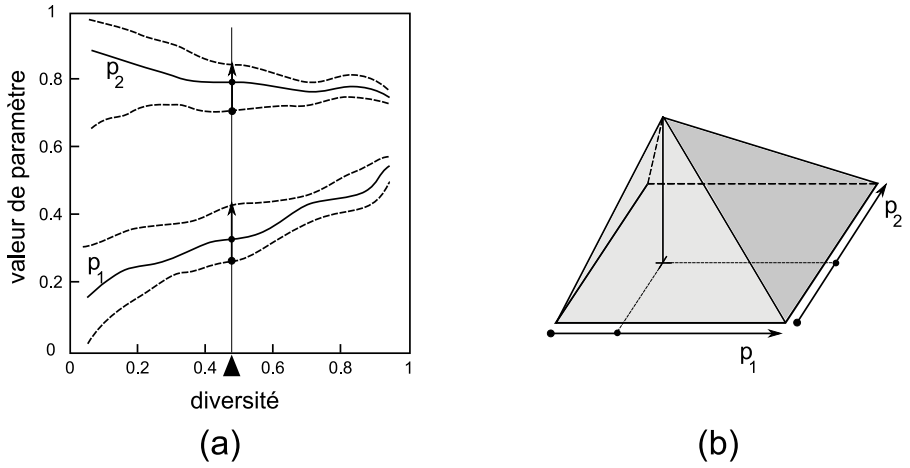

Fig. 3.16 – Intervalle de confiance pour le tableau CachedDiv (a). Distribution triangulaire pour le réglage de paramètres, basée sur les longueurs des intervalles (b)

Un intervalle de confiance est défini pour chaque paramètre et pour tous les niveaux de diversité. La largeur de l'intervalle dépend de l'exactitude de la diversité observée par rapport à la diversité commandée. Il est d'autant plus mince que les valeurs des paramètres qui produisent un certain niveau de diversité sont bien identifiées et plus large si plusieurs valeurs peuvent produire cette diversité. La figure 3.16.a montre le tableau cachedDiv avec l'intervalle de confiance pour chaque paramètre. La largeur de chaque niveau de diversité est obtenue en regardant le CF de diversité. Un seuil autour de la diversité est fixée, par exemple, à $\pm 2\%$, et maintenue en tant que limite inférieure et supérieure du cachedDiv.

Afin d'explorer l'espace de recherche des paramètres pendant l'exécution, l'instanciation de paramètres est faite en utilisant une distribution triangulaire, prenant sa valeur dans le tableau cachedDiv comme mode, et celles qui correspondent aux limites inférieures et supérieures en tant que limites de la distribution. La figure 3.16.b montre la fonction de densité de probabilité pour deux paramètres, en utilisant les intervalles correspondant à une diversité donnée (indiquée par un petit triangle au bas de figure 3.16.a).

Cette approche fournit un meilleur modèle de l'espace de recherche de paramètre.

Cependant, le coût d'obtention de cette information est trop élevée : le fait que les valeurs des paramètres soient instanciées dans un intervalle rend encore plus difficile l'obtention de la diversité demandée par le contrôleur. Cette alternative pourrait être employée si le but était uniquement de comprendre l'effet des paramètres sur l'EA, et non pas de le contrôler.

3.5 Conclusion du chapitre

Dans ce chapitre nous avons proposé une méthode qui crée automatiquement une abstraction des paramètres de l'AE. Tous les paramètres sont manipulés de la même manière, et leur contrôle est effectué au moyen d'un paramètre simple, qui ajuste l'équilibre entre l'exploration et l'exploitation. L'importance de cette abstraction se fonde sur un facteur humain : il est en effet beaucoup plus facile d'utiliser un paramètre simple et intuitif qu'un ensemble de paramètres méconnus. Notre but était de créer une méthode capable de fonctionner avec n'importe quel paramètre de manière abstraite, facile à intégrer dans un AE, et qui aide l'utilisateur à économiser du temps sur le processus de compréhension de l'effet des paramètres sur l'algorithme. À notre connaissance, aucun effort précédant n'avait réellement été conduit afin de créer une telle abstraction générale des paramètres des AEs.

Nous distinguons ici deux manières de mettre en œuvre le contrôle des paramètres. La plus utilisée consiste en définir une fonction qui calcule les valeurs des paramètres en accord avec une mesure de performance. Une approche différente, utilisée dans ce travail, se fonde sur l'idée de découvrir l'effet des paramètres en construisant un modèle à partir d'exemples obtenus du même algorithme.

Notre méthode est divisée en deux phases principales. La première est consacrée à la compréhension du fonctionnement de l'algorithme, et correspond à l'automatisation de la tâche de l'utilisateur lorsqu'il essaie d'établir les valeurs des paramètres. La deuxième phase correspond à la véritable exécution de l'algorithme, utilisant la connaissance obtenue avant.

Pour chaque paramètre nous considérons deux critères, communs à n'importe quel AE. Le premier est la qualité des solutions, qui est mesuré comme la moyenne de la fonction d'évaluation des individus. Le second est la diversité de population. Nous avons cherché à maximiser les deux critères afin de conserver un compromis entre ces deux objectifs et d'identifier les combinaisons de valeurs des paramètres correspondants à la frontière de Pareto. Étant donné que la forme des fonctions qui expriment la diversité et la qualité est inconnue, nous avons utilisé des contrôleurs flous pour les modéliser. La frontière de Pareto ainsi obtenue correspond à l'ensemble des combinaisons de valeurs de paramètres qui produisent les meilleurs compromis entre l'exploration et l'exploitation.

Pendant la première phase, nous analysons les principaux problèmes que nous avons rencontrés en collectant les exemples, à savoir la dimensionalité, l'inertie et le bruit. Nous avons proposé quelques mécanismes pour les atténuer. L'apprentissage est présenté comme une boîte noire à l'utilisateur, qui doit seulement ajouter deux méthodes simples pour intégrer le contrôleur dans son algorithme. Néanmoins, un utilisateur plus intéressé peut

tirer partie de la connaissance acquise afin de comprendre le comportement interne de l'algorithme.

Pendant la deuxième phase, plusieurs stratégies pour faire varier la diversité ont été comparées. Nous pouvons distinguer deux comportements extrêmes : des stratégies qui recherchent un niveau idéal de diversité, et d'autres qui favorisent une oscillation continue entre exploration et exploitation. Une comparaison expérimentale de ces stratégies a été présentée. Deux stratégies, MX et CD, sont ressorties de cette comparaison. MX est basée sur l'oscillation et CD sur la préservation d'un niveau stable de diversité. Bien que toutes les deux aient obtenu des résultats similaires, CD semble plus intéressante en raison de sa simplicité.

Notre approche a été testée sur 38 instances différentes du QAP, utilisant un AE avec trois opérateurs, dont les taux d'application étaient contrôlés. Nous avons également proposé une nouvelle mesure de diversité pour des codages de permutations, utilisé dans le QAP.

L'inconvénient principal de notre méthode est le temps requis pour collecter les exemples nécessaires afin d'établir le modèle. Dans nos expériences, 55% des générations sont consacrées à cette tâche. Cependant, il faut remarquer que cette méthode remplace le travail de l'utilisateur en essayant d'obtenir une combinaison appropriée de paramètres. Nous avons présenté quelques extensions de la méthode, destinées à réduire ce coût, qui peuvent guider les futurs travaux sur ce sujet.

D'autres directions peuvent inclure une méthode pour identifier les paramètres qui n'ont pas d'effet prépondérant sur l'exécution de l'AE, afin de les éliminer du contrôle et d'alléger la phase d'apprentissage. La parallélisation du calcul, ou le stockage des résultats d'une exécution à l'autre peuvent aussi être considérés afin de diminuer le temps d'apprentissage.

Parmi les extensions possibles de ce travail, on peut également inclure la recherche d'autres stratégies, en essayant d'identifier les plus simples et les plus performantes. On peut également penser à l'application de cette approche sur d'autres problèmes utilisant différents opérateurs et paramètres afin de valider la généralité de notre méthode. Il est aussi possible d'explorer d'autres paramètres tels que ceux des opérateurs locaux de recherche, la pression de sélection, ou la taille de population. Naturellement, puisque notre approche est basée sur la mesure de la diversité et de la qualité, des paramètres qui modifient la fonction de qualité ne pourraient pas être contrôlés par notre méthode.

Chapitre 4

Contrôle dynamique de paramètres

Sommaire

4.1 Introduction

Lorsqu'on contrôle les paramètres des AEs, nous pouvons distinguer fondamentalement deux approches. La première, vue dans le chapitre précédent, consiste à modéliser le comportement de l'AE en utilisant différentes valeurs de paramètres, souvent pendant une phase d'apprentissage.

Un exemple de cette première approche est la méthode APGAIN [Wong *et al.*, 2003], où la recherche est divisée en plusieurs époques, et chaque époque est à son tour divisée en deux périodes. La première période est consacrée à la mesure de la performance des opérateurs, qui sont appliqués aléatoirement. Pendant la deuxième période, les opérateurs sont appliqués en accord avec une probabilité proportionnelle à la performance observée précédemment. Trois valeurs (bas, moyen et haut) sont considérées pour chaque paramètre, et ajustées en les déplaçant vers la valeur la plus réussie. Un mécanisme de diversification, qui pénalise les individus similaires, est intégré dans la fonction d'évaluation. Environ un quart des générations est consacré à la première période (d'apprentissage), ce qui pourrait être nocif s'il y a des opérateurs destructifs.

Une deuxième approche consiste à fournir un contrôle en négligeant l'aspect de modélisation. En général, ces contrôleurs règlent les taux d'application des opérateurs en fonction de leurs performances passées.

Un exemple de cette deuxième approche est l'Adaptive Pursuit (AP) [Thierens, 2007], qui vise à ajuster les probabilités des opérateurs selon leurs performances, mesurées normalement comme l'amélioration de la qualité pendant les applications précédentes. Cette méthode peut adapter rapidement ces probabilités afin de récompenser les opérateurs les plus réussis. AP ne se soucie pas de comprendre le comportement de l'algorithme et se focalise immédiatement sur les meilleures valeurs, afin d'augmenter la performance. Il faut remarquer, cependant, que les algorithmes qui sont uniquement basés sur l'amélioration de la qualité peuvent subir convergence prématurée.

En général, on peut concevoir ce deuxième type de contrôleur comme un mécanisme de Sélection Adaptative d'Opérateurs (SAO) [Davis, 1989; Fialho *et al.*, 2008], constitué de deux parties où la première est consacrée à l'évaluation de la performance des opérateurs, et la deuxième à la sélection des opérateurs qui seront appliqués à chaque instant.

Dans ce chapitre, nous étudions une combinaison de ces deux parties afin de tirer bénéfice de leurs forces complémentaires, fournissant un contrôl abstrait et original des opérateurs des AEs. Notre contrôleur, appelé *Compass*, mesure les variations de la diversité et de la qualité de la population, produites par l'application d'un opérateur, ainsi que son temps d'exécution. Un paramètre de contrôle unique (Θ) permet d'ajuster le niveau de Exploration versus Exploitation (EvE), déterminant les taux d'application associés aux opérateurs. Les idées principales du chapitre précédent sont reprises, à savoir : l'abstraction du contrôle, l'apprentissage du comportement de l'AE, la prise en compte des deux objectifs principaux (maximiser la diversité et la qualité) et une architecture qui simplifie son intégration à l'AE. La méthode décrite dans ce chapitre a été presentée dans [Maturana and Saubion, 2008a; Maturana *et al.*, 2009a].

Nous expérimentons notre approche sur la résolution du problème de satisfaction booléenne SAT en le comparant à d'autres méthodes de contrôle adaptatives. Ensuite,

nous explorons une manière d'améliorer notre contrôleur, en intégrant un mécanisme plus sophistiqué pour effectuer la sélection des opérateurs.

Le chapitre est organisé comme suit. La section 4.2 expose les fondements de la *Sélection Adaptative d'Opérateurs* et les détails du contrôleur Compass. La section 4.3 décrit le cadre expérimental que nous avons utilisé, et la section 4.4 discute les résultats. Les sections 4.5, 4.6 et 4.7 présentent la description, le cadre expérimental et les résultats des améliorations de cette méthode. Finalement, des conclusions et des futures directions sont proposées dans la section 4.8.

4.2 Description de la méthode

4.2.1 Sélection Adaptative d'Opérateurs

Étant donné un ensemble d'opérateurs, le problème de choisir lequel sera appliqué dans la prochaine étape de recherche peut être perçu comme un problème d'optimisation. Nous voulons choisir les meilleurs opérateurs, cependant, ce choix "optimal" est inconnu, en raison de la nature dynamique des AEs.

La figure 4.1 illustre le schéma général de la *Sélection Adaptative d'Opérateurs* (SAO), et met en avant ses deux composants principaux : l'*Affectation de Crédits (AC)*, qui évalue l'exécution de chaque opérateur en fonction de son impact sur le processus de recherche ; et la *Sélection d'opérateurs (SO)*, qui choisit parmi les différents opérateurs en se basant sur les récompenses reçues. Entre les deux, le *Registre de Crédit* stocke les mesures de peformance calculées par l'*Affectation de Crédits* afin d'être utilisées par la *Sélection d'Opérateurs* .

L'implémentation de la *Sélection Adaptative d'Opérateurs* , vue en tant que boîte noire, est semblable à celle que nous avons utilisée dans le chapitre précédent (voir figure 3.7), et facilite l'intégration du contrôleur dans n'importe quel AE. La différence ici est la séparation explicite des deux tâches à l'intérieur du contrôleur.

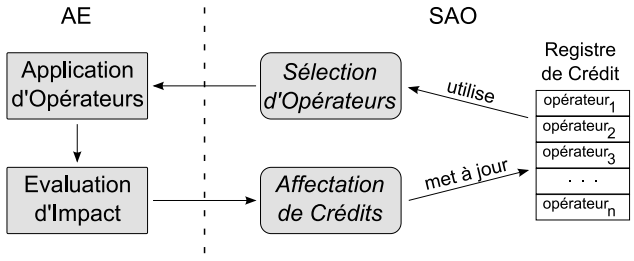

FIG. 4.1 – Schéma générale d'une SAO

Deux constatations principales doivent être faites dans la sélection de l'opérateur. D'une part, il est souhaitable d'appliquer les "bons" opérateurs, ceux qui ont démontré leur qualité dans un passé récent. D'autre part, des opérateurs moins populaires doivent être appliqués de temps en temps, afin de découvrir ceux qui pourraient être devenus plus adéquats à l'état actuel de la recherche. Cette situation est en effet un dilemme typique

entre l'exploration et l'exploitation, non pas au niveau de l'AE, mais au niveau de la *Sélection Adaptative d'Opérateurs* .

Nous allons décrire maintenant les principes de base des méthodes de la *Sélection d'Opérateurs* et de l'*Affectation de Crédits* .

Affectation de Crédits (AC)

À partir des premiers travaux sur les *Sélections Adaptatives d'Opérateurs* vers la fin des années 80 [Davis, 1989], plusieurs méthodes ont été proposées dans la littérature pour affecter des crédits (ou des récompenses) aux opérateurs de variation. Leur différence se situe dans la façon dont ils mesurent la qualité d'un opérateur en fonction du résultat de son application.

Traditionnellement, la plupart des méthodes sont focalisées sur la qualité de la population [Thierens, 2007; Fialho *et al.*, 2008]. Cette qualité peut être mesurée comme la différence entre celle des fils par rapport à celle de leurs parents [Lobo and Goldberg, 1997; Tuson and Ross, 1998], en choisissant le meilleur [Davis, 1989], ou l'individu médian [Julstrom, 1995]. Il peut être aussi intéressant d'utiliser des outils statiques plus sophistiqués, comme dans [Whitacre *et al.*, 2006], afin de détecter les valeurs de haute qualité et de favoriser les opérateurs qui ont le plus grand impact à un certain moment spécifique. On peut également considérer la généalogie de l'individu (i.e., l'évaluation des ancêtres), comme dans [Lobo and Goldberg, 1997; Tuson and Ross, 1998].

Une fois que les critères d'évaluation ont été établis, l'affectation de crédits peut être effectué d'une façon plus ou moins sophistiquée, par exemple en utilisant des fenêtres pour obtenir des valeurs moyennes ou extrêmes. L'objectif principal est de proposer un système de récompenses le plus générique et le plus robuste possible. Ce système de récompense devrait être également assez sensible aux changements de l'état des opérateurs, afin de permettre au mécanisme de contrôle de s'adapter de façon réactive. Ce schéma peut être vu comme une forme d'apprentissage dynamique de la qualité produite par les opérateurs pendant la recherche.

Sélection d'Opérateurs (SO)

La sélection d'opérateurs est basée sur les crédits précédemment stockés dans le registre de drédit, afin de choisir les opérateurs à employer lors de la prochaine étape de résolution.

La façon la plus directe (et ainsi la plus commune) de réaliser la sélection d'opérateurs est par *Probabilité Proportionnelle (PP)* [Goldberg, 1990a; Lobo and Goldberg, 1997; Barbosa and Sá, 2000]). Grossièrement, le processus choisit les opérateurs avec une probabilité proportionnelle à une mesure de qualité empirique (par exemple, la moyenne des récompenses reçues). Une probabilité minimale (P_{min}) est habituellement utilisée, afin d'avoir un minimum d'exploration et pour empêcher la disparition des opérateurs qui pourraient s'avérer bons plus tard. Clairement, si un certain opérateur ne reçoit aucune récompense pendant quelque temps, il est prévu que sa récompense diminue jusqu'à P_{min}.

Adaptive Pursuit (AP) [Thierens, 2005], une méthode proposée à l'origine pour l'apprentissage d'automates, implémente une stratégie qui choisit toujours le meilleur

opérateur. Un paramètre β, défini par l'utilisateur, commande le niveau de "gloutonnerie" de la stratégie, i.e., la rapidité avec laquelle le meilleur opérateur augmente sa probabilité, tandis que celles du reste diminuent.

4.2.2 Compass

La méthode Compass est un mécanisme de *Sélection Adaptative d'Opérateurs* dont l'intérêt primordial réside dans sa méthode d'*Affectation de Crédits* .

Afin de caractériser les opérateurs, Compass stocke les valeurs de variation de la diversité, variation de la qualité et le temps d'exécution, produites lors de l'application de chaque opérateur, dans des fenêtres glissantes de taille τ. Étant donné un opérateur $i \in [1 \dots k]$ et un nombre de générations t, soit d_{it}, q_{it}, T_{it} la variation moyenne – des valeurs stockées dans la fenêtre glissante – de la diversité de la population, la variation moyenne de la qualité, et temps d'exécution moyen de i au cours des dernières applications d'un opérateur. Au début de l'exécution, tous les opérateurs peuvent être appliqués avec la même probabilité.

Nous définissons un vecteur $o_{it} = (d_{it}, q_{it})$ pour caractériser les effets de l'opérateur sur la population en termes de la variation de la qualité et de la diversité (axe ΔD et ΔQ dans la figure 4.2). Comme la qualité et la diversité sont des objectifs plutôt opposés, la plupart des vecteurs seront placés dans les quadrants II (où la qualité augmente et la diversité diminue) et IV (où la diversité augmente et la qualité diminue), comme illustré dans la figure 4.2.a.

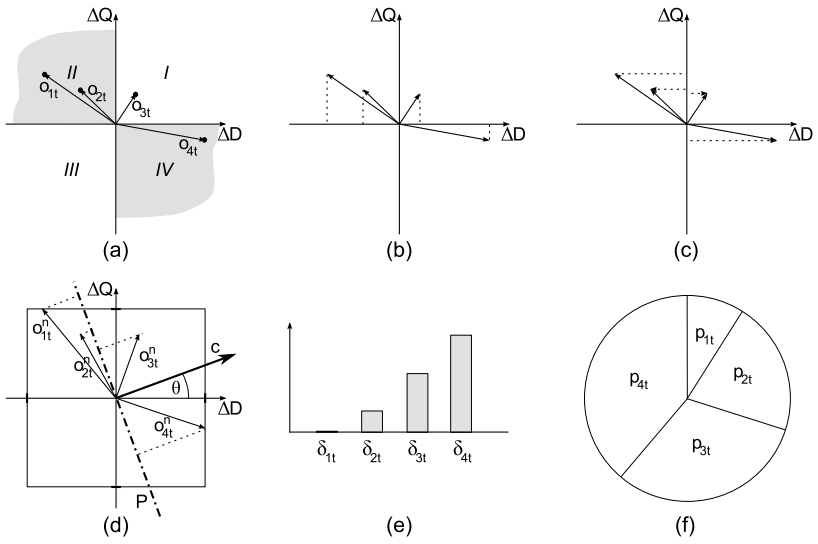

FIG. 4.2 – Méthode Compass. (a) points (d_{it}, q_{it}) et vecteurs correspondants o_{it}, (b) classification basée en qualité, (c) classification basée sur la diversité, (d) approche proposée, (e) valeurs de δ_{it}, (f) probabilités finales

Les algorithmes qui ne considèrent que l'amélioration de la qualité ajustent les probabilités des opérateurs en utilisant uniquement la projection du vecteur o_{it} sur l'axe des

ordonnées (lignes pointillées dans la figure 4.2.b). Par contre, si seule la diversité est prise en compte, les mesures sont calculées comme la projection sur l'axe des abscisses (figure 4.2.c).

Notre but est de considérer ces deux critères simultanément en choisissant une direction de recherche qui sera exprimée par un vecteur c (défini par son angle $\Theta \in [0, \frac{\pi}{2}]$) qui caractérise également le plan orthogonal P (figure 4.2.d).

Comme les mesures de diversité et de qualité ont normalement des échelles différentes, elles sont normalisées de la façon suivante :

$$d_{it}^n = \frac{d_{it}}{max_i\{|d_{it}|\}} \qquad \text{and} \qquad q_{it}^n = \frac{q_{it}}{max_i\{|q_{it}|\}} \tag{4.1}$$

Nous avons ainsi les vecteurs $o_{it}^n = (d_{it}^n, q_{it}^n)$. Les récompenses sont alors basées sur la projection des vecteurs o_{it}^n sur c, i.e., $|o_{it}|cos(\alpha_{it})$, α_{it} étant l'angle entre o_{it} et c. Une valeur de Θ proche de 0 encouragera l'exploration, alors qu'une valeur proche de $\frac{\pi}{2}$ favorisera l'exploitation. De cette façon, la gestion des taux d'application est abstraite par l'angle Θ, qui guide la direction de la recherche comme l'aiguille d'un compas montre le nord.

