

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (INP Toulouse)

Discipline ou spécialité :

Intelligence Artificielle

Présentée et soutenue par :

M. CHARLIE VANARET

le mardi 27 janvier 2015

Titre :

HYBRIDATION D'ALGORITHMES EVOLUTIONNAIRES ET DE METHODES D'INTERVALLES POUR L'OPTIMISATION DE PROBLEMES DIFFICILES

Ecole doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Institut de Recherche en Informatique de Toulouse (I.R.I.T.)

Directeur(s) de Thèse :

M. NICOLAS DURAND

M. JEAN-BAPTISTE GOTTELAND

Rapporteurs :

M. EL-GHAZALI TALBI, UNIVERSITE LILLE 1

M. GILLES TROMBETTONI, UNIVERSITE MONTPELLIER 2

Membre(s) du jury :

M. MARC SCHOENAUER, UNIVERSITE PARIS 11, Président

M. JEAN-BAPTISTE GOTTELAND, ECOLE NATIONALE DE L'AVIATION CIVILE, Membre

M. JEAN MARC ALLIOT, ECOLE NATIONALE DE L'AVIATION CIVILE, Membre

M. JIN-KAO HAO, UNIVERSITE D'ANGERS, Membre

M. NICOLAS DURAND, ECOLE NATIONALE DE L'AVIATION CIVILE, Membre

M. THOMAS SCHIEX, INRA TOULOUSE, Membre

Résumé

L'optimisation globale fiable est dédiée à la recherche d'un minimum global en présence d'erreurs d'arrondis. Les seules approches fournissant une preuve numérique d'optimalité sont des méthodes d'intervalles qui partitionnent l'espace de recherche et éliminent les sous-espaces qui ne peuvent contenir de solution optimale. Ces méthodes exhaustives, appelées branch and bound par intervalles, sont étudiées depuis les années 60 et ont récemment intégré des techniques de réfutation et de contraction, issues des communautés d'analyse par intervalles et de programmation par contraintes. Il est d'une importance cruciale de calculer i) un encadrement précis de la fonction objectif et des contraintes sur un sous-domaine ; ii) une bonne approximation (un majorant) du minimum global.

Les solveurs de pointe sont généralement des méthodes *intégratives* : ils invoquent sur chaque sous-domaine des algorithmes d'optimisation locale afin d'obtenir une bonne approximation du minimum global. Dans ce document, nous nous intéressons à un cadre *coopératif* combinant des méthodes d'intervalles et des algorithmes évolutionnaires. Ces derniers sont des algorithmes stochastiques faisant évoluer une population de solutions candidates (individus) dans l'espace de recherche de manière itérative, dans l'espoir de converger vers des solutions satisfaisantes. Les algorithmes évolutionnaires, dotés de mécanismes permettant de s'échapper des minima locaux, sont particulièrement adaptés à la résolution de problèmes difficiles pour lesquels les méthodes traditionnelles peinent à converger.

Au sein de notre solveur coopératif Charibde, l'algorithme évolutionnaire et l'algorithme sur intervalles exécutés en parallèle échangent bornes, solutions et espace de recherche par passage de messages. Une stratégie couplant une heuristique d'exploration géométrique et un opérateur de réduction de domaine empêche la convergence prématurée de la population vers des minima locaux et évite à l'algorithme évolutionnaire d'explorer des sous-espaces sous-optimaux ou non réalisables. Une comparaison de Charibde avec des solveurs de pointe (GlobSol, IBBA, Ibex) sur une base de problèmes difficiles montre un gain de temps d'un ordre de grandeur. De nouveaux résultats optimaux sont fournis pour cinq problèmes multimodaux pour lesquels peu de solutions, même approchées, sont connues dans la littérature. Nous proposons une application aéronautique dans laquelle la résolution de conflits est modélisée par un problème d'optimisation sous contraintes universellement quantifiées, et résolue par des techniques d'intervalles spécifiques. Enfin, nous certifions l'optimalité de la meilleure solution connue pour le cluster de Lennard-Jones à cinq atomes, un problème ouvert en dynamique moléculaire.

Abstract

Reliable global optimization is dedicated to finding a global minimum in the presence of rounding errors. The only approaches for achieving a numerical proof of optimality in global optimization are interval-based methods that interleave branching of the search-space and pruning of the subdomains that cannot contain an optimal solution. The exhaustive interval

branch and bound methods have been widely studied since the 1960s and have benefitted from the development of refutation methods and filtering algorithms, stemming from the interval analysis and interval constraint programming communities. It is of the utmost importance: i) to compute sharp enclosures of the objective function and the constraints on a given subdomain; ii) to find a good approximation (an upper bound) of the global minimum.

State-of-the-art solvers are generally *integrative* methods, that is they embed local optimization algorithms to compute a good upper bound of the global minimum over each subspace. In this document, we propose a *cooperative* framework in which interval methods cooperate with evolutionary algorithms. The latter are stochastic algorithms in which a population of individuals (candidate solutions) iteratively evolves in the search-space to reach satisfactory solutions. Evolutionary algorithms, endowed with operators that help individuals escape from local minima, are particularly suited for difficult problems on which traditional methods struggle to converge.