Les projections sont transformées en valeurs positives en les soustrayant la plus petite d'entre elles et en les divisant par le temps d'exécution, afin de favoriser les opérateurs les plus rapides (figure 4.2e). Ceci donne la valeur de performance δ_{it} :

$$\delta_{it} = \frac{|o_{it}^n|cos(\alpha_{it}) - min_i\{|o_{it}^n|cos(\alpha_{it})\}}{T_{it}}$$

La sélection d'opérateurs est faite proportionnellement aux valeurs de δ_{it} plus une constante ξ_t, qui assure que la probabilité la plus petite est égale à P_{min}, afin d'empêcher la disparition des opérateurs les moins performants (figure 4.2.f).

$$p_{it} = \frac{\delta_{it} + \xi_t}{\sum_{i=1}^{k} \delta_{it} + \xi_t}$$

4.3 Cadre expérimental

4.3.1 Algorithme évolutionnaire

Pour nos expériences, nous nous concentrons sur l'utilisation des AEs pour la résolution de problèmes combinatoires. Cette fois nous avons choisi le problème SAT, detaillé dans la section 1.4.1.

La première raison de ce choix est que SAT est probablement le problème combinatoire le plus connu, puisqu'il a été le premier à être prouvé NP complet [Cook, 1971] et qu'il a été employé pour coder des problèmes issus de nombreux domaines d'application différents. La deuxième raison est qu'il existe une bibliothèque impressionnante d'instances [Hoos and Stützle, 2000] et que leur difficulté a été profondément étudiée avec plusieurs résultats théoriques intéressants. Ceci nous permet de choisir différentes instances avec des paysages de recherche ayant diverses propriétés.

Nous utilisons une population codée par des nombres binaires (voir section 1.3.2), où chaque bit répresente la valeur de verité (vraie ou fausse) des variables du problème. L'AE applique un opérateur à chaque génération. La fonction d'évaluation considère le nombre de clauses satisfaites par un individu, dans le but de les maximiser. La diversité est classiquement calculée comme la distance de Hamming ou l'entropie [Lardeux *et al.*, 2006].

Dans le but d'évaluer notre approche de contrôle, nous la comparons avec Adaptive Pursuit (AP) [Thierens, 2007] et APGAIN [Wong *et al.*, 2003]. AP est représentative d'une classe de contrôleurs qui considèrent l'amélioration de la qualité en tant que critère de guidage tandis qu'APGAIN est représentative des méthodes qui essayent d'apprendre et de modéliser le comportement des opérateurs. En plus, nous avons également inclus le Choix Uniforme (CU) des opérateurs comme base de la comparaison. Afin de vérifier la robustesse de notre méthode, nous présentons 13 instances différentes de la bibliothèque SATLIB [Hoos and Stützle, 2000], mélangeant des problèmes de tailles et des familles différentes, y compris des instances générées aléatoirement, coloration de graphes, de planification logistique et des problèmes de "blocks world". La table 4.1 liste les instances utilisées, ainsi que leur famille et leur nombre de varibles et de clauses.

Tab. 4.1 – Instances SAT utilisées

Problème	Sat ?	# Vars.	# Clauses	Famille
4blocks	Oui	758	47820	Problème de Blocks World
aim	Oui	200	320	3-SAT Aléatoire
f1000	Oui	1000	4250	3-SAT Aléatoire
CBS	Oui	100	449	Controlled Backbone
Flat200	Oui	600	2237	Flat Graph Coloring
logistics	Oui	828	6718	Planification logistique
medium	Oui	116	953	Générée aléatoirement
Par16	Oui	1015	3310	Parity Learning Problem
sw100-p0	Oui	500	3100	Morphed Graph Coloring
sw100-p1	Oui	500	3100	Morphed Graph Coloring
Uf250	Oui	250	1065	Phase Transition Region
Uuf250	Non	250	1065	Phase Transition Region

Afin d'observer l'effet de la taille de population sur la performance des contrôleurs, nous avons exécuté des expériences avec des populations de taille 3, 5, 10 et 20. Des exécutions de 10.000 générations ont été effectuées, afin d'observer le comportement à long terme des contrôleurs.

Opérateurs

L'objetif de ce travail est de créer une abstraction des opérateurs, sans augurer de leur qualité, et de comparer des contrôleurs. Notre but n'est pas pour l'instant de développer

des AEs efficaces pour la résolution du problème SAT (à l'heure actuelle les solveurs CDCL (*Conflict-Driven Clause Learning*, minisat) sont les plus performants [Eén and Sörensson, 2004]). L'idée est également d'utiliser des opérateurs non standards, dont l'effet sur la diversité et la qualité est *a priori* inconnu. Par conséquent, nous proposons les suivants six opérateurs avec différents mécanismes, plus ou moins spécialisés par rapport au problème SAT.

Croisement en un point choisit aléatoirement deux individus et les croise en une position aléatoire. Exclusivement dans cet opérateur, le meilleur fils remplace le parent le moins bon.

Contagion choisit aléatoirement deux individus, et les variables dans les clauses fausses du plus mauvais d'entre eux sont remplacées par les valeurs correspondantes à celles du meilleur individu. Le individu modifié remplace le plus mauvais des parents.

Descente stricte choisit aléatoirement un individu et vérifie tous les voisins en flippant une variable, et se déplacent vers le meilleur, tant qu'il existe des améliorations possibles. Le individu ainsi obtenu remplace son parent.

Tunneling choisit aléatoirement un individu et flippe des variables sans diminuer le nombre de clauses vraies, en accord avec une liste taboue de longueur égale à $\frac{1}{4}$ du nombre de variables. Le individu ainsi obtenu remplace son parent.

Badswap choisit aléatoirement un individu et flippe toutes les variables qui apparaissent dans des clauses fausses. Le individu ainsi obtenu remplace son parent.

Wave choisit aléatoirement un individu et choisit la variable qui apparaît dans le nombre le plus élevé de clauses fausses et dans le nombre minimum de clauses seulement soutenues par elle [1], et la flippe. Le même processus est répété au plus $\frac{1}{2}$ fois le nombre de variables. Le individu ainsi obtenu remplace son parent.

4.3.2 Paramétrage de Compass

La taille de la fenêtre glissante, τ, a une valeur de 100, et P_{min} vaut $\frac{1}{3k}$ (voir la section 4.2.2). Des exécutions préliminaires ont suggérée qu'un valeur de Θ près de $\frac{\pi}{4}$ produise de bons résultats, raison pour laquelle nous utiliserons les valeurs 0.2π, $O.25\pi$ et 0.3π. Chaque exécution, constituée d'une instance spécifique de problème, une taille de population, un contrôleur et une valeur de Θ (uniquement pour Compass), a été répétée 30 fois pour obtenir des résultats statistiques significatifs. Les paramètres d'AP et d'APGAIN ont été fixés aux valeurs publiées ou bien réglées pour obtenir de bonnes performances. En accord avec les notations utilisées dans [Thierens, 2007; Wong *et al.*, 2003], les valeurs pour AP sont les suivantes : $\alpha = 0.8$, $\beta = 0.8$, $P_{min} = \frac{1}{2k}$. Pour APGAIN, ce sont les suivantes : $v_L = 0$, $v_U = 1$, $\delta = 0.05$, $\sigma = 700$, $\rho = \frac{\sigma}{4}$, $\xi = 10$, $\phi = 0.045$ (ce qui produit environ 10% de réévaluations).

[1] Ici on entend qu'une variable "soutient" une clause lorsque la clause est vraie uniquement à cause de cette variable.

4.4 Résultats et discussion

Le nombre moyen de clauses fausses obtenues sur 30 exécutions est indiqué dans la table 4.2. Des comparaisons ont été faites utilisant un test T de Student avec un niveau de confiance de 95%. Les valeurs sont marquées en **gras** lorsque Compass surpasse le CU, et en *italiques* lorsque le CU est meilleur que Compass. Aucune modification de police signifie que les résultats sont statistiquement indistingables. Les cellules sont marquées en gris quand Compass surpasse AP, en noir lorsque AP surpasse Compass. Des cellules blanches indiquent que les résultats sont statistiquement indistingables. Compass a surpassé APGAIN dans tous les cas, sauf dans les cas soulignés, où les résultats sont indistingables. Les temps d'exécution moyens d'AP, APGAIN et Compass, par rapport à ceux du CU, sont montrés dans la colonne la plus à droite de la table. Le nombre total de clauses de chaque problème apparaît en bas de la table. Dorénavant nous appellerons *C.2, C.25, C.3* à Compass avec des valeurs de Θ respectivement de 0.20π, 0.25π et 0.30π.

Tab. 4.2 – Nombre moyen de clauses fausses et comparaison de temps d'exécution

taille pop.	contrôleur	4-blocks	aim	f1000	CBS	flat200	logistics	medium	par16	sw100-0	sw100-1	uf250	uuf250	temps
	CU	12.9	3.2	52.6	5.4	38.4	16.7	7.7	124.2	23.2	9.4	8.3	11.7	1.00
	AP	7.5	2.1	37.7	3.4	19.6	8.9	3.5	71.3	16.2	3.3	5.6	8.3	0.86
	APGAIN	11.8	3.3	51.6	5.0	27.5	14.0	5.4	109.2	20.6	6.0	8.8	10.8	0.97
3	C.2	**5.8**	**2.1**	**25.5**	**2.1**	**13.2**	**8.1**	**2.0**	**41.6**	**13.7**	**1.3**	**3.3**	**5.5**	0.88
	C.25	**6.4**	**1.6**	**26.7**	**2.3**	**11.7**	**8.1**	**2.0**	**38.4**	**13.4**	**1.8**	**3.6**	**5.9**	0.89
	C.3	**6.1**	**1.6**	**26.8**	**2.8**	**15.9**	**8.1**	**3.0**	**47.2**	**15.5**	**2.2**	**4.3**	**6.4**	0.73
	CU	13.8	3.3	61.4	7.6	34.6	16.5	7.9	126.2	24.6	8.3	11.2	14.0	1.00
	AP	8.9	3.0	47.4	4.8	23.3	11.3	4.9	88.2	18.4	4.5	8.3	10.2	0.88
	APGAIN	11.2	4.9	60.1	6.6	31.1	14.0	6.4	118.7	20.1	6.0	9.8	13.6	1.09
5	C.2	**6.2**	**2.2**	**27.5**	**3.0**	**15.5**	**9.1**	**2.6**	**45.0**	**15.0**	**2.8**	**4.6**	**6.7**	0.89
	C.25	**6.3**	**1.9**	**27.2**	**2.7**	**16.2**	**8.8**	**2.6**	**43.4**	**14.8**	**2.9**	**4.4**	**7.1**	0.91
	C.3	**7.8**	**2.5**	**36.5**	**4.2**	**20.0**	**9.0**	**3.5**	**66.7**	**16.8**	**3.4**	**5.9**	**10.3**	0.80
	CU	13.8	3.3	54.2	6.3	28.8	15.5	7.0	110.0	19.4	6.1	10.1	12.2	1.00
	AP	9.9	3.9	55.2	5.0	26.3	12.9	5.7	98.6	18.1	5.9	9.3	12.2	1.00
	APGAIN	11.7	4.8	66.0	5.8	30.4	16.3	6.2	120.4	18.8	6.8	11.3	13.9	0.62
10	C.2	**8.3**	3.5	**44.9**	**3.9**	**21.9**	**11.3**	**4.5**	**72.5**	**17.9**	**4.3**	**7.5**	**10.4**	0.92
	C.25	**8.0**	2.9	**42.7**	**4.4**	**23.1**	**11.2**	**4.2**	**72.6**	**17.8**	**4.6**	**7.2**	**9.5**	0.83
	C.3	**9.1**	3.7	**49.3**	5.7	**24.7**	**11.5**	**5.1**	**96.8**	19.1	5.9	9.1	12.6	0.78
	CU	13.8	2.8	42.0	4.5	23.1	13.8	5.2	88.5	16.5	3.4	6.4	10.3	1.00
	AP	9.1	2.4	54.1	4.9	26.0	14.6	5.3	102.6	17.2	5.7	8.4	11.3	1.28
	APGAIN	11.3	4.3	68.0	5.5	30.0	17.8	6.2	120.1	18.7	6.3	11.1	13.7	2.38
20	C.2	**9.1**	*3.5*	*55.2*	4.8	*26.0*	13.3	5.2	90.4	*18.9*	*6.2*	*8.1*	*12.5*	0.92
	C.25	**9.2**	3.3	*53.2*	4.8	*25.2*	13.3	4.8	90.6	*17.8*	*6.2*	*8.7*	*13.2*	0.93
	C.3	**9.4**	*3.9*	*58.6*	4.9	*27.5*	13.4	6.2	*99.2*	*18.7*	*6.1*	*10.4*	*12.4*	0.88
N. clauses		47820	320	4250	449	2237	6718	953	3310	3100	3100	1065	1065	

Le nombre moyen de générations nécessaires pour atteindre les meilleures valeurs varie entre 1.000 et 7.000. Le nombre de générations permises (10.000) semble donc suffisant pour assurer une comparaison juste entre les contrôleurs. Naturellement, les résultats ne

sont pas concurrentiels avec les solveurs SAT spécifiques car nous n'employons pas les meilleurs opérateurs ni essayons d'optimiser les nôtres. Notre but ici est plutôt de souligner les différences entre les contrôleurs, en simplifiant l'AE, afin de faciliter la compréhension.

La domination de Compass, et particulièrement de C.25 sur CU, AP et APGAIN est manifeste, en particulier en utilisant de petites populations. Des résultats similaires sont produits avec C.2, et, dans une certaine mesure, avec C.3. Comme mentionné précédemment, une valeur $\Theta = 0.25\pi$ fonctionne bien avec la plupart des problèmes.

Les petites populations perdent facilement leur diversité, ainsi le contrôle de diversité est un point critique. APGAIN le fait en pénalisant les individus similaires. Cependant, quand tous les individus sont identiques, cette pénalisation n'est plus efficace. Il semble que dans la pratique, la diversité est la plupart du temps induite par la première période d'APGAIN (évaluation d'opérateurs). AP contrôle la diversité en définissant un taux minimum d'application égal à $\frac{1}{2k}$. Cette valeur pourrait être excessive si les opérateurs faisaient la plupart du temps l'exploitation. Une plus petite valeur de $\frac{1}{3k}$, utilisée dans Compass, accorde au contrôleur une plus grande liberté pour équilibrer l'EvE.

On pourrait se demander pourquoi les petites populations fournissent de meilleurs résultats que les plus grandes. Ceci est probablement dû aux opérateurs, qui ont été inspirés par des heuristiques de recherche locale : comme ils sont appliqués sur le même individu plus répétitivement dans les petites populations que dans les grandes, on obtient de meilleurs résultats. D'une façon surprenante, le CU est tout à fait concurrentiel à mesure que la taille de la population augmente. Il semble que le fait d'appliquer des opérateurs qui produisent des "mauvais" individus permet à ceux-ci d'échapper des optimums locaux. Néanmoins, cette pratique est bénéfique seulement si la population est assez grande pour garder les bons éléments simultanément.

Les temps d'exécution d'AP et d'APGAIN sont plus courts que ceux du CU dans de petites populations. Compass a des temps d'exécution courts et stables pour différentes tailles de population. Ce point est intéressant parce qu'il signifie que l'effort consacré à effectuer le contrôle réduit le temps d'exécution global.

Du point de vue de l'implémentation, nous avons constaté que Compass et AP étaient plus indépendants de la logique de l'AE qu'APGAIN, qui introduit son mcanisme de contrôle de diversité dans la fonction d'évaluation de l'AE. AP et Compass sont placés dans une couche de contrôle, indépendante de l'AE.

Application d'opérateurs

Les taux d'application des opérateurs sont mis à jour à chaque génération. Un phénomène intéressant, observé pendant des expériences précédentes, est le déplacement des points (d_{it}^n, q_{it}^n) dans le graphique $\Delta D/\Delta Q$ pendant l'exécution. Considérons par exemple que l'angle Θ soit fixé à $\frac{\pi}{4}$, de telle façon qu'une importance égale est donnée à ΔQ et à ΔD. Au début de la recherche, juste après que la population ait été aléatoirement créée, la diversité de la population est élevée et la qualité est basse, ainsi il est facile pour la plupart des opérateurs de se placer dans le quadrant II. Après quelques générations, la population commence à converger vers un certain optimum. Puisque l'amélioration devient donc difficile, les points dans le quadrant II, correspondants aux opérateurs exploiteurs, se

déplacent vers le bas. Lorsque les améliorations de qualité de la zone actuelle sont épuisées, les opérateurs exploiteurs sont dépassés par les explorateurs, entraînant une transition de la recherche vers la diversification, ce qui provoque l'abandon de la zone où se trouve l'optimum visité. L'implémentation de la visualisation des points pendant la recherche peut s'avérer un outil convenable pour comprendre le comportement des opérateurs et pour l'amélioration de l'AE.

Réglage du paramètre Θ

On pourrait s'interroger sur l'effet de Θ sur la recherche et pourquoi la valeur $\frac{\pi}{4}$ semble marcher mieux dans tous les cas. La figure 4.3 montre l'amélioration moyenne de la qualité du meilleur individu de la population sur 50 exécutions dans l'instance *par16-1* en utilisant les valeurs $0, 0.1\pi, .02\pi, 0.25, \pi 0.3\pi, 0.4\pi$ et 0.5π.

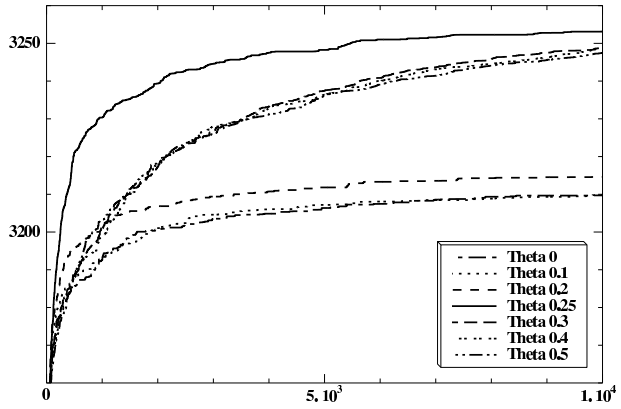

FIG. 4.3 – Effet des différentes valeurs de Θ : amélioration de qualité sur l'instance *par16*

On peut noter qu'il existe trois comportements bien différenciés. $\Theta = 0.25\pi$ produit les meilleurs résultats tandis que les valeurs inférieures produisent de mauvaises qualités dues à l'exploration excessive. D'autre part, les valeurs de Θ supérieures à 0.25π induisent une recherche de meilleure qualité, mais lente à cause de la difficulté à quitter les optimums locaux. Même une petite déviation de l'angle 0.25π produit l'application systématique des opérateurs soit "explorateurs", soit "exploiteurs". Ceci suggère que l'angle Θ doit rester fixe à la valeur 0.25π, sauf éventuellement pour des courtes périodes.

4.5 Amélioration de la *Sélection Adaptative d'Opérateurs*

En regardant le schéma SAO = AC + SO, on peut remarquer que Compass possède en effet un mécanisme de Sélection d'Opérateurs assez simple. Quelques auteurs considèrent que le schéma de probabilité proportionnelle induit une convergence trop lente des probabilités d'application des opérateurs [Thierens, 2005; Fialho *et al.*, 2008]. Ceci impliquerait

que l'on dédie trop de temps à la diversification (au niveau de la sélection d'opérateurs), sans se focaliser sur les meilleurs.

La section suivante présente une amélioration du Compass, en replaçant le mécanisme de *Sélection d'Opérateurs* par une méthode proposée dans [Fialho *et al.*, 2008], appelée *Extreme-Dynamic Multi-Armed Bandit (Ex-DMAB).*

4.5.1 Ex-DMAB

Les deux ingrédients de la méthode de *Sélection Adaptative d'Opérateurs* proposée dans [Fialho *et al.*, 2008] sont une *Affectation de Crédits* basée sur des valeurs extrêmes d'amélioration de qualité, et un mécanisme de *Sélection d'Opérateurs* basé sur le paradigme Multi-Armed Bandit (MAB).

Cette approche propose de considérer la valeur extrême d'amélioration de la qualité dans les κ dernières applications plutôt que la moyenne. Ceci est en accord avec [Whitacre *et al.*, 2006], et est implémentée en mesurant la différence de la qualité entre le parent et le fils produit par l'application d'un opérateur. Le maximum de telles améliorations (obtenu à partir d'une fenêtre glissante de taille κ) est utilisé en tant que récompense pour chaque opérateur.

En ce qui concerne le mécanisme de *Sélection d'Opérateurs* , l'idée explorée, d'abord proposée dans [Da Costa *et al.*, 2008], est que dans le choix d'un opérateur les meilleurs opérateurs doivent s'appliquer autant que possible, sans oublier pourtant le reste, dans l'éventualité que l'un d'entre eux devienne superieur à l'actuel meilleur. Ce dilemme a été intensivement étudié dans le cadre de la *théorie des jeux*, notamment dans le cadre des Multi-Armed du Bandits (MAB). Parmi les variantes existantes de MAB, c'est l'*Upper Confidence Bound (UCB)* [Auer *et al.*, 2002] qui a été choisi.

Plus formellement, l'algorithme d'UCB fonctionne comme suit : chaque opérateur de variation est observé en tant qu'un bras (*arm*) d'un problème MAB. La stratégie choisit toujours l'opérateur le plus efficace, donné par l'expression :

$$MAB_{o,t} = r_{o,t} + C\sqrt{\frac{\log \sum_k n_{k,t}}{n_{o,t}}} \tag{4.2}$$

où $r_{o,t}$ est la récompense obtenue par l'opérateur o au temps courant t, et $n_{o,t}$ est le nombre de fois où l'opérateur o a été appliqué jusqu'au temps t. Le premier terme encourage l'utilisation des meilleurs opérateurs, tandis que le deuxième favorise les opérateurs qui ont été appliqués moins souvent.

Autre aspect important est que la version originale de MAB est statique, alors que le scénario d'AOS est dynamique, car la qualité des opérateurs peut changer le long de la recherche. Le terme "explorateur" de l'expression 4.2 assure que tous les opérateurs seront utilisés un nombre infini de fois, ce qui permet au contrôleur de s'adapter au changements de performance des opérateurs. Néanmoins, l'évolution du classement des opérateurs pourrait prendre du temps avant de refléter la nouvelle situation.

Afin de s'adapter rapidement aux environnements dynamiques, l'intégration d'un test statistique qui détecte les changements de séries de données temporeles [Hartland *et al.*,

2006] a été proposé. Le test de *Page-Hinkley* (PH) [Page, 1954] détecte les changements dans les distributions des valeurs (e.g. lorsque le meilleur opérateur actuel ne l'est plus), pour réinitialiser les valeurs du MAB ($r_{o,t}$ et $n_{o,t}$) afin de redécouvrir rapidement le nouveau meilleur opérateur.

Le test de PH fonctionne comme suit : soit $\bar{r}_t$ la moyenne des récompenses $r_1, \dots r_t$ et soit e_t la différence $r_t - \bar{r}_t + \delta$, où δ est un paramètre de tolérance. Ce test considère la variable aléatoire $m_t = \sum_1^t e_i$. Lorsque la différence entre $M_t \equiv \max_{i \leq t} m_i$ et m_t dépasse un certain seuil γ, défini par l'utilisateur, le test de PH est déclenché.

La partie de MAB dans Ex-DMAB implique un méta-paramètre qui régle l'importance relative entre l'exploration et l'exploitation (C), alors que le test de PH implique deux paramètres, le seuil de détection γ et δ, qui renforce la robustesse du test dans des environnements qui varient lentement. Notons que, selon des expériences initiales de [Da Costa *et al.*, 2008], δ est maintenu fixe à la valeur 0.15. L'algorithme dynamique de MAB est appelé DMAB, et la combinaison complète de la *Sélection Adaptative d'Opérateurs* est appelée Ex-DMAB.

Ex-DMAB a été employé pour choisir dynamiquement parmi 5 opérateurs de mutation afin de résoudre le problème OneMax [Da Costa *et al.*, 2008], et également utilisé sur un autre problème unimodal : le long k-path [Fialho *et al.*, 2009]. Les expériences ont prouvé que l'utilisation des valeurs extrêmes plutôt que des moyennes est bénéfique. En outre, la technique de *Sélection Adaptative d'Opérateurs* a obtenu des résultats comparables à ceux d'Adaptive Pursuit (utilisant aussi l'amélioration extrême de qualité comme récompense), obtenant des performances proches de la stratégie déterministe optimale connue (trouvée au moyen de simulations de Monte Carlo).

4.5.2 ExCoDyMAB

Les forces et faiblesses de Compass et d'Ex-DMAB sont complémentaires : Compass mesure d'une manière holistique l'effet de l'application d'opérateurs sur la population, mais la règle de *Sélection d'Opérateurs* est plutôt rudimentaire, alors que Ex-DMAB a une façon efficace de choisir les opérateurs, mais son mécanisme d'*Affectation de Crédits* est probablement trop simpliste. Il semblait par conséquent normal de combiner les deux méthodes. Bien que la fusion des deux modules semble plutôt directe, quelques questions importantes doivent être explorées soigneusement :

- Compass emploie des fenêtres glissantes de taille τ dans l'*Affectation de Crédits* (appellés dorenavant W_1), en prenant une valeur unique de récompense comme sortie. Ex-DMAB considère les dernières κ valeurs (récompenses) du module d'*Affectation de Crédits* dans une fenêtre glissante (appellés W_2 par la suite). Devons-nous garder les deux fenêtres ou uniquement l'une d'elles ? Si nous n'en gardons qu'une, laquelle ?
- Un autre aspect au sujet des fenêtres glissantes est celui de déterminer quelle mesure doit être prise : l'algorithme devrait-il employer la valeur moyenne, l'extrême, ou tout simplement la valeur instantanée ? (Ce qui équivaut à ne pas employer de fenêtre). La valeur extrême s'est avérée bénéfique dans un problème unimodal [Fialho *et al.*, 2008], mais est-ce que ces résultats seront similaires dans un scénario complètement différent ?

– Finalement, il faut prendre en compte les autres méta-paramètres. Outre la taille et la mesure à prendre depuis W_1 et W_2, nous devons régler les paramètres C et γ du DMAB. Une façon de régler ces paramètres est d'essayer plusieurs configurations expérimentalement afin de trouver les valeurs les plus robustes.

La figure 4.4 montre l'intégration de Compass et Ex-DMAB. Chaque méthode possède ses composants d'*Affectation de Crédits* et *Sélection d'Opérateurs* , et la ligne en gras montre le point d'intégration : la sortie de l'*Affectation de Crédits* de Compass est insérée dans l'entrée de l'*Affectation de Crédits* de Ex-DMAB.

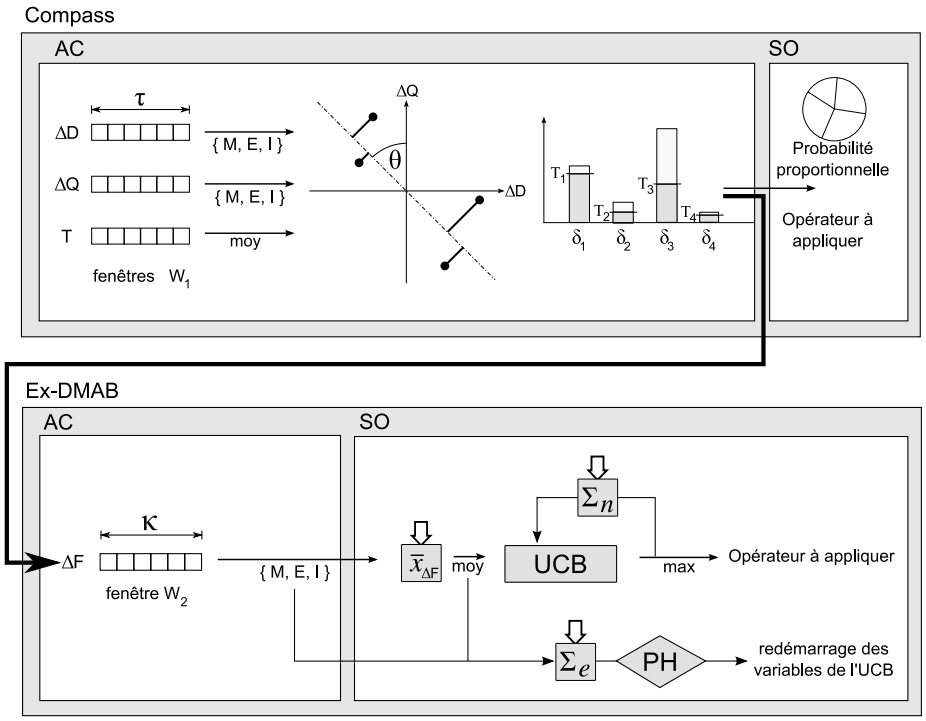

FIG. 4.4 – ExCoDyMAB : Intégration de Compass et Ex-DMAB

4.6 Cadre expérimental

4.6.1 Algorithme évolutionnaire

Nous avons enrichi l'ensemble d'instances de SAT employées dans l'étude précédente en ajoutant quelques exemples plus difficiles de la competition SAT 2006. La table 4.3 montre les instances utilisées ici, en précisant si elles sont satisfiables ou non, leur famille, et le nombre de variables et de clauses qu'elles possèdent. Chacune des 22 instances a été considérée pour la comparaison finale, mais seulement 7 d'entre elles ont été prises en considération pendant la définition de l'analyse de plate-forme et des méta-paramètres présentée dans la section 4.5.2. Ce sous-ensemble, identifié par une astérisque dans la table

4.3, a été choisi parmi les instances les plus difficiles avec des temps d'exécution suffisamment courts, afin de réduire le coût expérimental pour la définition de la plate-forme. En réglant les méta-paramètres sur un ensemble réduit d'exemples et en les appliquant sur des instances "non encore vues", nous avons pu vérifier la généralité des paramètres accordés.

Tab. 4.3 – Instances SAT utilisées

Problème	Sat ?	# Vars.	# Clauses	Famille
4blocks	Oui	758	47820	Problème de Blocks World
aim	Oui	200	320	3-SAT Aléatoire
f1000	Oui	1000	4250	3-SAT Aléatoire
CBS	Oui	100	449	Controlled Backbone
Flat200	Oui	600	2237	Flat Graph Coloring
logistics	Oui	828	6718	Planification logistique
medium	Oui	116	953	Générée aléatoirement
Par16	Oui	1015	3310	Parity Learning Problem
sw100-p0	Oui	500	3100	Morphed Graph Coloring
sw100-p1	Oui	500	3100	Morphed Graph Coloring
Uf250	Oui	250	1065	Phase Transition Region
Uuf250	Non	250	1065	Phase Transition Region
Color*	Non	1444	119491	Coloration d'échequier
G125*	Oui	2125	66272	Coloration de graphe
Goldb-heqc*	Non	5980	35229	Générée aléatoirement
Grieu-vmpc	Oui	729	96849	Générée aléatoirement
Hoons-vbmc*	Non	8503	25116	Générée aléatoirement
Schup	Non	14809	48483	Générée aléatoirement
Simon*	Non	2424	14812	Générée aléatoirement
Manol-pipe	Oui	14052	41596	Pipelined Machine Verification
Velev-eng*	Non	6944	66654	Pipelined Machine Verification
Velev-sss*	Non	1453	12531	Pipelined Machine Verification

La taille de population (3) et le nombre maximum des générations (5.000, le seul critère d'arrêt) ont été fixés en se basant sur les observations des expériences antérieures.

4.6.2 Paramétrage de ExCoDyMAB

Nous devons définir comment intégrer efficacement Compass et Ex-DMAB. La première décision concerne l'inclusion ou non des fenêtres glissantes W_1 et/ou W_2, et ce qu'il faut prendre en tant que sortie. Les mesures de sortie suivantes sont disponibles (pour chaque fenêtre) :

- Valeur *moyenne* (M) des mesures stockées.
- Valeur *extrême* (E) (maximum) des mesures stockées (sauf pour le temps d'exécution, où nous pronnons toujours la moyenne).

– Valeur *instantanée* (I), i.e., sans fenêtre glissante.

En outre, les méta-paramètres suivants doivent également être analysés et réglés :
– La taille de W_1, τ, et la taille de W_2, κ.
– Le paramètre C de DMAB, qui définit l'équilibre entre l'exploration et l'exploitation au niveau du choix de l'opérateur.
– Le paramètre γ de DMAB, correspondant au seuil du test de détection de changement qui déclenche les réinitialisations.

Les domaines pour les différents paramètres sont : $C \in \{5, 7, 10\}$; $\gamma \in \{1, 3, 5\}$; et le type de fenêtres (taille) $\in \{A(10), A(50), E(10), E(50), I(1)\}$ pour W_1 et W_2. Ainsi, le nombre de configurations possibles est de 225.

En accord avec la discussion de la section 4.4, la valeur de l'angle Θ reste fixée à la valeur 0.25π.

Racing

Étant donné le nombre levé de configurations possibles, une méthode de réglage de paramètres off-line a été employée pour faciliter cette analyse empirique : la F-Race [Birattari, 2004; Yuan and Gallagher, 2004]. Il s'agit d'une variante des méthodes de Racing qui emploie le test statistique de variance bi-directionnelle de Friedman pour éliminer les candidats les moins performants.

Le Racing est une alternative à une expérimentation complète de toutes les combinaisons possibles, dans laquelle M exécutions devraient être effectuées sur N instances pour chaque configuration. En utilisant le Racing, les configurations les moins performantes par rapport à la meilleure configuration sont éliminées au fur et à mesure qu'il existe assez d'évidence statistique pour les comparer. Des cycles "exécution - comparaison - élimination" sont répétés jusqu'à ce qu'il ne reste qu'une configuration ou que d'autres critères d'arrêt soient satisfaits. Ainsi, au lieu de perdre du temps de calcul dans les configurations prouvées inférieures, cette approche consacre les ressources aux configurations les plus prometteuses. Le racing exige typiquement 10-20 % du temps nécessaire pour réaliser une analyse combinatoire complète.

Dans notre cas, le Racing est arrêté lorsque 80 cycles ont été effectués ou qu'il ne reste qu'une configuration survivante. Des éliminations sont faites à partir du 11ème cycle. Comme recommandé dans [Birattari, 2004], afin de permettre une comparaison juste, toutes les expériences du même cycle utilisent la même population initiale.

4.7 Résultats et discussion

4.7.1 Determination des méta-paramètres d'ExCoDyMAB

À la fin du processus de Racing, 4 configurations étaient toujours "vivantes". Elles sont présentées dans la table 4.4. Ceci indique clairement que la fenêtre glissante la plus importante est W_1, et qu'elle devrait être employée dans sa configuration extrême avec une taille de 10 (i.e., en prenant la valeur maximale des 10 dernières mesures de chaque opérateur), qu'importe quelle taille ou mesure que l'on prenne par W_2.

Tab. 4.4 – Survivants du Racing

Nom	W_1 : type, taille	W_2 : type, taille	C	γ
A	Extrême, 10	Instantané	7	1
B	Extrême, 10	Moyenne, 10	7	1
C	Extrême, 10	Moyenne, 50	7	1
D	Extrême, 10	Extrême, 10	7	3

Les résultats renforcent l'idée d'utiliser les améliorations peu courantes (extrêmes) plutôt que les valeurs moyennes. En outre, la taille de 10 pour W_1 peut être interprétée par le raisonnement suivant : avec la politique extrême, une valeur plus grande de τ produirait une plus longue persistance des valeurs extrêmes, même si les comportements des opérateurs ont déjà changé. D'autre part, une valeur plus petite de $\tau = 1$ (équivalent à utiliser la valeur instantanée) oublierait ces cas rares mais bons. On pourrait supposer qu'une taille optimale pour W_1 dépend du paysage de recherche et des opérateurs utilisés. Plus d'investigations sont nécessaires pour mieux comprendre le réglage de τ.

4.7.2 Validation de la généralité

Afin de vérifier la généralité de ces paramètres, 50 exécutions ont été faites sur les 22 instances SAT avec chacune des 4 configurations survivantes, dans le but de les comparer et vérifier leurs performances par rapport aux méthodes qui servent de base de comparaison : les configurations originales de *Compass* et *Ex-DMAB*, et le *Choix Uniforme* des opérateurs. Les résultats de cette comparaison sont présentés dans la table 4.5. Chaque valeur représente le nombre de problèmes dans lesquels une architecture est sensiblement meilleure qu'autre (en utilisant un test T Student avec 95% de confiance). Par exemple, dans le coin gauche inférieur, "18-2" veut dire que D a surpassé Compass sur 18 instances, alors que l'opposé s'est produit seulement 2 fois. Finalement, la colonne à droite montre le nombre relatif de fois où une architecture gagne ou perd, comme une mesure globale de qualité comparative.

La dominance d'ExCoDyMAB est nette, et confirme l'hypothèse qui a motivé la combinaison des approches Compass et Ex-DMAB. Ces deux approches seules, dans leurs configurations originales respectives, présentent une performance plus ou moins équivalente, clairement inférieure à ExCoDyMAB, bien que supérieures au Choix Uniforme.

Un autre aspect important est la bonne capacité de généralisation d'ExCoDyMAB. Les meilleures configurations trouvées par le Racing ont une performance similaire lorsqu'elles résolvent de nouvelles instances. Ceci nous incite à penser que nous avons trouvé une configuration assez générale des méta-paramètres, au moins pour l'AE utilisé dans ce travail.

Il est également intéressant de noter que, car les récompenses obtenues par Compass sont normalisées par rapport à la plus haute valeur, les valeur livrées à DMAB sont toujours dans la même gamme de valeur. Puisque le but du méta-paramètre C est d'ajus-

Tab. 4.5 – Résultats comparatifs sur les 22 instances : chaque boîte montre le nombre de problèmes dans lesquels une architecture est sensiblement meilleure que l'autre (en utilisant un test T de Student avec un 95% de confiance)

	Compass	*Ex-DMAB*	*CU*	A	B	C	D	$\sum dom$
Compass		9-9	22-0	4-18	2-17	2-18	2-18	-39
Ex-DMAB	9-9		22-0	0-18	0-21	0-21	0-21	-59
CU	0-22	0-22		0-22	0-22	0-22	0-22	-132
A	18-4	18-0	22-0		0-1	0-5	0-2	46
B	17-2	21-0	22-0	1-0		0-2	3-1	59
C	18-2	21-0	22-0	5-0	2-0		4-0	70
D	18-2	21-0	22-0	2-0	1-3	0-4		55

ter l'importance entre les opérateurs plus performants et ceux qui le sont moins, une récompense normalisée pourrait signifier qu'une valeur de C proche de 7 est une valeur presque définitive pour les différentes instances d'AEs utilisant la combinaison de *Sélection Adaptative d'Opérateurs* proposée.

Le méta-paramètre γ est plus difficile à saisir. La fréquence à laquelle DMAB est réinitialisé semble être liée aux opérateurs employés par l'AE. Des opérateurs dont l'exécution dépend de la zone courante de l'espace de recherche exploré devraient être supervisés plus fréquemment que d'autres qui changent moins. Considérons par exemple l'opérateur de recherche locale basé sur la descente stricte : son exécution, en termes d'amélioration de qualité, variera selon que l'individu considéré soit sur une pente ou déjà dans un optimum local. D'autre part, un opérateur de mutation présentera une exécution semblable, en termes d'augmentation de diversité, sans importer la position de l'individu. D'ailleurs, le paysage de recherche pourrait influencer la façon dont γ affecte le contrôle de la recherche.

Les résultats sur les 22 instances de SAT, en utilisant la meilleure configuration de méta-paramètres trouvée pour ExCoDyMAB (C), ainsi que ceux de *Compass*, *Ex-DMAB* et le *Choix Uniforme* sont présentés dans la table 4.6. Les colonnes montrent le nombre moyen de clauses fausses après 5000 évaluations sur 50 exécutions, ainsi que l'écart type entre parenthèses. Les meilleurs résultats (et ceux qui sont indistingables, en utilisant un test T de Student avec 95% de confiance) apparaissent en gras.

Comme dit avant, les résultats de la table 4.6 montrent un clair avantage pour *ExCo-DyMAB*. L'ensemble d'instances utilisé est suffisamment difficile pour évidencier l'avantage d'employer la combinaison de *Compass* et d'*Ex-DMAB* plutôt que l'un ou l'autre séparément, ou bien un choix uniforme. Le choix délibéré d'utiliser plusieurs opérateurs non spécialisés est également un aspect important pour valider la capacité de contrôle d'ExCoDyMAB sur des opérateurs qui présentent d'efficacités très différentes.

Tab. 4.6 – Résultats sur 22 instances : moyenne (écart type) du nombre de clauses fausses, sur 50 exécutions

Méthode / Problème	*ExCoDyMAB (C)*		*Compass*		*Ex-DMAB*		*Choix Uniforme*	
4blocks	**2.8**	(0.9)	6.0	(0.9)	6.2	(0.9)	13.4	(0.6)
aim	**1.0**	(0.0)	**1.0**	(0.0)	1.2	(0.3)	3.6	(1.8)
f1000	**10.3**	(2.3)	30.9	(6.2)	16.4	(2.6)	55.8	(8.6)
CBS	**0.6**	(0.6)	**0.4**	(0.5)	1.0	(0.9)	7.0	(2.7)
Flat200	**7.2**	(1.7)	10.6	(2.1)	10.7	(2.2)	37.7	(5.5)
logistics	**6.5**	(1.3)	7.6	(0.5)	8.8	(1.5)	17.9	(4.1)
medium	1.5	(1.5)	**0.0**	(0.0)	1.8	(1.6)	8.8	(3.4)
Par16	**15.2**	(3.1)	64.0	(10.2)	24.1	(5.7)	131.1	(14.5)
sw100-p0	**9.2**	(1.2)	12.8	(1.4)	12.5	(1.7)	25.9	(3.4)
sw100-p1	**0.0**	(0.0)	0.5	(0.6)	1.1	(0.8)	11.3	(3.5)
Uf250	**0.9**	(0.7)	1.8	(0.9)	1.7	(0.8)	9.1	(3.3)
Uuf250	**2.5**	(1.0)	4.5	(1.2)	3.1	(1.1)	12.7	(3.2)
Color	**48.0**	(2.5)	61.3	(2.2)	49.3	(3.4)	80.4	(6.6)
G125	**8.8**	(1.3)	20.6	(2.0)	13.5	(1.7)	28.8	(4.6)
Goldb-heqc	**72.9**	(8.5)	112.2	(15.2)	133.2	(15.9)	609.7	(96.2)
Grieu-vmpc	16.7	(1.7)	**15.2**	(1.7)	19.6	(1.8)	24.1	(3.3)
Hoons-vbmc	**69.7**	(14.5)	268.1	(44.6)	248.3	(24.1)	784.5	(91.9)
Manol-pipe	**163.0**	(18.9)	389.6	(37.2)	321.0	(38.1)	1482.4	(181.5)
Schup	**306.6**	(26.9)	807.9	(81.8)	623.7	(48.5)	1639.5	(169.9)
Simon	**29.6**	(3.3)	43.5	(2.7)	35.3	(6.3)	72.6	(11.3)
Velev-eng	**18.3**	(5.2)	29.5	(7.3)	118.0	(37.1)	394.0	(75.8)
Velev-sss	**2.0**	(0.6)	4.6	(1.0)	5.9	(3.9)	62.7	(25.2)

4.8 Conclusion du chapitre

Dans ce chapitre nous avons présenté Compass, un contrôleur d'AEs qui fournit une abstraction des paramètres et simplifie le contrôle en ajustant les niveaux de l'exploration et de l'exploitation au cours de la recherche. Ce contrôleur mesure les effets des opérateurs sur la qualité de la population, sa diversité et le temps d'exécution des opérateurs. Compass est indépendant des AEs, afin de fournir une couche de contrôle additionnelle qui pourrait être employée par d'autres algorithmes basés sur des populations.

L'évaluation double de l'effet des opérateurs sur la recherche (i.e., variation de qualité et de diversité) est cohérent avec les principes directifs des algorithmes de recherche basés sur la population, c'est-à-dire, de maximiser la qualité des solutions tout en évitant la concentration de la population, afin de tirer bénéfice de leur nature parallèle. En considérant ces deux mesures, nous disposons d'un mécanisme naturel pour échapper aux optimums locaux.

Nous avons conservé les principes énoncés dans le chapitre précédent, c'est-à-dire l'abstraction du contrôle, l'apprentissage du comportement de l'AE, la prise en compte des deux objectifs principaux (maximiser la diversité et la qualité) et l'architecture simple. Les principales différences par rapport à l'approche antérieure sont que maintenant nous nous focalisons sur les paramètres liés aux opérateurs, et que nous surmontons le problème de la dimensionalité, ce qui permet à Compass de contrôler plusieurs opérateurs à la fois. Additionnellement, l'apprentissage est fait continuellement, et pas dans une phase séparée comme avant.