Within our cooperative solver Charibde, the evolutionary algorithm and the interval-based algorithm run in parallel and exchange bounds, solutions and search-space via message passing. A strategy combining a geometric exploration heuristic and a domain reduction operator prevents premature convergence toward local minima and prevents the evolutionary algorithm from exploring suboptimal or unfeasible subspaces. A comparison of Charibde with state-of-the-art solvers based on interval analysis (GlobSol, IBBA, Ibex) on a benchmark of difficult problems shows that Charibde converges faster by an order of magnitude. New optimality results are provided for five multimodal problems, for which few solutions were available in the literature. We present an aeronautical application in which conflict solving between aircraft is modeled by an universally quantified constrained optimization problem, and solved by specific interval contractors. Finally, we certify the optimality of the putative solution to the Lennard-Jones cluster problem for five atoms, an open problem in molecular dynamics.

Remerciements

Je dois mon entrée – somme toute assez imprévue – dans le monde académique à Jean-Marc Alliot. Mes premières semaines à la DTI, à la découverte de la programmation par contraintes, m'ont laissé entrevoir certains aspects de la recherche que je ne soupçonnais pas ; grand bien lui en a pris. Je remercie Jean-Baptiste Gotteland pour sa confiance, sa disponibilité et la grande latitude qu'il m'a laissée. Merci à Nicolas Durand pour sa générosité et ses encouragements. Je garde un souvenir particulièrement émouvant de notre tentative de record de la traversée du Massif central en TB-20.

Je tiens à remercier Gilles Trombettoni d'avoir répondu patiemment à toutes mes questions relatives à l'implémentation des contracteurs. Je suis reconnaissant à Gilles et à El-Ghazali Talbi d'avoir accepté de rapporter ma thèse. Je remercie mes examinateurs Thomas Schiex et Marc Schoenauer d'avoir fait le déplacement à Toulouse, et Jin-Kao Hao de m'avoir donné l'opportunité de faire ma première télé.

Que dire de mes « compagnons de galère » Richard Alligier et Mohammad Ghasemi Hamed... Partager un bureau à trois n'est pas toujours chose aisée, surtout lorsque mes jeux de mots du lundi matin valent ceux d'un vendredi après-midi. Z05 a été le théâtre de discours parfois animés, toujours passionnés, d'échanges constructifs et de synchronisation pour la pause thé.

Mon camarade de marave Cyril Allignol, le baryton Nicolas Barnier, David Gianazza maître ès foncteurs, Alexandre « Jean-Michel » Gondran, Sonia Cafieri, Loïc Cellier, Brunilde Girardet, Laureline Guys, Olga Rodionova, Nicolas Saporito et Estelle Malavolti – pour un temps mes voisins, entre deux valses des bureaux – tous ont contribué à la bonne humeur et à l'ambiance chaleureuse du bâtiment Z.

Je salue Daniel Ruiz pour sa gentillesse, et l'équipe APO pour leur accueil. Frédéric Messine et Jordan Ninin ont aimablement répondu à mes interrogations affines. Ma gratitude va à Christine Surly, garante de ma logistique pendant ces trois années chez Midival, et à Jean-Pierre Baritaud pour le soutien technique lors de ma soutenance.

Merci à Fabien Bourrel, mon relecteur officiel qui n'a jamais trouvé une seule typo, alors que... J'en profite pour passer un petit coucou à ma famille, aux Barousse, aux boys d'Hydra, à mes camarades de promo n7, aux expatriés viennois, au TUC Escrime et au Péry.

Merci à mes parents et mon frangin d'avoir fait le déplacement lors de ma soutenance.

Désolé pour le slide numéro 2.

Table des matières

Glossaire **xvii**

1 Optimisation continue **5**
1.1 Problèmes d'optimisation . . . 5
1.1.1 Minima locaux et globaux . . . 6
1.1.2 Existence d'un minimum . . . 7
1.1.3 Optimisation sans contraintes . . . 7
1.1.4 Optimisation sous contraintes . . . 8
1.2 Techniques de résolution . . . 10
1.2.1 Problèmes linéaires . . . 11
1.2.2 Problèmes convexes . . . 11
1.2.3 Problèmes non convexes . . . 12

2 Algorithmes à population **15**
2.1 Algorithmes à population . . . 16
2.1.1 Gestion des contraintes . . . 17
2.1.2 Algorithmes utilisés . . . 17
2.2 Algorithme génétique . . . 18
2.2.1 Sélection des parents . . . 18
2.2.2 Croisement et mutation . . . 19
2.2.3 Opérateur de scaling . . . 20
2.2.4 Opérateur de sharing . . . 21
2.2.5 Remplacement de la population . . . 21
2.3 Optimisation par essaims particulaires . . . 22
2.3.1 Formules de mise à jour . . . 22
2.3.2 Voisinage . . . 23
2.4 Algorithme à évolution différentielle . . . 23
2.4.1 Opérateur de croisement quaternaire . . . 24
2.4.2 Choix de l'individu de base . . . 24
2.4.3 Contraintes de bornes . . . 25
2.4.4 Gestion directe des contraintes . . . 25
2.5 Critères d'arrêt . . . 25

3 Analyse par intervalles **27**
3.1 Calcul par intervalles . . . 28
3.1.1 Contrôle des arrondis . . . 28
3.1.2 Arithmétique d'intervalles . . . 29
3.2 Fonctions d'inclusion . . . 31
3.2.1 Dépendance . . . 32
3.2.2 Formes du deuxième ordre . . . 33
3.2.3 Extension par monotonie . . . 36
3.2.4 Arithmétique affine . . . 37
3.3 Algorithme de branch and bound par intervalles . . . 37
3.3.1 Branch and bound . . . 38
3.3.2 Branch and bound par intervalles . . . 39
3.3.3 Heuristiques . . . 40
3.3.4 Techniques d'accélération . . . 41
3.4 Différentiation automatique . . . 43
3.4.1 Mode direct . . . 44
3.4.2 Mode adjoint . . . 44
3.5 Conclusion . . . 45

4 Contracteurs **47**
4.1 Opérateurs de cohérence partielle . . . 47
4.2 Cohérences locales . . . 49
4.2.1 Cohérence 2B . . . 49
4.2.2 Evaluation-propagation . . . 50
4.2.3 Contracteurs exploitant la monotonie . . . 51
4.2.4 Cohérence de boîte . . . 52
4.2.5 Algorithme de Newton par intervalles . . . 53
4.2.6 Point fixe . . . 55
4.3 Cohérences fortes . . . 56
4.3.1 Cohérence 3B . . . 56
4.3.2 Cohérence CID . . . 57
4.4 Cohérences globales . . . 59
4.4.1 Algorithme de Newton . . . 59
4.4.2 Convexification . . . 60
4.5 Contraction et différentiation automatique . . . 61

5 Charibde **65**
5.1 Hybridation de techniques d'optimisation . . . 66
5.1.1 Méthodes intégratives . . . 67
5.1.2 Méthodes coopératives . . . 67
5.1.3 Charibde : une approche coopérative . . . 68
5.2 Branch and contract par intervalles . . . 70
5.2.1 Contracteur pour l'optimisation sans contraintes . . . 71

5.2.2 Contracteur pour l'optimisation sous contraintes . . . 73
5.2.3 MaxDist : une heuristique de recherche géométrique . . . 76
5.3 Evolution différentielle . . . 78
5.3.1 Gestion rigoureuse de la fonction objectif . . . 79
5.3.2 Gestion rigoureuse des contraintes . . . 81
5.3.3 Evaluation par comparaison . . . 81
5.3.4 Exploitation du domaine sur intervalles . . . 84
5.4 Comparaison de solveurs sur la base COCONUT . . . 92
5.4.1 Base de comparaison . . . 92
5.4.2 Comparaison des résultats . . . 93
5.5 Nouveaux minima globaux de problèmes difficiles . . . 96
5.5.1 Fonctions de test . . . 96
5.5.2 Résultats expérimentaux . . . 98
5.5.3 Bénéfices de l'hybridation . . . 101
5.5.4 Comparaison de solveurs . . . 107
5.6 Conclusion . . . 109

6 Contributions à la résolution de conflits aériens 111
6.1 Conflits aériens . . . 112
6.1.1 Séparation . . . 112
6.1.2 Approches pour la résolution de conflits . . . 113
6.2 Approche centralisée par algorithmes évolutionnaires . . . 114
6.2.1 Modèle adopté . . . 114
6.2.2 Un problème d'optimisation sous contraintes . . . 115
6.2.3 Simulateur de trafic . . . 116
6.2.4 Problèmes de test . . . 117
6.2.5 Résultats expérimentaux . . . 118
6.2.6 Conclusion et limites du modèle . . . 128
6.3 Manœuvres latérales optimales . . . 128
6.3.1 Expression de la position . . . 129
6.3.2 Contracteur implémenté . . . 130
6.3.3 Problèmes de test . . . 134
6.3.4 Résultats expérimentaux . . . 137
6.4 Perspectives . . . 141

7 Preuve d'optimalité en dynamique moléculaire 143
7.1 Potentiel de Lennard-Jones . . . 143
7.2 Optimisation de la configuration spatiale . . . 144
7.3 Un problème ouvert . . . 145
7.4 La première preuve d'optimalité pour cinq atomes . . . 145
7.4.1 Réduction de la dépendance . . . 145
7.4.2 Réduction des symétries . . . 145
7.4.3 Preuve d'optimalité . . . 146