Le profil des opérateurs est facilement appréhendable en observant les points associés sur le graphique (ΔDiversité, ΔQualité), ainsi Compass pourrait également être utilisé comme un outil pour comprendre le rôle des opérateurs.

Des expériences ont été réalisées en utilisant un AE avec 6 opérateurs pour résoudre des instances du problème SAT. Des résultats ont été favorablement comparés contre des contrôleurs de l'état de l'art et le choix uniforme basique.

Une variante de Compass, appelée ExCoDyMAB a été également présentée. Cette variante est obtenue en intégrant l'affectation de crédits de Compass avec la sélection d'opérateurs de la méthode Ex-DMAB. Deux caractéristiques nouvelles d'ExCoDyMAB peuvent être soulignées ici : l'affectation de crédits extrême de Compass, qui obtient la valeur maximale des valeurs stockées de variation de diversité et de qualité, et la méthode de sélection d'opérateurs de DMAB, qui s'adapte rapidement aux changements d'efficacité des opérateurs grâce au redémarrage des variables du DMAB. Les deux dispositifs, issus de différents contextes, se sont avérés très complémentaires

Des résultats intéressants ont été obtenus en utilisant cette nouvelle approche, en dépassant les configurations originelles de Compass et Ex-DMAB.

Un inconvénient d'ExCoDyMAB demeure le réglage de ses méta-paramètres. Même si l'utilisation de F-Race pour l'obtention des méta-paramètres nous a permis d'obtenir des réglages assez généraux dans 15% du temps requis par une analyse factorielle complète, le réglage off-line demeure coûteux. Bien que la normalisation des améliorations de diversité et de qualité de Compass fournisse un réglage robuste pour le paramètre de graduation de l'équilibre de MAB (C), il faudrait une étude plus approfondie pour comprendre le réglage du méta-paramètre γ, qui détermine le déclenchement et le redémarrage des variables du MAB.

Une intéressante possibilité d'extension consiste à généraliser l'approche présenté dans ce chapitre vers des heuristiques d'un point. Tandis que la mesure de qualité peut être prise de la même façon, ceci n'est pas le cas pour la diversité. En effet, quand il n'existe pas de population, il est impossible de mesurer sa diversité. Une façon de faire ceci est de remplacer la diversité de la population par une diversité "temporelle" en mesurant la différence des variables du point pendant une fenêtre de temps [Robet *et al.*, 2009].

La gestion des opérateurs inconnus non standard ouvre également la perspective d'utiliser Compass pour évaluer des opérateurs produits automatiquement, ce qui sera l'objet d'étude du prochain chapitre.

Chapitre 5

Gestion autonome d'opérateurs

Sommaire

5.1 Introduction

Comme nous l'avons expliqué dans la section 2.1.2, nous distinguons deux catégories principales de paramètres : les paramètres *comportementaux*, qui correspondent à ceux que "ne changent pas" l'algorithme lui-même (par exemple les taux d'application des opérateurs ou la taille de la population), et les paramètres *structuraux*, qui modifient l'algorithme d'une manière plus radicale, le transformant éventuellement en un autre algorithme (par exemple, le choix du codage ou des opérateurs qui seront appliqués).

Même s'il est difficile de distinguer clairement les paramètres capables de transformer l'algorithme de ceux qui font des petites variations, nous pouvons identifier différentes vocations dans chaque catégorie. D'une part, les paramètres *structuraux* sont la plupart du temps liés à la conception de l'algorithme, permettant la spécialisation des AE standard, particulièrement en incluant des opérateurs *ad-hoc*. La conception des opérateurs ad-hoc ou le choix du codage exigent une expertise considérable, car l'impact sur l'algorithme est plus difficile à prévoir pour des opérateurs *ad-hoc* que pour des opérateurs standard. Il est alors plus difficile le réglage des paramètres associés à ces opérateurs. D'autre part, nous avons les paramètres *comportementaux*, liés à la manipulation de l'algorithme, comme nous l'avons vu jusqu'ici.

Dans ce chapitre, nous proposons une nouvelle approche, basée sur la méthode présentée dans le chapitre précédent, pour créer un contrôleur qui permetet une plus grande autonomie, en manipulant non seulement les paramètres qui déterminent le comportement, mais également ceux qui déterminent les opérateurs qui seront utilisés. Cette méthode, qui constitue la partie la plus récente de notre recherche, a été présentée dans [Maturana *et al.*, 2009b].

Dans cette nouvelle approche, nous considérons un ensemble d'opérateurs (qui peuvent être dynamiquement produits par une usine d'opérateurs), qui sont inclus dans l'algorithme à tout moment. Le contrôleur doit donc réaliser deux tâches principales. La première consiste à identifier les bons opérateurs, afin de composer un bon ensemble d'opérateurs, et la deuxième, déjà abordée dans le chapitre précédent, consiste à bien choisir les opérateurs à appliquer à chaque instant.

Dans cette étude, notre objectif est double :

- Montrer que l'AE peut choisir et employer de façon autonome les opérateurs adaptés à l'état actuel de la recherche, grâce au mécanisme de commande que nous définirons et,
- Montrer que, sans connaître les opérateurs les plus performants, nous pouvons reobtenir, grâce au mécanisme de commande du contrôleur, des résultats comparables à ceux que nous avons obtenus avec des opérateurs spécifiques, conçus à la suite d'analyses approfondies.

Le chapitre est organisé comme suit. La section 5.2 présente la méthode, qui constitue une extension de celle présentée dans le chapitre précédent. La section 5.3 décrit le cadre expérimental utilisé pour valider notre approche, dont les résultats sont présentés et discutés dans la section 5.4. Finalement, les conclusions du chapitre sont presentés dans la section 5.5.

5.2 Description de la méthode

Dans cette section, nous allons décrire précisément le mécanisme de contrôle de l'AE que nous proposons. L'architecture globale de notre approche est détaillée par la figure 5.1, sur laquelle on constate que le contrôleur comprend principalement deux composants.

Le premier composant est la *Sélection Adaptative d'Opérateurs (SAO)*, détaillée dans le chapitre précédent. Le deuxième composant, appelé *Forgeron*, est en rapport avec les paramètres structuraux de l'AE. Le forgeron est en effet l'"administrateur d'opérateurs" qui décide quels opérateurs seront disponibles pour la SAO (et par conséquent inclus dans l'AE) à chaque instant de la recherche. Les opérateurs sont construits selon des spécifications qui peuvent résulter d'une combinaison de différents composants basiques (comme c'est le cas dans ce travail) ou tout simplement à partir d'une liste de noms d'opérateur.

La figure 5.1 montre l'intégration du Forgeron avec la SAO à l'intérieur du Contrôleur, et l'interaction de ce dernier avec l'AE. Notons que l'interaction entre l'AE et la SAO reste inchangée par rapport aux schémas présentés précédemment (voir figures 4.1 et la section 3.7). L'inclusion du Forgeron est donc une spécialisation du contrôleur qui ne complexifie pas l'intégration du contrôleur à l'AE.

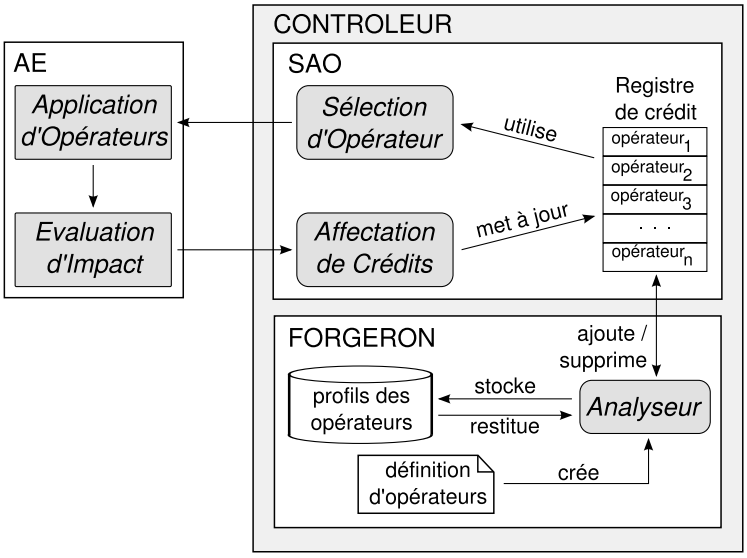

Fig. 5.1 – Schéma général du contrôleur, montrant ses deux composants principaux : la SAO et le Forgeron

Nous voulons remarquer la différence conceptuelle entre les deux composants du contrôleur. Le Forgeron "crée" bel et bien l'AE que l'AOS contrôlera. Dans cette approche, les choix de la SAO dépendent donc de ceux faits par le Forgeron.

5.2.1 Sélection Adaptative d'Opérateurs

Dans cette section nous allons décrire les mécanismes de sélection adaptative d'opérateurs (voir section 4.2.1) utilisées ici.

Affectation de Crédits

Afin d'évaluer la qualité des opérateurs, il faut mesurer leurs performances suite à leur application. La méthode présentée dans la section 4.2.2 prend en compte trois mesures différentes : la variation de diversité, la variation de qualité et le temps d'exécution. L'objectif est alors de maximiser les deux premiers critères et de minimiser le troisième. Comme ces objectifs sont la plupart de temps contradictoires, nous avons besoin de trouver une façon de les combiner pour obtenir une évaluation unique.

La méthode *Compass (C)* (Figure 5.2.a), prend en considération la distance d'un point $(\Delta Diversité, \Delta Qualité)$, représentant l'évaluation d'un opérateur o, à une ligne inclinée d'un angle $\Theta = \pi/4$. Plus un point est à droite et en hauteur, mieux il sera considéré. De plus, les temps d'exécution de chaque opérateur sont considérés, afin de récompenser les opérateurs les plus rapides.

Dans ce chapitre, deux autres méthodes de mesure sont considerés, tous les deux basés sur la notion de dominance Pareto (voir section 1.3.4). L'affectation de crédits dénommée *Dominance Pareto* (DP, figure 5.2.b) considère le nombre d'opérateurs que chaque opérateur domine, étant plus appréciés les opérateurs avec des valeurs grandes. Par contre, pour la mesure *Rang Pareto* (RP, figure 5.2.c), on regarde combien d'opérateurs dominent chaque opérateur (les valeurs les plus petites sont alors préférables). Une différence importante existe entre ces deux évaluations. Alors que le RP considère uniquement des opérateurs qui ne sont pas dominés, la DP récompense ceux qui sont en forte concurrence avec d'autres opérateurs. Un effet de type *essaim* se produit ici : il ne suffit pas d'être bon mais encore faut-il l'être là où la plupart des opérateurs sont placés.

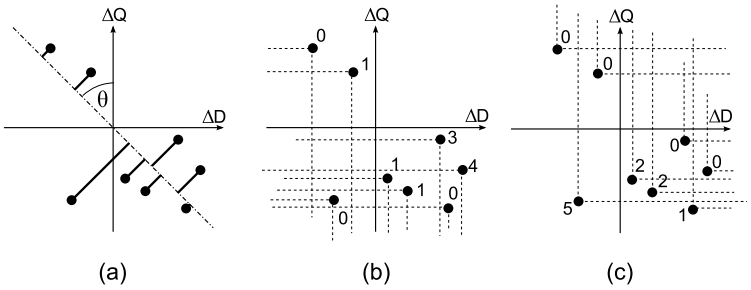

Fig. 5.2 – Schémas d'affectation de crédits. Compass (a), Dominance Pareto (b), Rang Pareto (c)

Après qu'un opérateur a été appliqué, les mesures de $\Delta Diversité$ et de $\Delta Qualité$ sont envoyées au contrôleur. Le module d'*Affectation de Crédits* calcule alors l'évaluation (C, DP ou RP, selon le schéma choisi) et normalise les valeurs par rapport au reste des opérateurs. Les valeurs normalisées sont stockées dans le Registre de Crédit en tant que

récompenses. Une liste des m dernières récompenses de chaque opérateur (correspondant à ses dernières m applications) est enregistrée dans le Registre de Crédit, afin de fournir des informations historiques mises à jour au module de *Sélection d'Opérateurs* , concernant chaque opérateur.

Sélection d'Opérateurs

Le module de *Sélection d'Opérateurs* choisit l'opérateur à appliquer à chaque instant, en se basant sur l'information recueillie lors des exécutions passées, sans négliger l'exploration au niveau de la *Sélection Adaptative d'Opérateurs* . L'idée de la *Sélection d'Opérateurs* qui a été présentée dans la section 4.5.2, appelée Ex-DMAB, est inspirée par les méthodes de bandits manchots à bras multiples (multi armed bandits) utilisés dans le cadre de la théorie des jeux. Cette stratégie choisit toujours l'opérateur le plus efficace, donné par la formule 4.2.

Notons cependant que l'expression 4.2 se fonde sur l'hypothèse que tous les opérateurs sont présents dans l'AE depuis le début de l'exécution. En effet, si un opérateur est inclus pendant l'exécution, sa valeur de $n_{o,t}$ serait si basse que l'AOS serait forcé de l'appliquer un grand nombre de fois afin de niveler la valeur de l'expression 4.2 avec le reste des opérateurs.

Puisque nous sommes intéressés par des opérateurs qui entrent et sortent de l'AE au cours de la recherche, nous avons reformulé l'expression 4.2 afin de traiter un ensemble dynamique d'opérateurs. Cette nouvelle expression est obtenue principalement en remplaçant la mesure correspondant au nombre de fois qu'un opérateur a été appliqué par un autre critère qui correspond au nombre de générations écoulées depuis la dernière application de l'opérateur (i.e., son temps "à vide"). Ceci permet à un nouvel opérateur de "se niveler" par rapport aux autres immédiatement en l'appliquant une fois. La nouvelle évaluation de performance est alors définie comme suit :

$$MAB2_{o,t} = r_{o,t} + 2 \times exp(p \times i_{o,t} - p \times x \times NO_t) \tag{5.1}$$

où $i_{o,t}$ est le temps "à vide" de l'opérateur o au temps t, NO_t est le nombre d'opérateurs considérés par l'AOS au temps t, et x est une constante multiplicative qui indique que le contrôleur doit attendre $x \times NO_t$ générations avant d'appliquer forcement l'opérateur o. Le comportement du composant d'exploration est mieux compris en regardant la figure 5.3. La valeur de cette partie reste près de zéro sauf lorsque la valeur de $i_{o,t}$ est proche de $x \times NO_t$. Puisque les valeurs de $r_{o,t}$ sont normalisés dans l'intervalle [0.1], lorsqu'un opérateur n'a pas été appliqué pendant longtemps, son application devient obligatoire. p est un paramètre qui ajuste la pente de l'exponentielle.

Dans le travail que nous présentons ici, nous comparons quatre SO différentes :

Aléatoire (A), choisit au hasard l'opérateur à appliquer.

Probabilité Proportionnelle (PP), choisit les opérateurs avec une probabilité proportionnelle aux récompenses enregistrées dans le registre de crédit.

MAB2 (M2), (déjà décrit) choisit toujours l'opérateur qui maximise l'expression 5.1

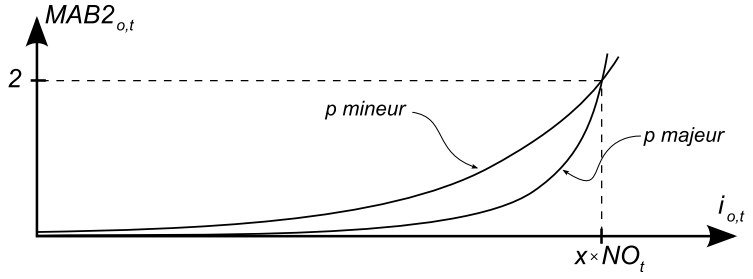

FIG. 5.3 – Comportement du composant exploratoire de l'expression 5.1

MAB2+Détection de Collusion (M2C), similaire à MAB2, il comporte additionnellement un mécanisme de détection de collusion des individus (nous choisissons ce terme, sans doute un peu fort, traduisant le regroupement autour des optimums locaux, qui nuit à la poursuite de la recherche). Ceci est réalisé en considérant la pente de la ligne résultant de la régression linéaire des valeurs de qualité moyenne de la population. Si la pente est proche de zéro, on considère que l'AE est "coincé"' et on démarre une phase d'exploration forcée, en ne prenant que les opérateurs qui se sont avérés exploratoires. Cette phase est abandonnée lorsque la diversité atteint une certaine fraction au-dessus de la diversité actuelle, quand il n'y a pas d'opérateurs de diversification, ou quand un nombre de générations se sont écoulées sans pouvoir augmenter la diversité.

5.2.2 Le forgeron

Comme mentionné ci-dessus, le contrôleur a affaire avec les opérateurs qui éventuellement entrent et sortent du Registre de Crédit. Le forgeron est le composant responsable de la gestion de l'inclusion ou l'exclusion de ces opérateurs, de façon que l'AE ait toujours des opérateurs adéquats. Puisque les opérateurs écartés pourraient devenir utiles à l'avenir, le Forgeron garde une trace d'eux. Nous distinguons trois états principaux pour les opérateurs (figure 5.4).

- **Non-né**, correspond aux opérateurs qui n'ont jamais été employés pendant l'exécution de l'algorithme.
- **Vivant**, correspond aux opérateurs qui sont actuellement dans le Registre de Crédit et donc utilisés par l'AE. Les opérateurs dans cet état possèdent deux structures de données associées : le *data*, correspondant aux mesures récentes de performance de l'exécution et le *profil*, qui récapitule l'information de *data*, en calculant des statistiques significatives.
- **Mort**, correspond aux opérateurs qui ont été éliminés du Registre de Crédit. Les opérateurs morts perdent leur structure *data*, mais gardent leur *profil*.

Notons que, à l'exception de leurs mesures d'exécution, la seule information connue par le contrôleur au sujet des opérateurs est leur nom. Le contrôleur est indépendant de l'AE et donc ne contient aucun détail de son implémentation. C'est le rôle de l'AE d'implémenter et d'appliquer les opérateurs. Cette architecture assure l'indépendance du contrôleur et sa

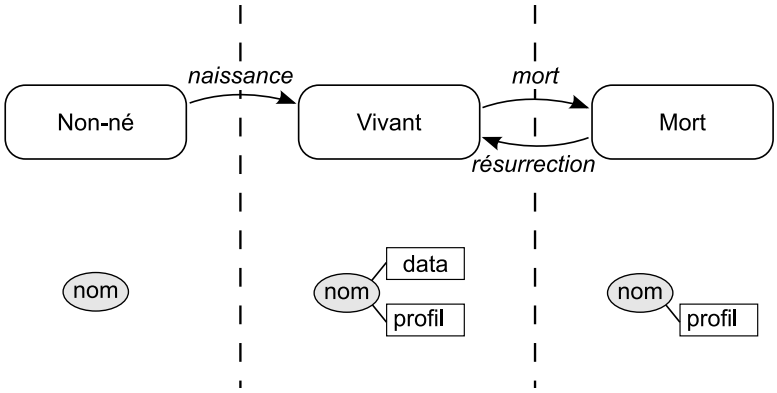

FIG. 5.4 – États des opérateurs et données associées

portabilité dans différents AEs, sans un effort de modification significatif.

Le Forgeron commande les paramètres structuraux, en décidant quand et si les opérateurs seront disponibles dans l'AE. Ses actions spécifiques sont détaillées ci-dessous (voir la figure 5.1) :

- **Crée** les opérateurs, en accord avec leur définition. Cette tâche peut être réalisée en composant les opérateurs à partir d'une définition combinatoire ou en prenant tout simplement leur définition (nom) depuis une liste d'opérateurs possibles.
- **Ajoute** des opérateurs au Registre de Crédit afin de les rendre disponibles à l'AE. Ceci produit la transition "*naissance*" représentée dans la figure 5.4.
- **Analyse** les opérateurs présents actuellement dans le Registre de Crédit, afin de décider si certains d'entre eux doivent être "tués" ou si un opérateur (non-né ou mort) doit être (ré-)incorporé au Registre.
- **Élimine** des opérateurs depuis le registre. Ceci produit la transition "*mort*" présentée dans la figure 5.4.
- **Stocke** le profil des opérateurs éliminés.
- **Restitue** des opérateurs morts dans le Registre lorsqu'il est nécessaire. Ceci produit la transition "*résurrection*" présentée dans la figure 5.4.

L'élimination d'un opérateur ne signifie nécessairement pas qu'il soit mauvais *per se*, mais plutôt dans le contexte des exigences actuelles de la recherche : un opérateur peut être un excellent diversificateur, mais être éliminé si l'état actuel de la recherche a besoin d'opérateurs d'exploitation pour converger davantage. Ceci explique pourquoi les profils des opérateurs sont stockés lorsqu'ils sont éliminés du registre : en cas de besoin, la résurrection d'un opérateur ne se fera pas aveuglément.

Dans notre implémentation, le registre de crédit est composé d'un nombre fixe d'opérateurs, et évalué à intervalles réguliers par le forgeron dans la recherche des opérateurs faibles qui peuvent être éliminés pour laisser place à un nouveau-né. Tous les opérateurs possibles sont créés puis essayés avant de rechercher à nouveau parmi les morts, ce qui permet de donner à tous une chance de prouver leurs capacités. Nous fournissons plus de détails dans la section 5.3.2.

5.2.3 Algorithmes évolutionnaires pour SAT

Les AEs pour SAT [De Jong and Spears, 1989; Fleurent and Ferland, 1996; Gottlieb and Voss, 2000; Rossi *et al.*, 2000; Lardeux *et al.*, 2006] font partie des méthodes de résolution approchées de ce problème. Comme cela a été rappelé dans l'introduction, le principe de base de ces algorithmes est de manipuler une population d'individus à l'aide d'opérateurs génétiques afin d'obtenir des individus de bonne qualité. Dans le cas du problème SAT (voir section 1.4.1), les individus sont des affectations des variables Booléennes et l'objectif est d'avoir des individus induisant le moins de clauses fausses possibles.

L'algorithme génétique pour SAT que nous allons utiliser pour nos expériences repose sur GASAT [Lardeux *et al.*, 2006] qui est actuellement l'un des AEs les plus efficaces pour SAT. Son principe basique est le suivant :

1. une population est générée aléatoirement ;
2. deux individus sont sélectionnés aléatoirement à partir d'une sous-population des meilleurs individus ;
3. l'opérateur de croisement CC (voir section 5.2.4) est appliqué sur ces individus ;
4. l'opérateur de mutation applique une recherche locale sur le fils obtenu (partie mémétique) ;
5. le nouvel individu est inséré dans la population et remplace le plus vieil individu s'il est meilleur que le moins bon individu de la sous-population ;
6. si aucune des conditions d'arrêt (affectation satisfaisant le problème ou nombre de croisements maximum atteint, entre autres) n'est atteinte, retour au pas 2.