7.4.4 Comparaison avec des solveurs de pointe 147

Conclusion et perspectives **149**

A Calcul par intervalles **153**
A.1 Arithmétique d'intervalles 153
A.1.1 Forme de Baumann optimale 153
A.1.2 Formes centrées 154
A.1.3 Formes bicentrée et kite 157
A.2 Arithmétique affine 157
A.3 Postprocessing d'un programme linéaire 160
A.3.1 Minorant du problème primal 160
A.3.2 Non réalisabilité du problème primal 161

Bibliographie **163**

Table des figures

1 Schéma de coopération de Charibde 2

1.1 Extrema locaux et globaux 6
1.2 Minimum global d'un problème sous contraintes 9

2.1 Echantillonnage d'un algorithme à population 16
2.2 Croisement entre deux chromosomes 20
2.3 Mutation d'un chromosome 20
2.4 Scaling affine 21
2.5 Croisement quaternaire de l'évolution différentielle 24

3.1 Forme de Taylor 35
3.2 Minoration par la forme de Taylor avec le centre optimal de Baumann . . . 35
3.3 Recherche du minimum global de $f(x) = x^2\cos(x) + x$ sur $X = [-5, 3]$ par un algorithme de branch and bound par intervalles 42

4.1 Contraction d'une boîte $\boldsymbol{X}$ par rapport à une contrainte c 48
4.2 HC4Revise : phase montante d'évaluation 50
4.3 HC4Revise : phase descendante de projection 51
4.4 Recherche des zéros de $f : x \mapsto x^2 - 2$ dans $X_0 = [-3, 2]$ par l'algorithme de Newton par intervalles 54
4.5 Rognage 3B pour $s_{3B} = 8$ 57
4.6 Rognage CID pour $s_{CID} = 4$ 58
4.7 Minorations linéaires de g par deux formes de Taylor 60
4.8 Polytope de la contrainte encadré par une boîte 62
4.9 Polytope du problème relaxé 63
4.10 Réduction du polytope par l'ajout d'une contrainte implicite 64

5.1 Schéma de coopération de Charibde 69
5.2 Différence de monotonie entre f_{near} et $x \mapsto \overline{F(x)}$ 80
5.3 Signe des contraintes 83
5.4 Comparaison entre les deux versions de l'évolution différentielle 85
5.5 Comparaison entre les deux versions de l'évolution différentielle (suite) . . 86
5.6 Réduction relative du domaine initial sur les problèmes de test 88

5.7 Evolution du volume du domaine (échelle logarithmique) avec le nombre d'évaluations de la fonction objectif dans l'évolution différentielle 89
5.8 Evolution du domaine de l'évolution différentielle avec les générations . . . 90
5.9 Profil de performance : pourcentage de problèmes résolus par Charibde et Ibex en fonction du temps (échelle logarithmique) 95
5.10 Fonctions de test multimodales ($n = 2$) 97
5.11 Evolution du temps de calcul de Charibde avec la taille du problème 103
5.12 Evolution du meilleur majorant de Charibde, de l'évolution différentielle et du branch and contract par intervalles en fonction du temps de calcul . . . 105
5.13 Evolution du meilleur majorant de Charibde et Couenne en fonction du temps de calcul 108

6.1 Carte aéronautique des routes aériennes 112
6.2 Espace de séparation pour la phase de croisière 113
6.3 Point tournant 114
6.4 Zones de séparation 115
6.5 Configuration initiale C_{10} en cercle 117
6.6 Configuration initiale B_{10} en cercle bruité 118
6.7 Solution du problème C_{25} par un algorithme à évolution différentielle . . . 119
6.8 Solutions du problème B_{20} 121
6.9 Problème C_{25} 122
6.10 Problème B_{20} 123
6.11 Problème B_{30} 124
6.12 Problème B_{40} 125
6.13 Problème B_{50} 126
6.14 Vecteurs directeurs de la manœuvre 129
6.15 Rognage du paramètre universellement quantifié 134
6.16 Configuration initiale en face à face 135
6.17 Configuration initiale en dépassement 135
6.18 Configuration initiale en croisement 135
6.19 Configuration initiale symétrique à trois avions 136
6.20 Configuration initiale bruitée à trois avions 136
6.21 Configuration initiale en rond-point à trois avions 136
6.22 Solution optimale à deux avions en face à face 138
6.23 Solution optimale à deux avions en croisement 138
6.24 Solution optimale à deux avions en dépassement 138
6.25 Solution optimale à trois avions en configuration symétrique 140
6.26 Solution optimale à trois avions en configuration bruitée 140
6.27 Solution optimale à trois avions en configuration rond-point 140

7.1 Potentiel de Lennard-Jones 144
7.2 Configuration optimale de Lennard-Jones à cinq atomes 146

7.3 Rond-point à trois avions : configuration initiale et solution optimale certifiée par Charibde 150

A.1 Forme de Taylor 153
A.2 Comparaison des minorants de $f(x) = x^2 - x$ sur $X = [-2, 2]$ fournis par la forme de Taylor et la forme centrée 156
A.3 Fonctions d'inclusion de Taylor, *lbvf* et kite 158
A.4 Approximations affines de exp par les méthodes de Tchebychev et min-range 159