L'objectif de notre travail étant de mettre en avant les propriétés du contrôleur, nous avons modifié GASAT pour n'en garder que le squelette et ainsi observer la seule action du contrôleur. Pour cela, nous avons remplacé l'opérateur de sélection par une sélection aléatoire et nous avons supprimé la mutation. L'opérateur d'insertion est modifié pour que le nouvel fils substitue automatiquement au plus vieil individu.

5.2.4 Une famille d'opérateurs de croisement pour SAT

L'opérateur de croisement permet à la population d'être régénérée avec de bons individus. Dans notre problématique de contrôle autonome, la définition d'un bon individu dépend de l'état de la recherche. Dans certains cas, il peut être plus intéressant d'obtenir un individu qui diversifie la population et dans d'autres, les individus améliorant la qualité seront les plus attendus. Pour cette raison, nous avons décidé de travailler avec plusieurs croisements ayant chacun une prédisposition, soit à améliorer la qualité, soit à favoriser la diversité. La plupart des croisements existants essaie de conserver les bonnes propriétés des parents afin de les reproduire chez le fils. Par exemple :

- Le croisement uniforme conserve les valeurs de vérité des variables qui sont identiques chez les deux parents.
- Le croisement proposé par [Fleurent and Ferland, 1996] utilise l'ensemble des clauses qui sont vraies chez un parent et fausses chez l'autre. Seules sont conservées les valeurs des variables apparaissant dans les clauses vraies de cet ensemble.

- Le croisement CC [Lardeux *et al.*, 2006] ne traite que les clauses fausses simultanément chez les deux parents et les rend vraies chez le fils en flippant une variable dans chacune d'elles.
- Le croisement CCTM [Lardeux *et al.*, 2006] opère de la même manière que CC mais il travaille ensuite sur les clauses vraies chez les deux parents afin de garantir que les clauses seront aussi vraies chez le fils.

Très peu d'opérateurs de croisements ont comme objectif principal la diversification. Nous en avons donc redéfini plusieurs qui essaient de détecter les bonnes propriétés des parents pour ensuite les casser. Par exemple :

- le croisement uniforme "inverse" qui flippe toutes les valeurs de vérité des variables qui sont identiques chez les deux parents ;
- le croisement NT ne traite que les clauses vraies chez les deux parents et les rend fausses chez le fils.

Nous utilisons un grand nombre d'autres croisements (307 au total) qui sont des combinaisons de croisements existants. Dans cet ensemble on trouve des croisements dont l'objectif principal est de diversifier, d'autres dont la tâche première est d'améliorer la qualité et enfin un grand nombre dont l'action est moins tranchée. La définition de chaque opérateur est donnée par la combinaison des critères suivants :

Sélection des clauses qui sont fausses dans les deux parents :

1. ne rien faire
2. les sélectionner dans l'ordre chronologique
3. choisir une clause aléatoirement
4. choisir une clause aléatoirement depuis l'ensemble des plus petites
5. choisir une clause aléatoirement depuis l'ensemble des plus grandes

Action sur chacune des clauses fausses :

1. ne rien faire
2. flipper la variable maximisant le nombre de clauses fausses qui deviennent vraies
3. idem que la précédente, mais également minimisant les clauses vraies qui deviennent fausses
4. flipper toutes les variables
5. flipper le literal qui apparaît moins souvent dans les autres clauses

Sélection des clauses qui sont vraies dans les deux parents :

1. ne rien faire
2. les sélectionner dans l'ordre chronologique
3. choisir une clause aléatoirement
4. choisir une clause aléatoirement depuis l'ensemble des plus petites
5. choisir une clause aléatoirement depuis l'ensemble des plus grandes

Action sur chacune des clauses vraies :

1. ne rien faire

2. mettre à vrai la variable dont le flip minimise le nombre de clauses fausses
3. mettre à vrai tous les literaux
4. mettre à vrai tous les literaux dont leur négation apparaît moins souvent dans les autres clauses
5. mettre à faux tous les literaux

5.3 Cadre expérimental

Nous présentons ici une analyse expérimentale de la méthode décrite ci-dessus, dans le cadre de la résolution du problème SAT. Nous comparons les différentes configurations du contrôleur, présentées dans la section 5.2.4, avec les croisements de l'état de l'art (FF, CC et CCTM). Comme base, nous avons également considéré un croisement uniforme, et un contrôleur qui applique aléatoirement une des 307 combinaisons possibles de croisement (appelé *A307*).

5.3.1 Algorithme évolutionnaire

Pour nos expériences nous avons sélectionné des instances venant de différentes compétitions SAT. Le choix des benchmarks essaie de couvrir les différentes familles d'instances (aléatoires, faites-main et industrielles). La liste des instances est présentée ci-dessous :

- *f500* et *uf250-010* sont des instances générées aléatoirement au seuil [Zecchina *et al.*, 1999].
- *aim-100-1_6-yes1-1* (notée aim-100 par la suite) est une instance aléatoire modifiée afin de n'avoir qu'une seule solution.
- *ibm-2004-29-k55* (notée ibm par la suite) est une instance industrielle (BMC, [Biere *et al.*, 2003]).
- *simon-s02b-r4b1k1.2* (notée simon par la suite) est une instance difficile des compétitions SAT.
- *bw_large.d* est une instance de planning (blocks world) ;
- *flat200-19* est une instance de coloration de graphe
- *engine_4_nd.cnf* (notée engine par la suite) est une instance d'intégration à très grande échelle (VLSI) ;

L'algorithme de base servant à nos expériences (voir section 5.2.3) est appliqué 50 fois pour chaque croisement et chaque contrôleur, et cela pour chacune des 8 instances testées. Lors d'une exécution, la population possède 100 individus et le nombre de croisements autorisé est de 100.000.

5.3.2 Méta-paramétrage du contrôleur

Notre objectif est d'essayer les différentes combinaisons d'AC et de SO présentées dans les sections 5.2.1 et 5.2.1. Ces combinaisons seront identifiées par la notation *X*-*Y*, où

$X \in \{C, PD, PR\}$ est le mécanisme d'AC et $Y \in \{M2, R, M2D, PM\}$ est le mécanisme de SO.

Les paramètres du contrôleur sont ceux du forgeron et ceux des *Sélections d'Opérateurs* M2 et M2C. Concernant le forgeron, le registre de crédit a une taille fixe de 20, c'est-à-dire, à tout moment l'AE compte avec 20 opérateurs. Toutes les 50 générations, l'analyseur du forgeron est appelé afin de trouver un opérateur faible et de le remplacer. Afin de éliminer des opérateurs suffisamment connus, nous considérons uniquement ceux qui ont été appliquées $\frac{1}{2}$ fois la taille de la fenêtre glissante, i.e., 5 fois. Si la récompense d'un des opérateurs connus est dans le tiers inférieur, par rapport à celle des autres, il est choisi pour être éliminé.

Les paramètres du M2 sont $p = 0.2$ et $x = 1.5$. M2C emploie les données des dernières 100 générations pour calculer la régression linéaire. La période de diversification est déclenchée quand la valeur de la pente est dans l'intervalle ± 0.0001 et la différence entre les valeurs maximales et minimales est inférieur à 0.001.

5.4 Résultats et discussion

5.4.1 Choix du contrôleur

Nous présentons ici les résultats expérimentaux que nous avons obtenus avec notre approche, en essayant d'en souligner les aspects les plus intéressants. La figure 5.5 montre la convergence du meilleur individu pour différentes configurations des contrôleurs et des croisements de l'état de l'art pour l'instance *ibm*.

La figure 5.6.a montre la diversité de population produite par les contrôleurs qui obtiennent des résultats similaires (RP-A et DP-M2). Notons que les contrôleurs qui obtiennent des niveaux semblables en termes de qualité, ne produisent pas nécessairement le même niveau de diversité. Ceci illustre des comportements différents : tandis que DP-M2 produit une exploitation forte qui améliore rapidement la qualité jusqu'à la génération 30000, RP-A explore pour produire des améliorations plus lentes mais constantes tout au long de la recherche.

La figure 5.6.b montre la diversité de la population produite par les contrôleurs qui ont produit les meilleurs résultats (DP-PP, DP-A), ainsi que celle produite par les croisements de l'état de l'art. On peut observer un niveau intermédiaire de diversité dans les meilleures configurations, notamment dû à leurs *Sélections d'Opérateurs* exploratoires (PP et A), permettant une rapide – bien que prudente – convergence pour améliorer les résultats.

La tendance à la hausse de la diversité pour DP-PP à partir de la génération 8.000 est due à l'inclusion des critères de diversification dans l'évaluation des opérateurs. Au début de la recherche, la population initiale se compose des individus produits aléatoirement, ceci favorise la tâche des opérateurs d'exploration, qui diminuent la diversité pendant que la qualité augmente. Cependant, à partir d'un moment donné, l'amélioration devient difficile, ainsi le contrôleur se tourne vers l'exploration, dans le but d'éloigner les individus des optimums locaux. Ce comportement est précisément celui que nous avons cherché en incluant la diversité parmi les mesures envoyées au contrôleur.

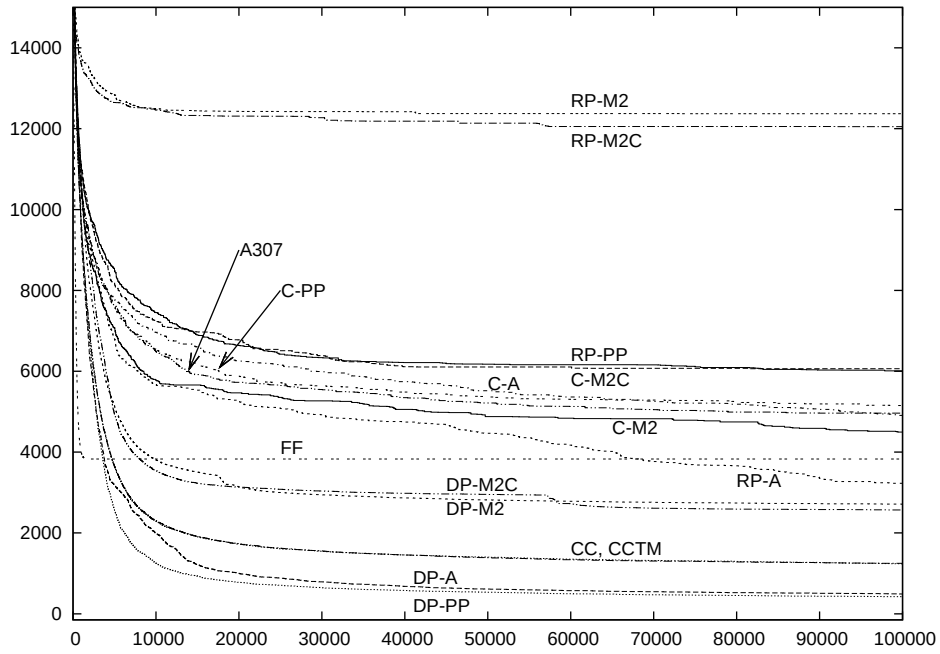

FIG. 5.5 – Nombre de clauses fausses du meilleur individu, obtenue par différents contrôleurs et croisements de l'état de l'art, sur l'instance *ibm*

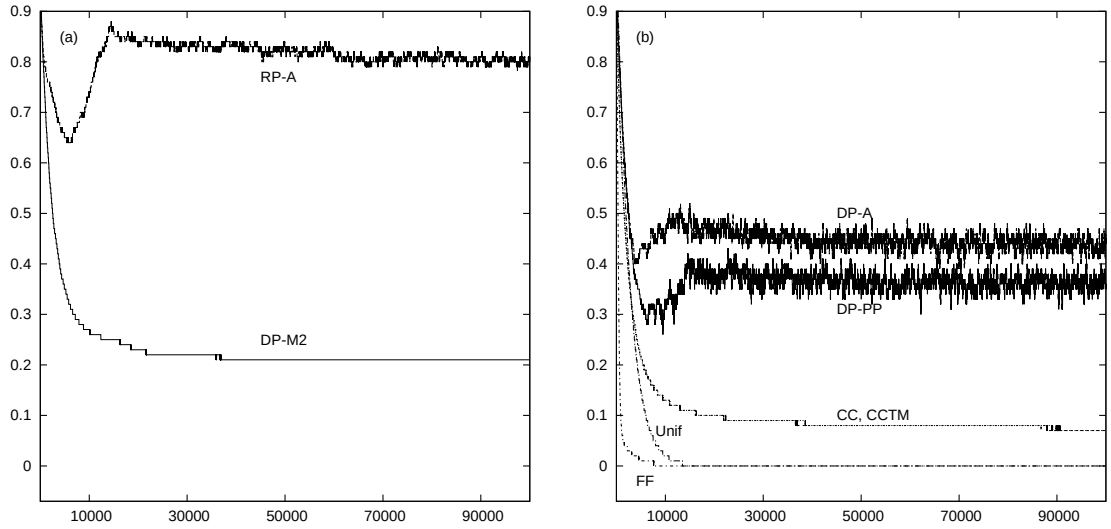

FIG. 5.6 – Diversité produite par des différentes méthodes en résolvant l'instance *ibm*. (a) qui obtiennent des résultats similaires, (b) qui obtiennent les meilleurs résultats, et des croisements de l'état de l'art

Cette tendance est particulièrement forte dans les *Affectations de Crédits* C et PR. La figure 5.7 montre l'évolution de la diversité pour différentes méthodes d'*Affectation de Crédits* : C, DP et RP, utilisés ensemble avec la *Sélection d'Opérateurs* A, sur l'exemple *simon*. Dans cette figure, on peut apprécier l'exploration excessive induite par C, qui pourrait expliquer pourquoi ses résultats ne sont pas les meilleurs de la série.

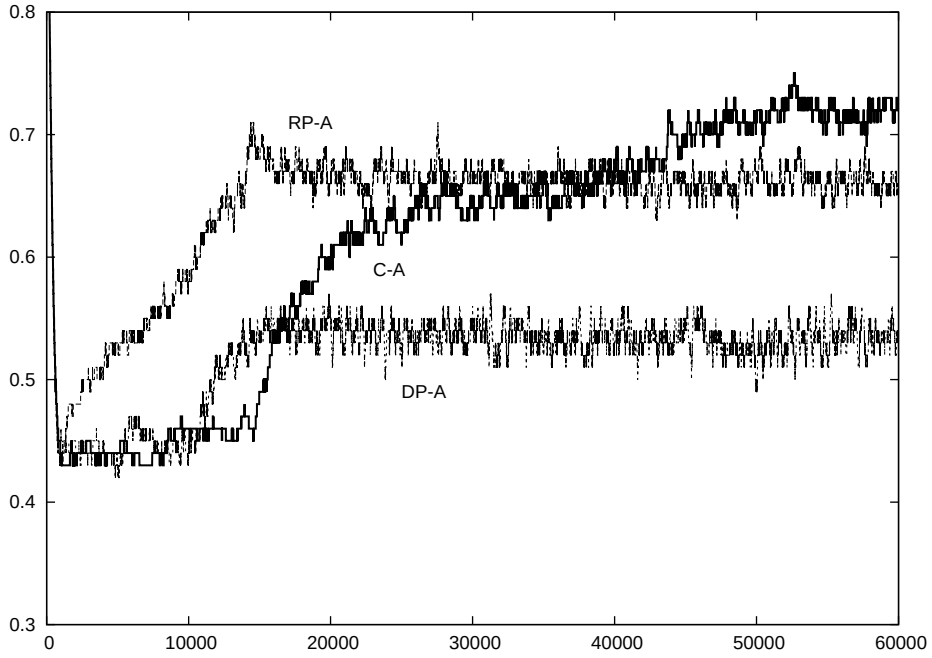

Fig. 5.7 – Diversité produite par ACs C, DP et RP, ensemble avec la SO A, en montrant le passage de l'exploitation à l'exploration

La table 5.1 montre le nombre moyen de clauses fausses et leur écart type (entre parenthèses) sur 50 exécutions des différents contrôleurs et des croisements de l'état de l'art FF, CC et CCTM. Les meilleurs résultats (et ceux qui sont indistingables, en utilisant un test T de Student avec 95% de confiance) apparaissent en gras.

Les meilleurs résultats ont été obtenus par les configurations DP-PP et DP-A, qui surpassent les croisements de l'état de l'art dans 7 des 8 instances. La table 5.2 montre le pourcentage d'amélioration de DP-PP et DP-A par rapport aux croisements de l'état de l'art.

Le croisement uniforme a produit de loin les plus mauvais résultats, raison pour laquelle nous l'ignorerons à partir de maintenant. On peut noter que les contrôleurs DP-PP et DP-A produisent des résultats comparables à ceux obtenus avec les meilleurs opérateurs sans contrôleur (CC et CCTM). Cependant, remarquons que le développement de CC et de CCTM (les meilleurs croisements) se fonde sur un travail de plusieurs semaines de comparaisons, d'analyses et d'expériences [Lardeux *et al.*, 2006], alors que le contrôleur effectue un travail semblable en quelques minutes et, ce qui est plus important, sans intervention humaine. Nous voulons également remarquer la performance de contrôleurs

TAB. 5.1 – Nombre de clauses fausses et écart type

	f500		aim-100		ibm		simon	
C-M2	25.9	(24.0)	2.4	(1.9)	6009.7	(3024.6)	189.0	(17.3)
C-M2C	16.8	(19.7)	2.6	(1.9)	6063.4	(2171.8)	194.4	(23.9)
C-PP	56.3	(33.6)	2.2	(1.9)	5151.9	(2758.8)	209.2	(41.8)
C-A	21.0	(14.4)	1.1	(0.2)	4908.4	(1623.0)	183.7	(16.7)
DP-M2	67.4	(62.7)	3.3	(2.2)	2712.0	(3523.9)	94.3	(103.0)
DP-M2C	53.3	(59.0)	2.5	(1.5)	2567.1	(4206.3)	107.9	(102.5)
DP-PP	**6.0**	(1.4)	**1.0**	(0.0)	**423.8**	(75.2)	93.5	(7.7)
DP-A	**5.6**	(1.2)	**1.0**	(0.0)	491.1	(66.9)	102.9	(9.7)
RP-M2	98.2	(53.8)	2.9	(1.7)	12370.3	(5214.2)	201.0	(188.7)
RP-M2C	104.2	(52.5)	2.6	(1.8)	12050.3	(5141.1)	277.9	(200.0)
RP-PP	7.8	(1.4)	**1.0**	(0.0)	4495.9	(791.9)	148.9	(11.9)
RP-A	6.9	(1.1)	**1.0**	(0.0)	3228.6	(913.8)	145.2	(9.2)
A307	49.4	(4.3)	1.0	(0.0)	4962.7	364.9)	187.4	(11.7)
Unif	218.4	(5.7)	11.2	(1.0)	34149.6	(138.5)	2872.6	(33.9)
FF	30.2	(4.9)	1.9	(0.6)	3827.8	(160.5)	137.5	(9.7)
CC	7.2	(1.3)	1.9	(0.6)	1247.7	(98.7)	**81.6**	(5.4)
CCTM	7.3	(1.4)	1.8	(0.6)	1237.2	(78.1)	**81.2**	(5.3)

	bw-large.d		flat200-19		uf250		engine	
C-M2	427.1	(287.1)	54.4	(40.1)	12.3	(13.9)	761.6	(477.3)
C-M2C	596.3	(329.1)	40.7	(37.4)	7.5	(11.5)	1045.0	(369.8)
C-PP	650.5	(816.6)	67.3	(41.9)	21.9	(15.2)	752.8	(490.4)
C-A	306.4	(194.4)	52.5	(28.9)	6.7	(5.8)	577.3	(262.3)
DP-M2	1575.6	(1697.3)	58.3	(44.9)	30.8	(23.0)	838.9	(836.3)
DP-M2C	1553.9	(1804.1)	52.7	(48.4)	26.3	(25.2)	911.4	(824.0)
DP-PP	**78.1**	(3.2)	10.7	(2.1)	2.2	(1.3)	**15.4**	(3.3)
DP-A	83.2	(3.5)	**9.2**	(2.1)	**1.5**	(0.9)	18.4	(3.1)
RP-M2	2455.1	(1678.0)	100.7	(56.7)	41.0	(24.3)	1267.1	(966.1)
RP-M2C	2367.9	(1713.0)	99.8	(54.0)	34.9	(24.0)	1064.7	(964.8)
RP-PP	187.9	(146.9)	31.6	(20.5)	**1.7**	(0.8)	462.1	(328.8)
RP-A	272.5	(268.5)	16.3	(10.5)	**1.5**	(0.5)	415.6	(302.4)
A307	207.5	(22.9)	25.3	(7.2)	17.8	(2.6)	534.8	(54.0)
Unif	27237.3	(301.9)	408.7	(20.6)	98.8	(3.6)	12663.6	(289.1)
FF	126.6	(10.6)	44.9	(4.4)	15.1	(3.4)	465.3	(49.8)
CC	579.0	(17.6)	12.0	(2.1)	3.4	(1.4)	67.9	(12.8)
CCTM	580.3	(17.6)	12.7	(2.1)	3.2	(1.2)	68.9	(11.4)

DP-PP et DP-A sur les instances industrielles *ibm* et *engine*, où le nombre moyen de clauses fausses est équivalent à au moins $\frac{1}{3}$ de ceux obtenus avec CC et CCTM.

En comparant les différentes méthodes d'*Affectation de Crédits* , nous notons que DP apparaît dans les contrôleurs les plus efficaces, suivi de C et puis de RP. Nous pouvons nous interroger sur la différence entre les deux contrôleurs basés sur les notions de la dominance de Pareto et pourquoi DP marche mieux que les deux autres. Pour répondre à cette question, il est nécessaire d'étudier leur comportement pendant des exécutions individuelles. La figure 5.8 montre la qualité moyenne de la population en utilisant DP-M2, RP-M2 et C-M2 sur l'instance *ibm*.