Liste des tableaux

1.1 Méthodes d'optimisation continue 10

3.1 Différentiation automatique en mode adjoint 45

4.1 Newton par intervalles 55

5.1 Comparaison des heuristiques de recherche de Charibde 78
5.2 Temps de convergence des deux versions de l'évolution différentielle 87
5.3 Comparaison du temps de convergence (en s) et du nombre de bissections des solveurs GlobSol, IBBA, Ibex et Charibde sur des problèmes difficiles d'optimisation sous contraintes 93
5.4 Hyperparamètres de Charibde sur les problèmes de test contraints 94
5.5 Expressions et domaines des fonctions de test multimodales 96
5.6 Temps CPU moyen et nombre moyen d'évaluations après 100 exécutions de Charibde 99
5.7 Hyperparamètres de Charibde sur les cinq fonctions de test multimodales . 99
5.8 Fonction Rana : comparaison du temps de convergence de Charibde sur deux expressions syntaxiques équivalentes ($NP = 70$, $W = 0.7$, $CR = 0.5$) 100
5.9 Minima globaux de la fonction Michalewicz certifiés par Charibde ($\varepsilon = 10^{-8}$) 101
5.10 Solutions de la fonction Michalewicz pour $n = 1$ à 70 101
5.11 Minima globaux des fonctions Sine Envelope Sine Wave, Eggholder, Keane et Rana certifiés par Charibde 102
5.12 Valeur supposée du minimum global en fonction de la taille du problème . 102
5.13 Comparaison de Charibde, de l'évolution différentielle et du branch and contract par intervalles sur les fonctions de test 106
5.14 Comparaison de Charibde et sept solveurs du serveur NEOS sur les fonctions de test 109

6.1 Durée moyenne des conflits et valeur objectif moyenne sur 20 exécutions, pour 120000 évaluations de la fonction objectif 127
6.2 Conflits à deux avions : hyperparamètres de Charibde 137
6.3 Conflits à deux avions : résultats de Charibde 137
6.4 Conflits à trois avions : hyperparamètres de Charibde 139
6.5 Conflits à trois avions : résultats de Charibde 139

7.1 Solution optimale de Lennard-Jones à cinq atomes 146
7.2 Cluster de Lennard-Jones : hyperparamètres de Charibde 147
7.3 Cluster de Lennard-Jones : résultats moyens de Charibde sur 100 exécutions 147
7.4 Cluster de Lennard-Jones : comparaison de BARON, Couenne et Charibde 148

A.1 Comparaison d'inclusions pour $f(x) = x^2 - x$ 157

Liste des algorithmes

1 Algorithme à population 16
2 Algorithme génétique 18
3 Optimisation par essaims particulaires 22
4 Evolution différentielle 23
5 Branch and bound par intervalles 40
6 MohcRevise 52
7 Boucle de propagation (point fixe) 55
8 Rognage 3B 57
9 Rognage CID 59
10 Charibde : algorithme de branch and contract par intervalles 70
11 Contracteur pour l'optimisation sans contraintes 71
12 Calcul d'un minorant par une forme de Baumann optimale 72
13 Test de monotonie pour les problèmes sous contraintes de bornes 73
14 Boucle de contraction pour les problèmes sous contraintes de bornes 73
15 Contracteur pour l'optimisation sous contraintes 74
16 Calcul d'un minorant et contraction par un contracteur linéaire 75
17 Calcul d'un minorant par monotonie et contraction 76
18 Distance entre un point et une boîte 77
19 Charibde : algorithme à évolution différentielle 79
20 Mise à jour du meilleur individu à la fin de chaque génération 80
21 Evaluation d'un individu relative à son parent 82
22 Evaluation rigoureuse des contraintes 82
23 Test de satisfaction des contraintes en arithmétique flottante 84
24 Test de satisfaction des contraintes en arithmétique d'intervalles 84
25 Evaluation d'un individu sur la boîte le contenant 91
26 Calcul de la boîte la plus proche d'un individu 92
27 Contraction par rapport à une inégalité universellement quantifiée 131
28 Contraction 2B par rapport à une inégalité universellement quantifiée . . . 131
29 Instanciation du paramètre universellement quantifié 132
30 Contraction par rapport à la négation d'une inégalité universellement quantifiée 132
31 Rognage du paramètre universellement quantifié 133

Glossaire

AA arithmétique affine.
AE algorithme évolutionnaire.
AF arithmétique flottante.
AG algorithme génétique.
AI arithmétique d'intervalles.

BB branch and bound.
BBI branch and bound par intervalles.
BCI branch and contract par intervalles.

CID constructive interval disjunction.
CSP problème de satisfaction de contraintes.

DA différentiation automatique.

ED évolution différentielle.

FMS flight management system.
FPU unité de calcul en virgule flottante.

IUQ inégalité universellement quantifiée.

NCSP problème numérique de satisfaction de contraintes.

OEP optimisation par essaims particulaires.

PPCI programmation par contraintes sur intervalles.
PUQ paramètre universellement quantifié.

Introduction

Problématique

Les calculs numériques en arithmétique flottante sont sujets à des erreurs d'arrondis ; l'accumulation de celles-ci produit parfois des résultats aberrants, souvent désastreux dans les systèmes critiques (fusées, missiles). Les seules méthodes permettant de borner de manière fiable les résultats intermédiaires d'un calcul numérique sont basées sur l'analyse par intervalles, une branche de l'analyse numérique étendant l'arithmétique flottante aux intervalles. La capacité de l'analyse par intervalles à raisonner de manière ensembliste a depuis quelques décennies ouvert de nouveaux horizons pour la communauté d'optimisation globale.

Les méthodes exhaustives fiables d'optimisation globale, appelées branch and bound par intervalles, partitionnent l'espace de recherche et éliminent par réfutation les sous-espaces qui ne peuvent contenir de solution optimale. En minimisation, la réfutation consiste à interrompre l'exploration des sous-domaines pour lesquels un minorant de la fonction objectif est plus élevé que la meilleure évaluation connue. Des techniques de filtrage (ou contraction) de la communauté d'optimisation combinatoire ont récemment inspiré des opérateurs visant à réduire les bornes des sous-domaines sans éliminer de solution optimale. Malgré tout, le processus de partitionnement reste parfois inévitable. Sa complexité exponentielle en le nombre de variables limite la taille des problèmes à au plus quelques dizaines de variables.

L'utilisation de méthodes exhaustives pour résoudre des problèmes non convexes et multimodaux peut s'avérer vaine. En ce cas, il est envisageable d'avoir recours à des techniques approchées, les métaheuristiques, qui renoncent à l'optimalité de la solution, mais produisent généralement de bonnes solutions en un temps réduit. Parmi les métaheuristiques manipulant itérativement un ensemble de solutions candidates (individus), les algorithmes évolutionnaires s'inspirent de mécanismes naturels pour guider une marche aléatoire vers de bons individus. Dotés d'opérateurs qui aident les individus à s'échapper des minima locaux, ils ont acquis leurs lettres de noblesse dans la communauté d'optimisation globale.

Les solveurs exhaustifs intègrent généralement à leur schéma de branch and bound des algorithmes d'optimisation locale afin d'obtenir une approximation du minimum global. Très peu combinent une méthode globale exacte (de type branch and bound) et une méthode globale stochastique (de type algorithme évolutionnaire) ; les approches existantes sont essentiellement séquentielles (une exécution après l'autre) ou intégratives (l'un des deux algorithmes, l'esclave, est inclus dans l'autre, le maître).

Approche adoptée et contributions

Deux étapes s'avèrent cruciales dans les algorithmes de branch and bound : générer une bonne solution réalisable (dont l'évaluation est un majorant du minimum global) et calculer un encadrement précis de la fonction objectif et des contraintes sur un sous-domaine. Dans ce document, nous nous intéressons à un cadre coopératif combinant des algorithmes évolutionnaires et des méthodes d'intervalles avancées. Dans notre solveur hybride Charibde, les deux algorithmes exécutés en parallèle échangent bornes, solutions et espace de recherche par passage de messages (figure 1). L'algorithme évolutionnaire explore rapidement l'espace de recherche et fournit une bonne solution réalisable. Son évaluation est alors transmise à la méthode d'intervalles afin d'intensifier les coupes de l'espace de recherche. Lorsqu'une solution ponctuelle trouvée par la méthode d'intervalles améliore la meilleure solution trouvée, elle est en retour intégrée à la population courante afin de guider la population et d'empêcher la convergence prématurée vers des minima locaux. Une stratégie couplant une heuristique d'exploration géométrique et un opérateur de réduction de domaine évite à l'algorithme évolutionnaire d'explorer les sous-espaces sous-optimaux ou non réalisables.