Tab. 5.2 – Amélioration des contrôleurs DP-PP et DP-A, par rapport aux croisements de l'état de l'art

DP-PP	f500	aim-100	ibm	simon	bw-large.d	flat200	uf250-010	engine-4-nd
Unif	97.23%	91.07%	98.76%	96.75%	99.71%	97.37%	97.79%	99.88%
FF	80.01%	47.92%	88.93%	31.99%	38.34%	76.09%	85.58%	96.68%
CC	16.57%	47.92%	66.03%	-14.62%	86.52%	10.20%	35.50%	77.25%
CCTM	17.03%	45.05%	65.74%	-15.15%	86.55%	15.17%	31.01%	77.59%

DP-A	f500	aim-100	ibm	simon	bw-large.d	flat200	uf250-010	engine-4-nd
Unif	97.44%	91.07%	98.56%	96.42%	99.69%	97.75%	98.48%	99.85%
FF	81.46%	47.37%	87.17%	25.16%	34.28%	79.51%	90.07%	96.05%
CC	22.22%	47.37%	60.64%	-26.10%	85.63%	23.33%	55.88%	72.90%
CCTM	23.29%	44.44%	60.31%	-26.72%	85.66%	27.56%	53.12%	73.29%

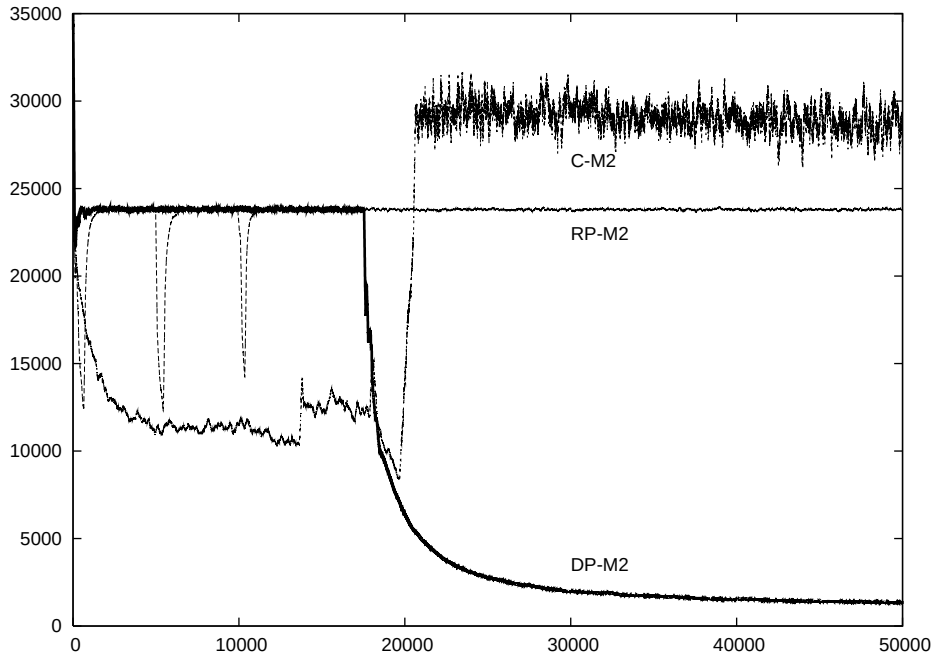

Fig. 5.8 – Comparaison entre le Rang Pareto et la Dominance de Pareto : Nombre moyen de clauses fausses sur l'instance *ibm*

Le RP considère de la même manière tous les opérateurs placés sur le front Pareto (points dans la figure 5.2.c avec valeur 0), en induisant un équilibre entre les tendances à explorer et à exploiter et en empêchant l'AE de pencher d'un côté ou de l'autre. Notons que les tentatives d'augmenter la qualité de RP-PP sont modérées par cet équilibre, forçant la qualité moyenne à revenir à une valeur intermédiaire. Un comportement semblable peut être observé pour Compass, étant donné sa méthode de mesure de performance. D'autre part, en utilisant la DP, la meilleure appréciation des opérateurs qui suivent la tendance

générale (points dans la figure 5.2.b avec des valeurs élevées), permet à l'AE de casser le *status quo* et finalement d'améliorer la qualité de la population. Cet "équilibre flexible" constitue l'atout principal de cette méthode d'*Affectation de Crédits* .

Il est intéressant de noter que les *Sélections d'Opérateurs* les plus exploratoires (PP et A) ont produit certains des meilleurs résultats. Il peut paraître étonnant – et contradictoire avec les résultats obtenus lors du dernier chapitre – qu'une *Sélection d'Opérateurs* aléatoire puisse surpasser des méthodes sophistiquées qui essayent soigneusement d'équilibrer l'EvE au niveau de sélection d'opérateur. Une hypothèse possible pour ces bons résultats est que la combinaison des croisements fonctionne mieux que l'application d'un seul, pourtant les faibles résultats obtenus par A307 prouvent que ceci n'en est pas la cause.

Une meilleure explication est que lorsque le Forgeron analyse l'ensemble des opérateurs pour remplacer certains d'entre eux, il choisit toujours les plus mauvais. Ceci est alors une action en faveur de l'exploitation, basée sur les récompenses stockées dans le Registre de Crédit. Le fait que le contrôleur choisisse constamment les opérateurs qui seront disponibles pour l'AE produit un déplacement de l'exploitation au niveau de l'EvE depuis la *Sélection d'Opérateurs* vers le Forgeron. Dans ce nouveau scénario, la *Sélection d'Opérateurs* est seulement responsable de l'exploration, qui est limitée aux opérateurs que le Forgeron laisse exister.

En général, nous pouvons conclure que l'aspect le plus important du contrôleur est le réglage des paramètres structuraux (i.e., le choix des opérateurs qui seront disponibles pendant la course), suivie par la façon dont ces opérateurs sont évalués (i.e., la méthode d'*Affectation de Crédits*) et finalement comment ils sont appliqués (mécanisme de *Sélection d'Opérateurs*).

5.4.2 Validation de généralité

Afin de vérifier la généralité des contrôleurs DP-PP et DP-A, nous avons étendu la comparaison avec les croisements de l'état de l'art sur un ensemble de 26 exemples supplémentaires, en incluant des instances faites-main, aléatoires et industrielles [1]. La table 5.3 montre la moyenne et l'écart type (entre parenthèses) de 25 exécutions de 40.000 générations chacune. Les meilleurs résultats (et ceux qui sont indistingables, en utilisant un test T de Student avec un 95% de confiance) apparaissent en gras.

[1] Les instances faites-main sont ezfact64_3.sat05-450.reshuffled-07 (C1), ezfact64_4.sat05-451.reshuffled-07 (C2), ezfact64_5.sat05-452.reshuffled-07 (C3), ezfact64_6.sat05-453.reshuffled-07 (C4), hgen2-v500-s1216665065.sat05-467.reshuffled-07 (C5), hgen3-v400-s344840348.sat05-470.reshuffled-07 (C6), hgen3-v400-s553296708.sat05-471.reshuffled-07 (C7), hgen3-v500-s1349121860.sat05-473.reshuffled-07 (C8), pyhala-braun-unsat-40-4-02.sat05-459.reshuffled-07 (C9). Les instances aléatoires sont unif2p-p0.9-v630-c2280-S1071799860-07-UNSAT (R1), unif2p-p0.9-v630-c2280-S1244126495-18-SAT (R2), unif2p-p0.9-v630-c2280-S1501024241-13-UNSAT (R3), unif2p-p0.9-v630-c2280-S1788789488-19-SAT (R4), unif-k3-r4.261-v650-c2769-S1089058690-02.SAT.shuffled (R5), unif-k3-r4.261-v650-c2769-S1159448555-06.SAT.shuffled (R6), unif-k3-r4.261-v650-c2769-S1172355929-14.SAT.shuffled (R7), unif-k3-r4.261-v650-c2769-S1341479044-12.UNSAT.shuffled (R8), unif-k3-r4.261-v650-c2769-S1470952774-07.SAT.shuffled (R9), unif-k3-r4.2-v10000-c42000-S1173369833-06 (R10). Les instances industrielles sont AProVE07-03 (I1), AProVE07-21 (I2), eq.atree.braun.13.unsat (I3), eq.atree.braun.10.unsat (I4), vmpc_26 (I5), vmpc_24 (I6), sortnet-6-ipc5-h11-unsat (I7).

Tab. 5.3 – Comparaison de DP-PP et de DP-A avec les croisements de l'état de l'art sur les instances faites-main, aléatoires et industrielles. Nombre de clauses fausses et écart type

	DP-PP		DP-A		FF		CC		CCTM	
C1	**35.4**	(5.4)	**34.8**	(2.8)	503.2	(41.0)	44.7	(5.2)	42.1	(4.7)
C2	**35.8**	(2.6)	38.0	(4.2)	509.4	(31.6)	46.0	(4.4)	47.6	(4.9)
C3	**35.4**	(3.7)	**35.6**	(3.6)	490.0	(37.7)	48.4	(4.1)	47.1	(3.3)
C4	45.1	(3.8)	**43.4**	(4.6)	491.6	(36.5)	48.7	(3.0)	48.2	(3.4)
C5	**10.5**	(1.8)	**9.8**	(2.8)	47.9	(4.2)	11.6	(1.8)	10.2	(1.5)
C6	**8.6**	(1.9)	**8.3**	(1.7)	36.9	(3.3)	**8.4**	(1.6)	**8.7**	(1.4)
C7	8.8	(1.8)	**8.0**	(1.9)	38.7	(4.2)	**8.4**	(1.2)	**8.7**	(1.7)
C8	**10.0**	(2.4)	**9.7**	(2.5)	48.2	(4.1)	11.3	(1.4)	11.6	(1.6)
C9	150.9	(31.2)	**123.3**	(28.8)	973.2	(77.4)	214.7	(15.9)	217.0	(14.8)
R1	**7.5**	(1.5)	**7.2**	(1.1)	34.2	(5.4)	9.5	(1.9)	9.7	(1.8)
R2	6.4	(1.3)	**5.7**	(1.4)	30.6	(3.8)	7.3	(1.4)	7.7	(1.6)
R3	**8.4**	(1.4)	**8.2**	(1.5)	32.1	(3.8)	10.6	(1.6)	10.9	(1.9)
R4	4.2	(1.5)	**3.5**	(1.4)	26.3	(3.8)	7.4	(1.2)	7.4	(1.8)
R5	**8.2**	(2.1)	**7.8**	(1.8)	40.0	(6.0)	**8.4**	(1.5)	9.1	(1.4)
R6	**6.7**	(1.6)	**7.9**	(1.6)	44.2	(6.4)	8.7	(1.5)	8.8	(1.4)
R7	**6.1**	(1.7)	**5.8**	(2.1)	39.4	(5.5)	7.6	(1.6)	7.8	(1.4)
R8	**9.0**	(1.2)	**8.8**	(1.6)	49.2	(5.3)	10.3	(1.9)	9.9	(1.7)
R9	**9.1**	(1.6)	**9.0**	(1.7)	41.9	(5.7)	10.0	(1.7)	**9.0**	(1.5)
R10	**110.1**	(5.7)	115.1	(8.3)	654.0	(39.5)	153.0	(9.2)	150.0	(7.9)
I1	**123.6**	(11.4)	167.6	(32.3)	439.3	(27.3)	354.4	(11.4)	349.6	(11.7)
I2	**99.7**	(8.2)	134.7	(22.5)	469.1	(26.3)	372.0	(35.5)	367.8	(32.2)
I3	2.7	(2.9)	8.6	(7.7)	216.5	(18.9)	**1.0**	(0.0)	**1.0**	(0.0)
I4	2.8	(2.4)	6.3	(4.8)	116.2	(11.2)	**1.0**	(0.2)	**1.1**	(0.4)
I5	59.4	(98.5)	**38.0**	(1.4)	12567.6	(547.1)	10044.2	(384.4)	9928.1	(382.0)
I6	127.8	(317.1)	**35.1**	(1.5)	9736.2	(404.9)	7567.7	(238.0)	7521.2	(272.9)
I7	**44.2**	(1.3)	48.4	(2.2)	1877.6	(195.1)	61.8	(1.7)	61.6	(1.8)

DP-A obtient des résultats supérieurs dans 19 instances et DP-PP dans 17, alors que CC et CCTM seulement dans 5 fois. Même si les configurations de contrôleur ont obtenu des mauvais résultats sur deux instances industrielles, nous observons les meilleures améliorations sur cette famille, particulièrement sur I5 et I6, où les AEs contrôlés ont obtenu jusqu'à 260 fois moins de clauses fausses par rapport aux croisements de l'état de l'art. Les résultats présentés dans cette comparaison confirment la généralité de DP-PP et de DP-A sur différentes familles et types d'instances. Les coûts en temps d'exécution de cette amélioration sont vraiment modestes, puisque le temps consacré au contrôle représente moins de 10% du temps total d'exécution.

5.5 Conclusion du chapitre

Dans ce chapitre nous avons étendu la SAO présentée précédemment, afin d'employer un nombre considérable d'opérateurs dans un AE. Comme précédemment, les contrôleurs sont basés sur l'idée de considérer la qualité et la diversité produites par des opérateurs

en tant qu'objectifs principaux de tout AE.

Le contrôleur autonome gère les opérateurs de l'AE à deux niveaux :

- il décide quels opérateurs seront inclus et utilisés dans l'AE.
- il choisit, parmi ceux disponibles, quel opérateur sera appliqué à chaque étape de la recherche.

Afin de réaliser ces tâches, le contrôleur se compose de deux modules. Le premier, la *Sélection adaptative d'Opérateur*, reçoit la rétroaction de l'AE afin de mettre à jour le Registre de Crédit, qui est employé plus tard pour choisir l'opérateur qui s'appliquera. Le deuxième module, appelé *Forgeron*, décide quels opérateurs seront inclus dans l'AE, basé sur leurs performances respectives.

Nous avons examiné plusieurs contrôleurs avec différentes configurations de SAO, sur un AE pour la résolution du problème de satisfiabilité SAT. Plus de 300 opérateurs de croisement ont été livrés au contrôleur, et les résultats ont été comparés aux croisements de l'état de l'art, conçus dans des études spécifiques préalables. Les comparaisons sur des instances provenant de différentes familles et types de jeux d'essai (faites-main, aléatoires et industrielles) ont montré un avantage évident pour deux configurations de contrôleur (DP-PP et DP-A). Sur 34 instances, DP-PP a obtenu les meilleurs résultats dans 27 cas, des résultats similaires dans 4 et moins bons dans seulement 3. Pour le même nombre d'instances, DP-A a obtenu les meilleurs résultats dans 28 cas, semblables dans 3 et pires dans 3.

La contribution principale de ce travail est notre cadre générique qui permet de choisir quel opérateur appliquer et de décider quels opérateurs seront disponibles dans l'AE à chaque moment. Nous avons également défini une nouvelle méthode d'Affectation de Crédit (DP), qui combine plusieurs critères en considérant la tendance générale des opérateurs. Nous avons aussi proposé une méthode de Sélection d'Opérateur capable de traiter un ensemble dynamique d'opérateurs (M2). Notre méthode pourrait être facilement appliquée à d'autres algorithms basées sur des populations, et même aux approches basées sur un seul individu, où la diversit de la population pourrait tre remplacée par une diversité temporelle, qui évalue la variation des configurations visitées par l'individu unique.

Le composant Forgeron, conjointement avec l'Affectation de Crédit, joue le rôle de l'exploitation au niveau du choix d'opérateur, qui est réalisé par la SO lorsqu'on utilise des paramètres statiques. Dans ce travail, nous avons utilisé une configuration très simple pour le Forgeron, qui garde un ensemble d'opérateurs de taille fixe. Étant donné l'importance de la commande des paramètres structuraux, un travail supplémentaire est nécessaire afin de développer des configurations plus sophistiquées, par exemple en employant la programmation génétique pour générer des opérateurs.

Dans la plupart des exemples, nous avons noté une stabilisation des niveaux de diversité et de qualité à partir de la génération 30.000. Cette situation, et le fait que nous employions les mêmes critères (comparaison des mesures de diversité et de qualité) pour effectuer la commande de paramètres aux niveaux structurel et comportemental, suggèrent que nous pourrions obtenir quelques améliorations en incorporant une stratégie de recherche [Maturana and Saubion, 2007a]. Cette stratégie pourrait inclure des critères complémentaires afin d'améliorer le réglage de l'exploration et l'exploitation dans l'espace de recherche du problème.

Afin d'évaluer l'intérêt du contrôleur en tant qu'outil pour la conception d'algorithmes, on pourrait inclure dans ce contrôleur un mécanisme qui identifie les opérateurs les plus performants. Il serait également intéressant de détecter les relations entre les opérateurs. Notons que nous n'avons pas identifié un croisement individuel plus performant que CC ou CCTM, il existe donc une possibilité que la bonne performance de DP-PP/A soit plutôt due à l'effet conjoint de plusieurs croisements, qu'à la présence d'un meilleur croisement.

Conclusion Générale

L'état actuel du développement des Algorithmes Évolutionnaires (AEs) peut être comparé à celui des automobiles au XIXème siècle. Ce sont principalement des prototypes expérimentaux, développés sur mesure et utilisés par des spécialistes. Cette utilisation passe notamment par le réglage des paramètres, présents parfois en grand nombre. Leur manque de généralité et leur difficulté d'utilisation compliquent leur application systématique aux problèmes réels.

La principale motivation de cette thèse a été la création d'un contrôleur générique et facile à intégrer dans un algorithme de recherche, afin de permettre à un utilisateur non-spécialiste de se focaliser sur sa tâche principale, c'est-à-dire, résoudre son problème.

Afin de créer un tel contrôleur nous avons dû faire abstraction du processus de recherche, pour nous concentrer sur les principes fondamentaux des algorithmes. Ces principes sont principalement : bien parcourir l'espace de recherche, afin d'identifier les meilleures zones (exploration), et bien inspecter les zones prometteuses, afin d'en tirer le plus grand bénéfice possible (exploitation). L'exploration est typiquement associée à des populations diverses (i.e., étendues dans l'espace de recherche), tandis que la exploitation est associée à des populations de haute qualité. Nous avons donc considéré ces deux mesures – diversité et qualité – en tant qu'indicateurs, pour connaître l'état courant de l'algorithme.

Différentes approches ont été étudiées, notamment dans le cadre des AEs. D'abord, nous avons proposéune méthode permettant de comprendre l'effet des paramètres sur la recherche. Pour ce faire, la méthode essaye d'abord plusieurs combinaisons de valeurs de paramètres, et construit un modèle qui permet d'identifier les combinaisons optimales pour obtenir plusieurs niveaux de diversité. Dans un deuxième temps, ces valeurs optimales sont utilisées pour guider la recherche, en réglant le niveau de diversité (intimement lié à l'équilibre entre l'exploration et l'exploitation) à l'aide d'une stratégie de contrôle.

Cette approche implémente une abstraction des paramètres, simplifiant le contrôle grâce au réglage du paramètre unique. De plus, les modèles obtenus, facilement compréhensibles par les utilisateurs, peuvent être utilisés afin de connaître le fonctionnement interne de l'AE. Malgré ces avantages, cette méthode nécessite un nombre exponentiel d'expériences par rapport à la quantité de paramètres contrôlés, ce qui limite son application à un nombre réduit de paramètres.

Un deuxième approche se focalise sur l'efficacité du contrôle, en laissant de côté l'aspect lié à l'analyse. Nous avons proposé la méthode *Compass*, qui prend en compte les deux critères dejà utilisés (diversité et qualité) et additionnellement le temps d'exécution, afin de comparer les opérateurs appliqués dans un AE. Cette approche, articulée en tant que *Selection Adaptive d'Opérateurs (SAO)*, permet de considérer plus de paramètres, et implémente un apprentissage mis a jour constamment. Des modifications de *Compass* ont été étudiées, notamment en le combinant avec une méthode issue de la théorie de jeux, utilisée précédemment dans le cadre du contrôle des AEs.

Finalement, une extension de la dernière approche a été proposée, afin de modifier

la structure de l'algorithme contrôlé pendant son exécution. Le *Forgeron* gère l'ensemble d'opérateurs disponibles à l'AE, en le modifiant selon les nécessités courantes de la recherche. Les résultats montrent que cette approche peut obtenir en quelques minutes des performances comparables à celles obtenues avec des opérateurs qui ont nécessité un étude approfondie être définis. Nous avons aussi proposé une architecture qui vise à standardiser l'implémentation des contrôleurs génériques.

Perspectives de Recherche

Plusieurs perspectives s'ouvrent suite à ce travail. La principale est peut-être l'adaptation des principes utilisés ici (généralité, simplicité) au contrôle des différents types d'algorithmes de recherche et d'optimisation. On peut envisager, par exemple, de remplacer la diversité de la population par une mesure équivalente afin d'adapter cette méthode aux heuristiques qui travaillent avec un seul individu.

Bien que le *Forgeron* ait été conçu pour la gestion des paramètres structurels, notre recherche s'est limitée à étudier le choix parmi plusieurs opérateurs de croisement. Une paramétrisation plus ample, qui permettrait à l'AE de transformer plus radicalement sa structure, peut s'avérer une recherche très intéressante. On peut penser par exemple à la façon de sélectionner les parents ou comment les fils seront insérés dans la population. On pourrait avoir aussi des opérateurs qui modifient la taille de la population, ou bien les mécanismes d'évaluation. Le fait de considérer tous ces éléments en tant qu'opérateurs facilite la tâche de conception de l'algorithme, qui pourrait en autre s'automatiser à l'aide de la programmation génétique, où les opérateurs évolueraient au même temps que la recherche avance.

L'étude peut être approfondie en ce qui concerne le *Forgeron*. Vu son impact dans le contrôle, il est envisageable d'intégrer de dispositifs qui incluent des stratégies de recherche basées sur des critères complémentaires à ceux pris en compte par la *SAO*.

Le fait d'étudier la conception des méthodes de contrôle génériques pose la question de traiter des bancs d'essai réunissant des problèmes variés. Pour nos expériences, nous avons utilisé des instances de QAP et de SAT. Bien qu'ambitieux, il serait alors très intéressant de créer un banc d'essai avec des problèmes de tout type.

Liste de publications personnelles

Revues internationales

Jorge Maturana and Frédéric Saubion, "*From parameter control to search control : Parameter Control Abstraction in Evolutionary Algorithms*", Constraint Programming Letters, Special Issue on Autonomous Search, Volume 4. pp. 39-65, 2008.

Conférences internationales avec comité de sélection

Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer and Michèle Sebag "*Compass and Dynamic Multi-Armed Bandits for Adaptive Operator Selection*", Proceedings of IEEE International Conference on Evolutionary Computation (CEC'2009), May 2009, Norway. (à paraître).

Jorge Maturana and Frédéric Saubion "*A Compass to Guide Genetic Algorithms*", Lecture Notes in Computer Science, Volume 5199 pp. 256-265, Proceedings of the 1Oth International Conference on Parallel Problem Solving from Nature (PPSN X), September 2008, Dortmund, Germany.