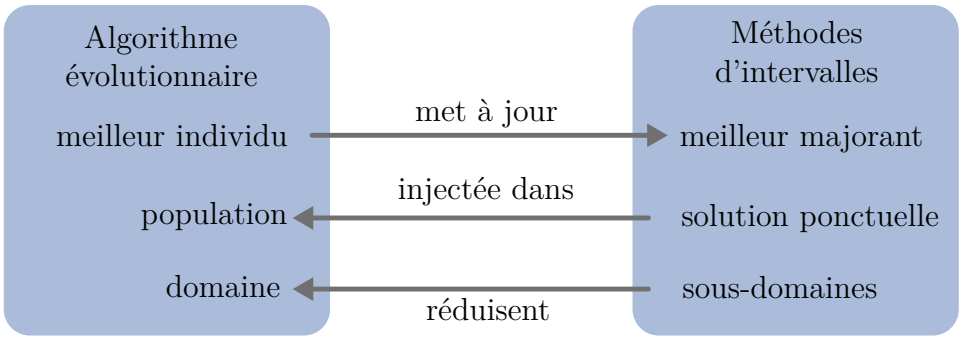


FIGURE 1 – Schéma de coopération de Charibde

Charibde se montre compétitif par rapport aux solveurs de pointe : une comparaison avec les solveurs GlobSol, IBBA et Ibex sur des problèmes difficiles de la base COCONUT montre un gain en temps de calcul d'un ordre de grandeur. De nouveaux résultats optimaux sont fournis pour cinq problèmes multimodaux (Michalewicz, Sine Wave Sine Envelope, Eggholder, Keane, Rana) pour lesquels peu de solutions, même approchées, sont connues. La meilleure solution connue pour le cluster de Lennard-Jones à cinq atomes, un problème ouvert en dynamique moléculaire, est numériquement certifiée par Charibde. Nous montrons que les solveurs non-linéaires BARON et Couenne fournissent des solutions numériquement erronées. Enfin, nous proposons une application aéronautique dans laquelle la résolution de conflits est modélisée par un problème d'optimisation sous contraintes : une comparaison de trois algorithmes à population sur différentes instances de test met en évidence la complexité du problème et les caractéristiques de chacune des méthodes. Le problème est ensuite résolu sur de petites instances par des méthodes d'intervalles, en considérant les contraintes de séparation comme des contraintes universellement quantifiées.

Organisation du document

Ce document est constitué de sept chapitres. Le chapitre 1 situe le contexte mathématique de l'étude et fournit quelques rappels sur l'existence d'un minimum dans les problèmes contraints et non contraints, et les différentes techniques de résolution exploitant la structure du problème. Nous détaillons trois algorithmes à population dans le chapitre 2. Le chapitre 3 introduit les méthodes d'intervalles, leur application à l'optimisation globale et les techniques de différentiation automatique. Le chapitre 4 étend le précédent chapitre et compare les techniques de contraction sur intervalles, héritées des communautés d'analyse par intervalles et de programmation par contraintes. Notre solveur Charibde est présenté dans le chapitre 5. Nous détaillons l'implémentation adoptée et les techniques avancées spécifiques à notre solveur. Nous traitons dans le chapitre 6 une application aéronautique, la résolution de conflits entre avions en phase de croisière, consistant à garantir la séparation entre appareils à tout instant. Elle est modélisée par un problème d'optimisation sous contraintes, puis traitée par des algorithmes à population et des méthodes d'intervalles. Dans le chapitre 7, nous résolvons un problème ouvert de dynamique moléculaire, le problème de cluster de Lennard-Jones à cinq atomes.

Chapitre 1

Optimisation continue

Sommaire

1.1 Problèmes d'optimisation 5
1.1.1 Minima locaux et globaux 6
1.1.2 Existence d'un minimum 7
1.1.3 Optimisation sans contraintes 7
1.1.4 Optimisation sous contraintes 8
1.2 Techniques de résolution 10
1.2.1 Problèmes linéaires 11
1.2.2 Problèmes convexes 11
1.2.3 Problèmes non convexes 12

L'optimisation est la discipline permettant de déterminer analytiquement ou numériquement la meilleure solution à un problème, au sens d'un certain critère. Elle est fondamentale dans la résolution de nombreux problèmes liés à l'industrie, à l'économie ou aux sciences physiques, en vue d'obtenir un gain d'effort ou de temps. La qualité de la solution obtenue dépend généralement du modèle utilisé pour représenter le problème réel et de la méthode de résolution adoptée.

La section 1.1 traite des problèmes d'optimisation (non contraints et sous contraintes), ainsi que des conditions nécessaires et suffisantes de l'existence d'un minimum. Les principales méthodes de résolution sont mentionnées dans la section 1.2.

1.1 Problèmes d'optimisation

Un problème d'optimisation continue s'exprime sous la forme standard :

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \text{ sous la contrainte } \boldsymbol{x} \in D \tag{1.1}$$

Les variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ sont les *variables de décision*. La fonction $f : D \subset \mathbb{R}^n \to \mathbb{R}$ est appelée *fonction objectif* et D est le *domaine réalisable*. Tout point $\boldsymbol{x} \in \mathbb{R}^n$ appartenant

à D est appelé *point réalisable*. Observons que maximiser une fonction f est équivalent à minimiser la fonction $-f$:

$$\max_{\boldsymbol{x}\in D} f(\boldsymbol{x}) = -\min_{\boldsymbol{x}\in D}(-f(\boldsymbol{x})) \tag{1.2}$$

1.1.1 Minima locaux et globaux

Résoudre un problème d'optimisation consiste à chercher un minimum local ou global (définition 1) d'une fonction, et (ou) l'ensemble des minimiseurs correspondants.

Définition 1 (Minima et minimiseurs) *Soit* $\boldsymbol{x} \in \mathbb{R}^n$.