Jorge Maturana and Frédéric Saubion "*On the Design of Adaptive Control Strategies for Evolutionary Algorithms*", Lecture Notes in Computer Science (LNCS), Volume 4926, pp. 303-315, Springer-Verlag. Evolution Artificielle Conference (EA'2007), October 2007, Tours, France.

Jorge Maturana and Frédéric Saubion "*Towards a Generic Control Strategy for Evolutionary Algorithms : an Adaptive Fuzzy-Learning Approach*", Proceedings of IEEE International Conference on Evolutionary Computation (CEC'2007), pp. 4546-4553. September 2007, Singapore.

Conférences nationales avec comité de sélection

Jorge Maturana, Frédéric Lardeux and Frédéric Saubion, "*Génération et contrôle autonomes d'opérateurs pour les algorithmes évolutionnaires*", Actes des Cinquièmes Journées Francophones de Programmation par Contraintes, pp. 185-194, June 2009, Orléans, France.

Ateliers

Jorge Maturana and Frédéric Saubion "*Automated Parameter Control for Evolutionary Algorithms*", First Workshop on Autonomous Search In conjunction with CP'2007. September 2007, Providence, Rhode Island, USA.

En soumission

Jorge Maturana, Frédéric Lardeux and Frédéric Saubion, "*Autonomous Operator Management for Evolutionary Algorithms*", Journal of Heuristics, Special Issue on Hyperheuristics in Search and Optimisation, 2009.

Jorge Maturana, Frédéric Lardeux and Frédéric Saubion "*Controlling behavioral and structural parameters in Evolutionary Algorithms"*, 9th international conference on Artificial Evolution, October 2009, Orléans, France.

Annexes

Annexe A

Apperçu de l'implémentation

Un des aspects intéressants de notre travail est en lien avec l'implémentation d'un contrôleur générique peu dépendant de l'agorithme contrôlé. Cette annexe décrit les principales structures utilisées pour implémenter un tel contrôleur, ainsi que leurs interactions. Le contrôleur décrit ici correspond à celui utilisé dans le chapitre 5.

Diagramme de classes

Notre contrôleur à été codé en utilisant un paradigme objet. Nous allons premièrement décrire un diagramme de classes simplifié, presenté dans la figure A.1. On peut d'abord noter qu'il existe une classe *Controller* qui est connecté à l'algorithme évolutionnaire (AE). En effet, l'AE doit avoir un objet de classe *Controller* qui fournira le service de contrôle de paramètres. Cette classe possède deux méthodes publiques : *which_next* et *feedback*. *which_next* retourne le nom de l'opérateur à appliquer, tandis que la méthode *feedback* reçoit les mésures de performance issues de l'application de l'opérateur (voir figure 5.1). Le Contrôleur est composé de trois objets des classes *Depot*, *AOS* (SAO) et *Blacksmith* (Forgeron).

Un objet de la classe *Depot* est chargé de stocker toute l'information des opérateurs. Il possède trois listes, afin de stocker les opérateurs dans les trois états possibles (*non-nés*, *vivants* et *morts*, voir figure 5.4). La liste des opérateurs non-nés est composée uniquement de leurs noms. Celle des opérateurs morts est composée d'objets de classe *profile*, et celle des opérateurs vivants comporte additionnellement des objets de classe *data* (voir section 5.2.2). Les détails de ces classes sont ignorés ici par simplicité.

Des méthodes publiques sont fournies dans le but de gérer les opérateurs vivants : *add_operator* fait naître ou ressuscite les opérateurs, *delete_operator* les tue. Le stockage d'informations dans les structures *data* et *profile* se fait à l'aide de la méthode *putdata*, et l'obtention de ces informations se fait à travers la méthode *getdata*. Finalement, la mèthode *update_profile* met à jour les profils à partir des données stockées dans les structures *data*.

Lorsque l'objet de classe *Controller* est créé par l'AE, il crée à son tour un objet de la classe *Depot*, qui est visible pour les objets des classes *AOS* et *Blacksmith*. Tous les deux gardent une réference au dépôt.

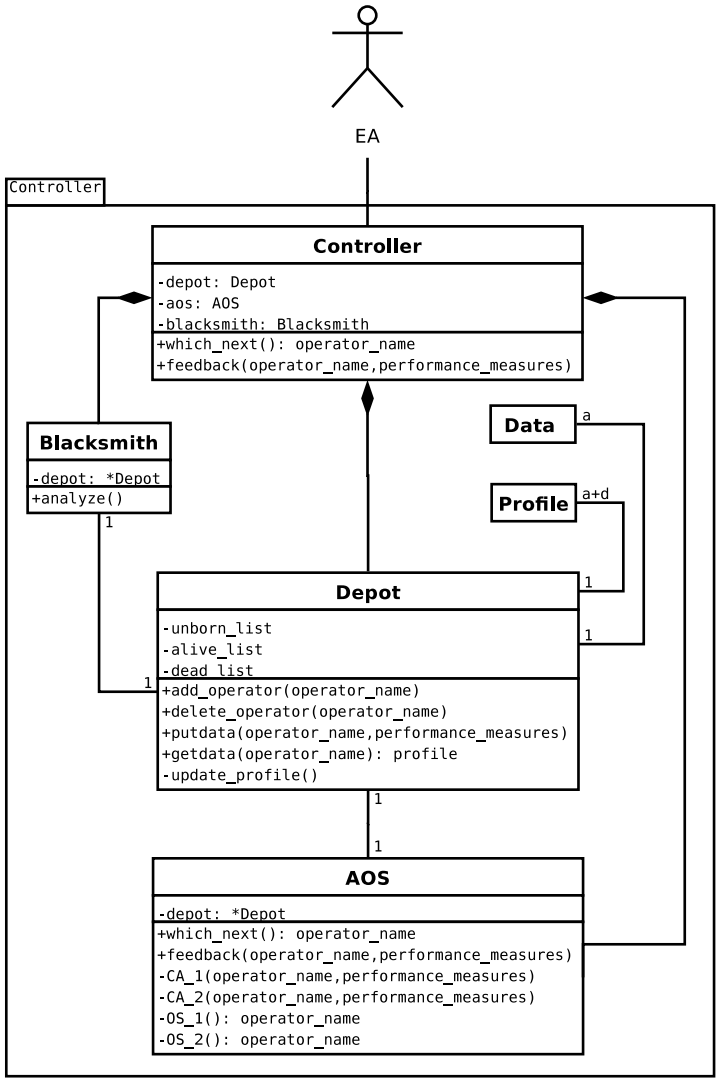

Fig. A.1 – Diagramme de classes

La classe *AOS* a deux méthodes publiques, *which_next* et *feedback*, qui sont utilisées lorsque l'AE appelle les méthodes du contrôleur. La méthode *feedback* correspond à l'implémentation de l'Affectation de Crédits (voir section 4.2.1), et retourne simplement le résultat des méthodes *CA_**, qui implémentent vraiment les fonctionalités d'affectation de crédits. Plusieurs méthodes d'affectation de crédits peuvent être implémentées à l'intérieur de cette classe. Un paramètre lu depuis un fichier de configuration du contrôleur détermine laquelle de ces méthodes doit être appelée par *feedback*.

La méthode *which_next*, pour sa part, correspond à l'implémentation de la Sélection d'Opérateurs et fonctionne de manière analogue à *feedback* en appelant les méthodes *OS_**.

La classe *Blacksmith* maintient aussi une réference au dépôt. La méthode *analyze* est

chargée d'analyser l'état du dépôt afin de modifier l'ensemble d'opérateurs vivants, et éventuellement ajouter des opérateurs (les faire naître ou ressuciter), ou bien les éliminer (tuer).

Diagramme de séquences

Passons maintenant au diagramme de séquences, montré dans la figure A.2. Ce diagramme montre la communication entre les classes lors des trois appels depuis l'AE : la création de l'objet *controller* et l'appel aux méthodes *which_next* et *feedback*.

Lorsque l'AE crée l'objet *controller*, ce dernier crée à son tour l'objet *depot*, puis l'objet *blacksmith*, qui fait naître l'ensemble initial d'opérateurs. Finalement, l'objet *aos* est crée.

Chaque fois que l'AE appelle la méthode *which_next* du contrôleur, celui-ci appelle la mèthode *wich_next* d'*aos*. Cette dernière appelle la méthode de sélection d'opérateurs *OS_** spécifié par un méta-paramètre du contrôleur. La méthode de sélection d'opérateurs determine l'opérateur à appliquer, en se basant sur les profils stockés dans le dépôt. La réponse est transmisse à *controller*, puis à l'AE.

Une fois l'opérateur appliqué, l'AE transmet au contrôleur les résultats de l'opération, en utilisant la méthode *feedback* du contrôleur. Le contrôleur appelle la méthode *putdata* du dépôt, afin d'enregistrer ces mesures. Ensuite, le contrôleur appelle la méthode feedback d'*aos*, qui à son tour appelle la méthode d'affectation de crédits *CA_** indiquée dans un méta-paramètre du contrôleur. Cette méthode calcule la récompense obtenue par l'opérateur et appelle la méthode *putdata* du dépôt, cette fois pour stocker la récompense calculée. Le dépôt met à jour le profil de l'opérateur en appelant sa méthode *update_profile*. Ceci fait, le contrôleur appelle la méthode *analyze* du forgeron, qui obtient les profils des opérateurs afin de décider s'il faut ajouter ou effacer un d'eux.

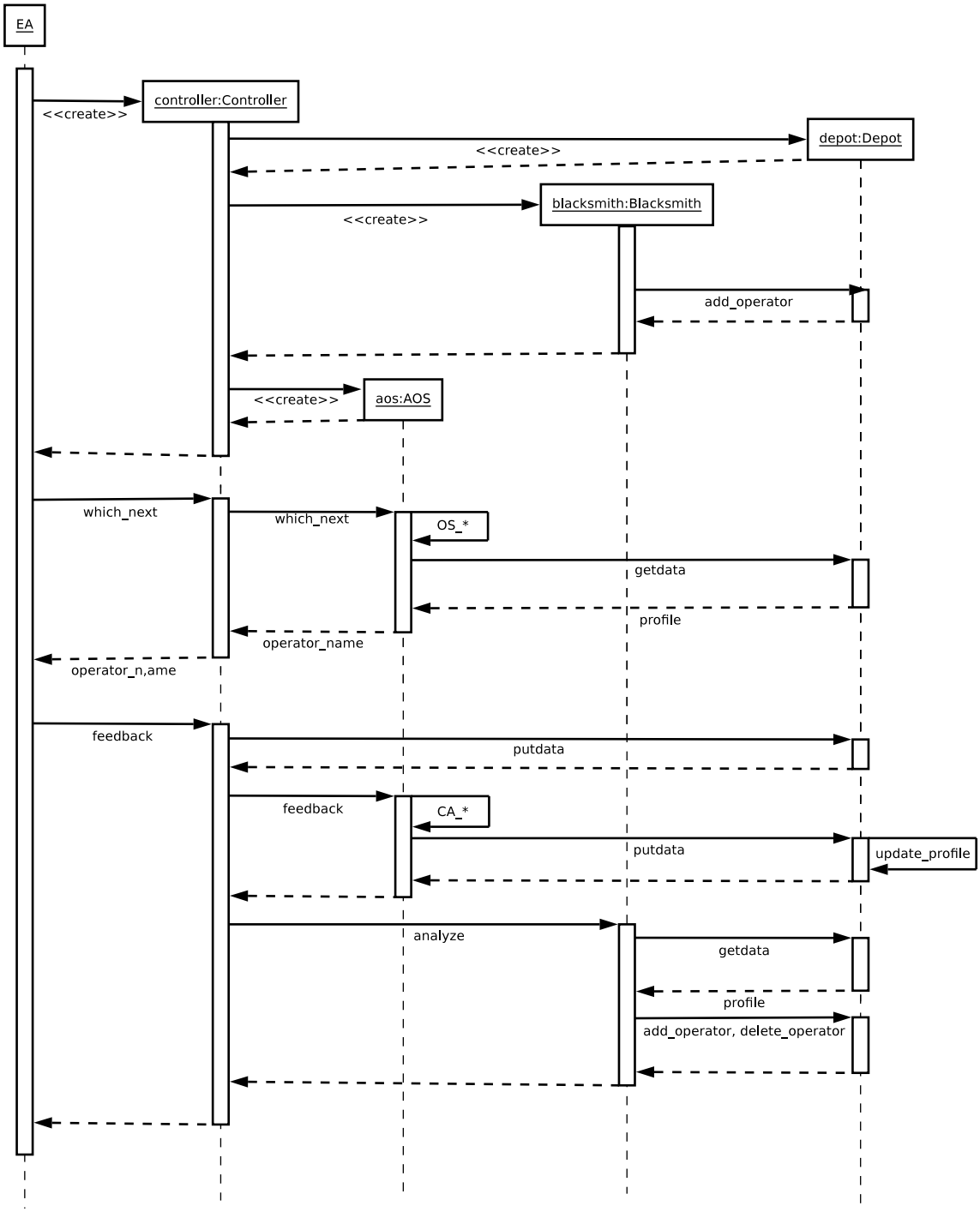

Fig. A.2 – Diagramme de séquences

Liste des figures

Listes des tables

Références bibliographiques

[Aarts and Korst, 1989] cité page 18, 26
E. Aarts and J. Korst. *Simulated Annealing and Boltzman Machines*. John Wiley, 1989.

[Alexouda, 2005] cité page 21
G. Alexouda. A user-friendly marketing decision support system for the product line design using evolutionary algorithms. *Decision Support Systems*, 38 :495–509, 2005.

[Angeline, 1995] cité page 31
P.J. Angeline. Adaptive and self-adaptive evolutionary computations. In *Computational Intelligence : A Dynamic Systems Perspective*, pages 152–163. IEEE Press, 1995.

[Auer *et al.*, 2002] cité page 78
P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2/3) :235–256, 2002.

[Baeck *et al.*, 1997] cité page 21
T. Baeck, D.B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Taylor and Francis, 1997.

[Baker, 1985] cité page 20
J.E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111, 1985.

[Barbosa and Sá, 2000] cité page 70
H. J. C. Barbosa and A. M. Sá. On adaptive operator probabilities in real coded genetic algorithms. In *XX Intl. Conf. of the Chilean Computer Science Society*, 2000.

[Battiti and Brunato, 2008] cité page 2
R. Battiti and M. Brunato. *Learning and Intelligent Optimization. Proc. Int. Conf., LION 2007*, volume 5313 of *LNCS*. Springer, 2008.

[Battiti *et al.*, 2008] cité page 2
R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*, volume 45 of *Operations Research/Computer Science Interfaces*. Springer Verlag, 2008.

[Bentley, 1999] cité page 21
P.J. Bentley, editor. *Evolutionary Design by Computers*. Morgan Kaufmann, 1999.

[Berghman *et al.*, 2009] cité page 21
L. Berghman, D. Goossens, and R. Leus. Efficient solutions for mastermind using genetic algorithms. *Computers & Operations Research*, 36 :1880–1885, 2009.

[Beyer and Schwefel, 2002] cité page 35
H. G. Beyer and H. P. Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing, Springer*, 1(1) :3–52, 2002.

[Biere *et al.*, 2003] cité page 96
A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58 :118–149, 2003.

[Birattari, 2004] cité page 35, 82, 82
M. Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2004.

[Blum and Merkle, 2008] cité page 18
C. Blum and D. Merkle, editors. *Swarm Intelligence, Introduction and Applications*. Springer, 2008.

[Borisovsky *et al.*, 2009] cité page 21
P. Borisovsky, A. Dolgui, and A. Eremeev. Genetic algorithms for a supply management problem : Mip-recombination vs greedy decoder. *European Journal of Operational Research*, 195 :770–779, 3 2009.

[Buckley, 1993] cité page 34
J. J. Buckley. Sugeno type controllers are universal controllers. *Fuzzy Sets and Systems, Elsevier*, 53(3) :299–303, 1993.

[Burkard *et al.*, 1997] cité page 53
R. E. Burkard, S. E. Karisch, and F. Rendl. QAPLIB - a quadratic assignment problem library. *Journal of Global Optimization, Springer*, 10 :391–403, 1997.

[Burke *et al.*, 2003] cité page 31
E.K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. *Handbook of Meta-heuristics*, chapter Hyper-heuristics : An Emerging Direction in Modern Search Technology, pages 457–474. Kluwer, 2003.

[Cantú-Paz, 1997] cité page 19
E. Cantú-Paz. A survey of parallel genetic algorithms. 97003, Illinois Genetic Algorithms Laboratory, 1997.

[Chen and Chang, 2009] cité page 21
Y-H Chen and F-J Chang. Evolutionary artificial neural networks for hydrological systems forecasting. *Journal of Hydrology*, 2009.

[Chiu, 1997] cité page 34
S. Chiu. *Extracting Fuzzy Rules from Data for Function Approximation and Pattern Classification*, chapter 9. John Wiley & Sons, 1997.

[Chung *et al.*, 2009] cité page 21
J-W Chung, S-M Oh, and I-C Choi. A hybrid genetic algorithm for train sequencing in the korean railway. *Omega*, 37 :555–565, 2009.

[Clark *et al.*, 2000] cité page 21
D.E. Clark, R. Mannhold, H. Kubinyi, and H. Timmerman, editors. *Evolutionary Algorithms in Molecular Design*. WILEY-VCH, 2000.

[Coley, 1999] cité page 21
D.A. Coley. *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific, 1999.

[Cook, 1971] cité page 11, 23, 72
S.A. Cook. The complexity of theorem-proving procedures. In *STOC '71 : Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, USA, 1971. ACM.

[Cordon *et al.*, 2004] cité page 33
O. Cordon, F. Gomide, F. Herrera, F. Hoffmann, and L. Magdalena. Ten years of genetic fuzzy systems : Current framework and new trends. *Fuzzy Sets and Systems*, 141(1) :5–31, 2004.

[Costa-Branco and Dente, 1999] cité page 34
P. J. Costa-Branco and J. A. Dente. Noise effects in fuzzy modelling systems. *Computational Intelligence and Applications, World Scientific and Engineering Society Press*, pages 103–108, 1999.

[Costa-Branco and Dente, 2001] cité page 34
P.J. Costa-Branco and J.A. Dente. Fuzzy systems modeling in practice. *Fuzzy Sets Syst.*, 121(1) :73–93, 2001.

[Da Costa *et al.*, 2008] cité page 78, 79, 79
L. Da Costa, Á. Fialho, M. Schoenauer, and M. Sebag. Adaptive operator selection with dynamic multi-armed bandits. In M. Keijzer et al., editor, *Proc. GECCO'08*, pages 913–920. ACM Press, 2008.

[Davis, 1985] cité page 16
L. Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 162–164, 1985.

[Davis, 1989] cité page 31, 32, 68, 70, 70
L. Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 61–69, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[De Jong,] cité page 9
K.A. De Jong. *Evolutionary computation : a unified approach*. MIT Press.

[De Jong and Spears, 1989] cité page 93
K.A. De Jong and W.M. Spears. Using genetic algorithm to solve NP-complete problems. In *Proc. of the 3rd International Conference on Genetic Algorithms (ICGA'89)*, pages 124–132, Virginia,USA, 1989.

[De Jong, 1975] cité page 35
K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, 1975.

[De Jong, 2007] cité page 19, 27
K.A. De Jong. *Parameter Setting in Evolutionary Algorithms*, chapter Parameter Setting in EAs : a 30 Year Perspective, pages 1–18. Volume 54 of Lobo et al. [2007], 2007.

[Eén and Sörensson, 2004] cité page 73
N. Eén and N. Sörensson. An extensible sat-solver. *Theory and Applications of Satisfiability Testing*, LNCS 2919 :333–336, 2004.

[Eiben and Schippers, 1998] cité page 21
A.E. Eiben and C.A. Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35 :356–50, 1998.

[Eiben and Smith, 2003] cité page 9
A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003.

[Eiben *et al.*, 1999] cité page 29, 30
A.E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3 :124–141, 1999.

[Eiben *et al.*, 2000] cité page 30
A.E. Eiben, B. Jansen, Z. Michalewicz, and B. Paechter. Solving csps using self-adaptive constraint weights : how to prevent eas from cheating. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 128–134, 2000.

[Eiben *et al.*, 2004] cité page 32
A.E. Eiben, E. Marchiori, and V.A. Valkó. Evolutionary algorithms with on-the-fly population size adjustment. In *Proceedings of Parallel Problem Solving from Nature (PPSN)*, volume 3242 of *LNCS*, pages 41–50. Springer, 2004.

[Eiben *et al.*, 2006] cité page 32
A.E. Eiben, M. Horvath, W. Kowalczyk, and M.C. Schut. Reinforcement learning for online control of evolutionary algorithms. In *Engineering Self-Organising Systems, 4th International Workshop*, volume 4335 of *LNCS*, pages 151–160. Springer, 2006.

[Eiben *et al.*, 2007] cité page 29
A.E. Eiben, Z. Michalewicz, M. Schoenauer, and J.E. Smith. *Parameter Setting in Evolutionary Algorithms*, chapter Parameter Control in Evolutionary Algorithms, pages 19–46. Volume 54 of Lobo et al. [2007], 2007.

[Elaoud *et al.*, 2007] cité page 21
S. Elaoud, J. Teghem, and B. Bouaziz. Genetic algorithms to solve the cover printing problem. *Computers & Operations Research*, 34 :3346–3361, 2007.

[Elferchichi *et al.*, 2009] cité page 21
A. Elferchichi, O. Gharsallah, I. Nouiri, F. Lebdi, and N. Lamaddalena. The genetic algorithm approach for identifying the optimal operation of a multi-reservoirs on-demand irrigation system. *Biosystems Engineering*, 2009.

[Fialho *et al.*, 2008] cité page 68, 70, 77, 78, 78, 79
Á. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Extreme value based adaptive operator selection. In G. Rudolph et al., editor, *Proc. PPSN'08*, pages 175–184. Springer, 2008.

[Fialho *et al.*, 2009] cité page 79
Á. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Dynamic multi-armed ban-

dits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In *Proc. LION'09*, 2009.

[Fleurent and Ferland, 1996] cité page 93, 94
C. Fleurent and J.A. Ferland. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In *Cliques, Coloring, and Satisfiability : Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 619–652, 1996.

[Fogel *et al.*, 1996] cité page 9
L.J. Fogel, A.J. Owens, and M.J.Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley., 1996.

[Gen and Cheng, 1997] cité page 21
M. Gen and R. Cheng. *Genetic Algorithms and Engineering Design*. John Willey & Sons, 1997.

[Glover, 1989] cité page 18
F. Glover. Tabu search – part i. *ORSA Journal of Computing*, 1(3) :190–206, 1989.

[Glover, 1990] cité page 18
F. Glover. Tabu search – part ii. *ORSA Journal of Computing*, 2(1) :4–32, 1990.

[Goldberg and Lingle, 1985] cité page 16
D.E. Goldberg and R. Lingle. Alleles, loci, and the tsp. In *Proceedings of First International Conference on Genetic Algorithms*, pages 154–159, 1985.

[Goldberg, 1990a] cité page 70
D.E. Goldberg. Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding. *Machine Learning*, 5(4) :407–426, 1990.

[Goldberg, 1990b] cité page 12
D.E. Goldberg. Real-coded genetic algorithms, virtual alphabets and blocking. Technical Report 90001, University of Illinois at Urbana-Champaign, 1990.

[Gottlieb and Voss, 2000] cité page 93
J. Gottlieb and N. Voss. Adaptive fitness functions for the satisfiability problem. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo Hans-Paul Schwefel, editor, *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Paris, France, September 16-20 2000. Springer Verlag. LNCS 1917.

[Hamadi *et al.*, 2008] cité page 2
Y. Hamadi, E. Monfroy, and F. Saubion. Special issue on autonomous search. *Constraint Programming Letters*, 4, 2008.

[Harik and Lobo, 1999] cité page 35
G.R. Harik and F.G. Lobo. A parameter-less GA. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 258–265, 1999.

[Hartland *et al.*, 2006] cité page 78
C. Hartland, G. Sylvain, N. Baskiotis, O. Teytaud, and M. Sebag. Multi-armed bandit, dynamic environments and meta-bandits. In *Online Trading of Exploration and Exploitation Workshop, NIPS*, 2006.

[Haupt and Haupt, 1998] cité page 21
R.L. Haupt and S.E. Haupt. *Practical Genetic Algorithms*. John Willey & Sons, 1998.

[Herrera and Lozano, 1996] cité page 33
F. Herrera and M. Lozano. Adaptation of genetic algorithm parameters based on fuzzy logic controllers. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 95–125. Physica-Verlag, 1996.

[Herrera and Lozano, 2003] cité page 34
F. Herrera and M. Lozano. Fuzzy adaptive genetic algorithms : Design, taxonomy and future directions. *Soft Computing*, 7(8) :545–562, 2003.

[Holland, 1975] cité page 9, 20
J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[Hoos and Stützle, 2000] cité page 72, 73
H.H. Hoos and T. Stützle. *SATLIB : An Online Resource for Research on SAT*, pages 283–292. IOS Press, www.satlib.org, 2000.

[Hurley *et al.*, 1998] cité page 21
S. Hurley, L. Moutinho, and S.F. Witt. Genetic algorithms for tourism marketing. *Annals of Tourism Research*, 25 :498–514, 1998.

[Igel and Kreutz, 2001] cité page 32
C. Igel and M. Kreutz. Operator adaptation in structure optimization of neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, page 1094. Morgan Kaufmann, 2001.

[Jawahar and Balaji, 2009] cité page 21
N. Jawahar and A.N. Balaji. A genetic algorithm for the two-stage supply chain distribution problem associated with a fixed charge. *European Journal of Operational Research*, 194 :496–537, 2009.

[Julstrom, 1995] cité page 32, 70
B.A Julstrom. What have you done for me latey ? adapting operator probabilities in a steady-state genetic algorithm. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 81–87. Morgan Kaufmann, 1995.

[Kee *et al.*, 2001] cité page 32, 47
E. Kee, S. Airey, and W. Cyre. An adaptive genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 391–397. Morgan Kaufmann, 2001.

[Kim *et al.*, 2007] cité page 21
D-K Kim, H. Cao, K-S Jeong anf F. Recknagel, and G-J Joo. Predictive function and rules for population dynamics of microcystis aeruginosa in the regulated nakdong river (south korea), discovered by evolutionary algorithms. *Ecological Modelling*, 203 :147–156, 2007.

[Kim *et al.*, 2008] cité page 21
J.S. Kim, J-W Lee, Y-K Noh, J-Y Park, D-Y Lee, K-A Yang, Y.G. Chai, J.C. Kim, and

B-T Zhang. An evolutionary monte carlo algorithm for predicting dna hybridization. *Biosystems*, 91 :69–75, 2008.

[Koopmans and Beckmann, 1957] cité page 23
Tj. C. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25 :53–76, 1957.

[Koza, 1992] cité page 14
J.R. Koza. *Genetic Programming.* MIT Press, 1992.

[Kulkarni, 2001] cité page 33, 33
A. Kulkarni. *Fuzzy Logic Fundamentals*, chapter 3, pages 61–103. Prentice Hall PTR, 2001.

[Lardeux *et al.*, 2006] cité page 72, 93, 94, 94, 95, 99
F. Lardeux, F. Saubion, and Jin-Kao Hao. GASAT : A genetic local search algorithm for the satisfiability problem. *Evolutionary Computation*, 14(2) :223–253, 2006.

[Lee and Takagi, 1993] cité page 33
M.A. Lee and H. Takagi. Dynamic control of genetic algorithms using fuzzy logic techniques. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 76–83, San Mateo, CA, 1993. Morgan Kaufmann.

[Levitin, 2006] cité page 21
G. Levitin. Genetic algorithms in reliability engineering. *Reliability Engineering & System Safety*, 91 :975–976, 2006.

[Lima and Lobo, 2004] cité page 35
C. Lima and F. Lobo. Parameter-less optimization with the extended compact genetic algorithm and iterated local search. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2004.

[Lis, 1996] cité page 32
J. Lis. Parallel genetic algorithm with dynamic control parameter. In *Proc. of IEEE Intl. Conference on Evolutionary Computation (CEC)*, pages 324–329, 1996.

[Liu and Lampinen, 2005] cité page 33
J. Liu and J. Lampinen. A differential evolution based incremental training method for rbf networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 881–888. ACM Press, 2005.

[Liu *et al.*, 2008] cité page 21
S. Liu, W. Huang, and H. Ma. An effective genetic algorithm for the fleet size and mix vehicle routing problems. *Transportation Research Part E : Logistics and Transportation Review*, 2008.

[Lobo and Goldberg, 1997] cité page 32, 70, 70, 70
F.G. Lobo and D.E. Goldberg. Decision making in a hybrid genetic algorithm. In T. Bäck et al., editor, *Proc. of IEEE Intl. Conference on Evolutionary Computation (CEC)*, pages 121–125. IEEE Press, 1997.

[Lobo *et al.*, 2007] cité page 2, 26, 123, 124
F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[Loonen *et al.*, 2006] cité page 21
W. Loonen, P.S.C. Heuberger, A.H. Bakema, and P. Schot. Application of a genetic algorithm to minimize agricultural nitrogen deposition in nature reserves. *Agricultural Systems*, 88 :360–375, 2006.

[Mahfoud, 1995] cité page 19
S.W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 1995.

[Maturana and Riff, 2007] cité page 14
J Maturana and M.C. Riff. Solving the short-term electrical generation scheduling problem by an adaptive evolutionary approach. *European Journal of Operational Research*, 179 :677–691, 2007.

[Maturana and Saubion, 2007a] cité page 44, 104
J. Maturana and F. Saubion. On the design of adaptive control strategies for evolutionary algorithms. In *Proceedings of 8th International Conference on Artificial Evolution. LNCS 4926, Springer*, 2007.

[Maturana and Saubion, 2007b] cité page 44
J. Maturana and F. Saubion. Towards a generic control strategy for EAs : an adaptive fuzzy-learning approach. In *Proceedings of IEEE International Conference on Evolutionary Computation (CEC)*, pages 4546–4553, 2007.

[Maturana and Saubion, 2008a] cité page 68
J. Maturana and F. Saubion. A compass to guide genetic algorithms. In *Proceedings of the 1Oth International Conference on Parallel Problem Solving from Nature (PPSN X)*, number 5199 in Lecture Notes in Computer Science, pages 256–265. Springer, 2008.

[Maturana and Saubion, 2008b] cité page 44
J. Maturana and F. Saubion. From parameter control to search control : Parameter control abstraction in evolutionary algorithms. *Constraint Programming Letters*, 4, Special Issue on Autonomous Search :39–65, 2008.

[Maturana *et al.*, 2009a] cité page 68
J. Maturana, Á. Fialho, F. Saubion, M. Schoenauer, and M. Sebag. Compass and dynamic multi-armed bandits for adaptive operator selection. In *Conference on Evolutionary Computation (CEC) (Accepted)*. IEEE Press, 2009.

[Maturana *et al.*, 2009b] cité page 88
J. Maturana, F. Lardeux, and F. Saubion. Génération et contrôle autonomes d'opérateurs pour les algorithmes évolutionnaires. In Y. Deville, editor, *Actes des Cinquièmes Journées Francophones de Programmation par Contraintes*, pages 185–194, 2009.

[Mendoza *et al.*, 2009] cité page 21
J.E. Mendoza, A.L. Medaglia, and N. Velasco. An evolutionary-based decision support system for vehicle routing : The case of a public utility. *Decision Support Systems*, 46 :730–742, 2009.

[Michalewicz and Fogel, 1998] cité page 21
Z. Michalewicz and D.B. Fogel. *How to Solve It : Modern Heuristics*. Springer, 2nd edition, 1998.

[Michalewicz, 1996] cité page 12
Z. Michalewicz. *Genetics Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.

[Miller and Goldberg, 1995] cité page 20
B.L. Miller and D.E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. Technical Report 95006, Illinois Genetic Algorithms Laboratory, 1995.

[Mohebbi *et al.*, 2008] cité page 21
M. Mohebbi, J. Barouei, M.R. Akbarzadeh-T, A.R. Rowhanimanesh, M.B. Habibi-Najafi, and M. Yavarmanesh. Modeling and optimization of viscosity in enzyme-modified cheese by fuzzy logic and genetic algorithm. *Computers and Electronics in Agriculture*, 62 :260–265, 2008.

[Moscato *et al.*, 2007] cité page 21
P. Moscato, A. Mendes, and R. Berretta. Benchmarking a memetic algorithm for ordering microarray data. *Biosystems*, 88 :56–75, 2007.

[Moscato, 1989] cité page 9
P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms. Technical Report 826, Caltech Concurrent Computation Program, 1989.

[Mühlenbein and Paaß, 1996] cité page 9
H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions i. binary parameters. In *Proceedings of Parallel Problem Solving from Nature (PPSN)*, pages 178–187, 1996.

[Nannen and Eiben, 2007] cité page 32, 35
V. Nannen and A.E. Eiben. Relevance estimation and value calibration of EA parameters. In *Proc. of 20th Int.Joint Conf. on Artificial Intelligence IJCAI-07*, pages 975–980, 2007.

[Nozaki *et al.*, 1997] cité page 34
K. Nozaki, H. Ishibuchi, and H. Tanaka. A simple but powerful heuristic method for generating fuzzy rules from numerical data. *Fuzzy Sets Syst.*, 86(3) :251–270, 1997.

[Oliver *et al.*, 1987] cité page 16, 53
I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the TSP. In *Proceedings of the 2nd Int.Conf.on Genetic Algorithms and their applications, Lawrence Erlbaum Associates*, USA, 1987.

[Page, 1954] cité page 78
E.S. Page. Continuous inspection schemes. *Biometrika*, 41 :100–115, 1954.

[Papadimitriou, 1994] cité page 22
C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994.

[Pareto, 1896] cité page 18
V. Pareto. *Cours d'économie politique*. Université de Lusanne, 1896.

[Passino and Yurkovich, 1998] cité page 33
K.M. Passino and S. Yurkovich. *Fuzzy Control*. Addison Wesley Longman, 1998.

[Petrovic and Epstein, 2006] cité page 33
S. Petrovic and S. Epstein. Relative support weight learning for constraint solving. In *Workshop on Learning for Search, AAAI-06*, pages 115–122, 2006.

[Piegat, 2001] cité page 33, 33, 34, 62
A. Piegat. *Fuzzy Modeling and Control*. Springer-Verlag, 2001.

[Poli *et al.*, 2008] cité page 21
R. Poli, W.B. Langdon, and N. Freitag McPhee. *A Field Guide to Genetic Programming*. 2008.

[Rechenberg, 1973] cité page 9, 18, 30, 31
I. Rechenberg. *Evolutionsstrategie – Optimierung technisher Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, GER, 1973.

[Reynolds, 1999] cité page 9
R.G. Reynolds. *An overview of Cultural Algorithms*. Advances in Evolutionart Computation. McGraw Hill, 1999.

[Rich and Knight, 1991] cité page 3
E. Rich and K. Knight. *Artificial Intelligence*. McGraw-Hill College, 1991.

[Riid and Rüstern, 2004] cité page 34, 62
A. Riid and E. Rüstern. Heuro-fuzzy extraction of interpretable fuzzy rules from data. *Proc.IEEE International Conference on Systems, Man And Cybernetics*, 3 :2266–2271, 2004.

[Robet *et al.*, 2009] cité page 86
J. Robet, F. Lardeux, and F. Saubion. Une approche de contrôle autonome pour la recherche locale. In Y. Deville, editor, *Actes des Cinquièmes Journées Francophones de Programmation par Contraintes*, pages 205–214, 2009.

[Rodríguez-Tello, 2007] cité page 17
E. Rodríguez-Tello. *Nouvelles Fonctions d'Évaluation pour les Problèmes d'Étiquetage de Graphes BMP et MinLA*. PhD thesis, Université d'Angers, 2007.

[Rom and Slotnick, 2009] cité page 21
W.O. Rom and S.A. Slotnick. Order acceptance using genetic algorithms. *Computers & Operations Research*, 36 :1758–1767, 2009.

[Rossi *et al.*, 2000] cité page 93
C. Rossi, E. Marchiori, and J.N. Kok. An adaptive evolutionary algorithm for the satisfiability problem. In *Proc. of the ACM Symposium on Applied Computing (SAC '00)*, pages 463–470. ACM press, 2000.

[Sahni and González, 1976] cité page 23
S. Sahni and T. González. P-complete approximation problems. *J. ACM*, 23(3) :555–565, 1976.

[Setnes *et al.*, 1998] cité page 34
M. Setnes, R. Babuska, and H.B. Verbruggen. Rule-based modeling precision and transparency. *IEEE Transactions on Systems, Man, and Cybernetics–Part C : Applications and Reviews*, 28(1) :165–169, 1998.

[Setzkorn *et al.*, 2007] cité page 21
C. Setzkorn, A.F.G. Taktak, and B.E. Damato. On the use of multi-objective evolutionary algorithms for survival analysis. *Biosystems*, 87 :31–48, 2007.

[Shao *et al.*, 2009] cité page 21
X. Shao, X. Li, L. Gao, and C. Zha. Integration of process planning and scheduling–a modified genetic algorithm-based approach. *Computers & Operations Research*, 36 :2082–2096, 2009.

[Silva *et al.*, 2009] cité page 21
T.P.C. Silva, E. Silva de Moura, J.M.B. Cavalcanti, A.S. da Silva, M. Gomes de Carvalho, and M.A. Gonçalves. An evolutionary approach for combining different sources of evidence in search engines. *Information Systems*, 34 :276–289, 2009.

[Smith and Timmis, 2008] cité page 21
S.L. Smith and J. Timmis. An immune network inspired evolutionary algorithm for the diagnosis of parkinson's disease. *Biosystems*, 94 :34–46, 2008.

[Soak *et al.*, 2008] cité page 21
S-M Soak, S-W Lee, G-T Yeo, and M-G Jeon. An effective evolutionary algorithm for the multiple container packing problem. *Progress in Natural Science*, 18 :337–344, 2008.

[Soleimani *et al.*, 2009] cité page 21
H. Soleimani, H.R. Golmakani, and M.H. Salimi. Markowitz-based portfolio selection with minimum transaction lots, cardinality constraints and regarding sector capitalization using genetic algorithm. *Expert Systems with Applications*, 36 :5058–5063, 2009.

[Storn and Price, 1995] cité page 9
R. Storn and K. Price. Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, University of Califirnia Berkeley, 1995.

[Subbu and Bonissone, 2003] cité page 33
R. Subbu and P. Bonissone. A retrospective view of fuzzy control of evolutionary algorithm resources. *Proc. FUZZ-IEEE*, pages 143–148, 2003.

[Takagi and Sugeno, 1985] cité page 34
T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 15 :116–132, 1985.

[Terfloth and Gasteiger, 2001] cité page 21
L. Terfloth and J. Gasteiger. Neural networks and genetic algorithms in drug design. *Drug Discovery Today*, 6 :102–108, 2001.

[Thierens, 2005] cité page 32, 70, 77
D. Thierens. An adaptive pursuit strategy for allocating operator probabilities. In

Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pages 1539–1546, 2005.

[Thierens, 2007] cité page 32, 68, 70, 73, 74
D. Thierens. *Parameter Setting in Evolutionary Algorithms*, chapter Adaptive Strategies for Operator Allocation, pages 77–90. Volume 54 of Lobo et al. [2007], 2007.

[Tsutsui *et al.*, 1997] cité page 19
S. Tsutsui, Y. Fujimoto, and A. Ghosh. Forking GAs : GAs with search space division schemes. *Evolutionary Computation*, 5(1) :61–80, 1997.

[Tuson and Ross, 1998] cité page 70, 70
A. Tuson and P. Ross. Adapting operator settings in genetic algorithms. *Evolutionary Computation*, 6(2) :161–184, 1998.

[Ursem, 2002] cité page 31, 35
R.K. Ursem. Diversity-guided evolutionary algorithms. In *Proceedings of Parallel Problem Solving from Nature (PPSN)*, volume 2439 of *LNCS*, pages 462–474. Springer, 2002.

[Wang and Mendel, 1992] cité page 34
L. X. Wang and J. M. Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man and Cybernetics*, 22(6) :1414–1427, 1992.

[Weinberger, 1990] cité page 11
E. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63 :325–336, 1990.

[Whitacre *et al.*, 2006] cité page 33, 70, 78
J. Whitacre, T. Pham, and R. Sarker. Use of statistical outlier detection method in adaptive evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1345–1352. ACM, 2006.

[Wineberg and Chen, 2004] cité page 19
M. Wineberg and J. Chen. The shifting balance genetic algorithm as more than just another island model GA. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 318–329, 2004.

[Wolpert and Macready, 1997] cité page 36
D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, IEEE Press*, 1 :67–82, 1997.

[Wong *et al.*, 2003] cité page 32, 68, 73, 74
Wong, Lee, Leung, and Ho. A novel approach in parameter adaptation and diversity maintenance for GAs. *Soft Computing*, 7(8) :506–515, 2003.

[Wu, 1999] cité page 21
D.J. Wu. Discovering near-optimal pricing strategies for the deregulated electric power marketplace using genetic algorithms. *Decision Support Systems*, 27 :25–45, 1999.

[Xu *et al.*, 2008] cité page 31
L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown. Satzilla : Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32 :565–606, 2008.

[Yuan and Gallagher, 2004] cité page 82
Bo Yuan and M. Gallagher. Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In Xin Yao et al., editor, *Proc. PPSN'04*, pages 172–181. Springer Verlag, 2004.

[Yuan and Gallagher, 2007] cité page 35, 61
B. Yuan and M. Gallagher. *Parameter Setting in Evolutionary Algorithms*, chapter Combining Meta-EAs and racing for Difficult, pages 121–142. Volume 54 of Lobo et al. [2007], 2007.

[Zecchina *et al.*, 1999] cité page 96
R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400 :133–137, 1999.

[Zhou *et al.*, 2009] cité page 21
H. Zhou, W. Cheung, and L.C. Leun. Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. *European Journal of Operational Research*, 194 :637–649, 2009.

[Zio *et al.*, 2009] cité page 21
E. Zio, P. Baraldi, and N. Pedroni. Optimal power system generation scheduling by multi-objective genetic algorithms with preferences. *Reliability Engineering & System Safety*, 94 :432–444, 2009.

Contrôle Générique de Paramètres pour les Algorithmes Évolutionnaires

Résumé

Les paramètres des Algorithmes Évolutionnaires (AEs) ont une forte influence sur leur capacité à produire de bons résultats. Ces paramètres définissent les aspects structuraux et comportementaux de l'AE, comme la définition des composants qui seront inclus (e.g., l'ensemble d'opérateurs à utiliser, le codage, le schéma de sélection) ou la façon dont ces composants seront employés (e.g., le taux d'application des opérateurs).
La paramétrisation des AEs a été longtemps un domaine de spécialistes, et même si de nombreuses études ont abordé le problème de la paramétrisation, il existe un déficit d'approches génériques qui puissent être appliquées à une grande variété d'AEs d'une manière simple.
Cette thèse traite le problème de la conception d'un contrôleur générique, qui puisse être inclus dans n'importe quel AE avec un minimum d'efforts. Ceci est accompli en incorporant un composant d'apprentissage adaptatif, qui surveille l'état de la recherche et modifie les valeurs des paramètres. Le contrôleur se focalise sur les objectifs communs à tout EA, i.e., la maximisation de la diversité de la population et de la qualité des individus, afin de maintenir un équilibre convenable entre l'exploration et l'exploitation.
Des nombreuses configurations des mécanismes d'apprentissage et d'ajustement ont été essayées et analysées, en utilisant AEs différents qui résolvent des problèmes combinatoires connus. Les bons résultats obtenus suggèrent que notre objectif de construire un contrôleur générique et facile à utiliser constitue une approche encourageante.

Mots-clés : Contrôle de paramètres, recherche autonome, abstraction d'implementation.

Generic Parameter Control for Evolutionary Algorithms

Abstract

Parameters of Evolutionary Algorithms (EAs) greatly influence their ability to produce good results. These parameters define structural and behavioral aspects of the EA, ranging from choosing which features will be included (e.g. set of operators, encoding, selection scheme) to how these features will be used (e.g. operators' application rates).
Parametrization of EAs have been longly a specialist domain, and even though many previous studies have addressed the parametrization problem, there is a lack of generic approaches that could be applied to a wide range of EAs in a simple way.
This thesis deals with the problem of creating a generic controller, that could be included in any EA with a minimum effort. Generality is accomplished by incorporating a learning/adaptive component, that monitors the state of the search and modify parameter values. The controller focuses in common goals of all EAs, that is to say, to maximize both the diversity of the population and the quality of the individuals, in order to maintain a convenient balance between exploration and exploitation.
Several configurations of learning tools and adjustment methods were tried and analyzed, using different EAs solving well known combinatorial problems. Positive results suggest that our goal of building a generic, easy-to-use controller is a feasible approach.

Keywords : Parameter control, autonomous search, implementation abstraction.