- *on dit que* $\boldsymbol{x}$ *est un* minimiseur local *de* f *sur* $D \subset \mathbb{R}^n$ *lorsque* $\boldsymbol{x} \in D$ *et il existe un voisinage ouvert* V *de* $\boldsymbol{x}$ *tel que :*

$$\forall \boldsymbol{y} \in D \cap V, \quad f(\boldsymbol{x}) \leq f(\boldsymbol{y}) \tag{1.3}$$

  $f(\boldsymbol{x})$ *est alors un* minimum local *de* f *sur* D *;*
- *on dit que* $\boldsymbol{x}$ *est un* minimiseur global *de* f *sur* $D \subset \mathbb{R}^n$ *lorsque* $\boldsymbol{x} \in D$ *et :*

$$\forall \boldsymbol{y} \in D, \quad f(\boldsymbol{x}) \leq f(\boldsymbol{y}) \tag{1.4}$$

  $f(\boldsymbol{x})$ *est alors un* minimum global *de* f *sur* D.

On définit de manière similaire les maximiseurs et maxima locaux et globaux.

La figure 1.1 illustre les extrema (minima et maxima) locaux et globaux d'une fonction réelle univariée continue sur un intervalle. Le caractère local d'un extremum n'est déterminé que dans un certain voisinage.

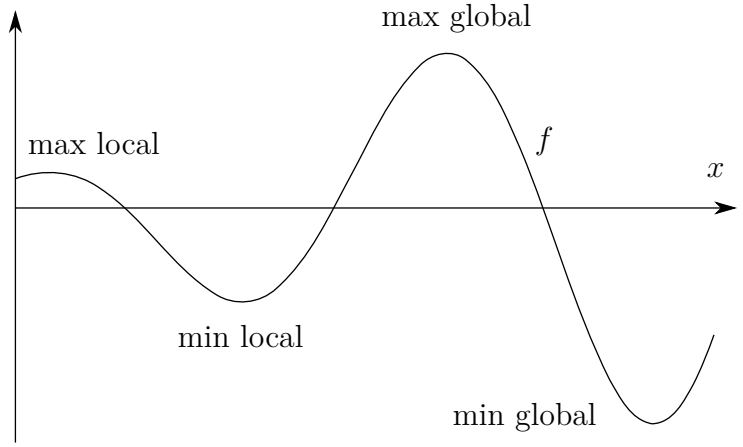


FIGURE 1.1 – Extrema locaux et globaux

1.1.2 Existence d'un minimum

Le théorème de Weierstrass (théorème 1) indique que le problème 1.1 admet un minimum lorsque f est continue et D est fermé borné.

Théorème 1 (Théorème de Weierstrass) *Soient D un ensemble fermé borné non vide de $\mathbb{R}^n$ et $f : D \to \mathbb{R}$ une application continue sur D. Alors f est bornée et atteint ses bornes. En particulier, il existe $\boldsymbol{x} \in D$ tel que :*

$$\forall y \in D, \quad f(x) \leq f(y) \tag{1.5}$$

1.1.3 Optimisation sans contraintes

Nous nous intéressons dans cette section à la recherche des points qui minimisent une fonction $f : \mathbb{R}^n \to \mathbb{R}$:

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) \tag{1.6}$$

où f est supposée au moins différentiable.

Lorsque f est différentiable, on note ∇f son gradient. Lorsque f est deux fois différentiable, on note $\nabla^2 f$ la matrice hessienne de f ayant $\frac{\partial^2 f}{\partial x_i \partial x_j}$ pour élément en ligne i et en colonne j. Le théorème 2 introduit les conditions nécessaires d'optimalité qui caractérisent les minima locaux de f.

Définition 2 (Point stationnaire) *Soit $f : \mathbb{R}^n \to \mathbb{R}$ une application différentiable. On appelle* point stationnaire *de f tout point $\boldsymbol{x} \in \mathbb{R}^n$ vérifiant :*

$$\nabla f(\boldsymbol{x}) = 0 \tag{1.7}$$

Théorème 2 (Conditions nécessaires d'optimalité locale) *Soit $f : \mathbb{R}^n \to \mathbb{R}$ une application différentiable. Si $\boldsymbol{x}^* \in \mathbb{R}^n$ est un minimum local de f :*

1. *$\boldsymbol{x}^*$ est un point stationnaire (définition 2) de f (condition du premier ordre) ;*
2. *si f est deux fois différentiable dans un voisinage ouvert de $\boldsymbol{x}^*$, alors $\nabla^2 f(\boldsymbol{x}^*)$ est semi-définie positive (condition du deuxième ordre).*

Remarque 1 *La stationnarité d'un minimum local est une condition* nécessaire *mais non* suffisante *: la fonction $f(x) = x^3$ admet un point stationnaire en $x = 0$ vérifiant également la condition du deuxième ordre, mais il ne s'agit pas d'un minimum local.*

Bien que non suffisantes, les conditions du théorème 2 permettent de sélectionner des points qui sont de potentiels minima locaux. Afin de déterminer s'il s'agit effectivement de minima locaux, la condition suffisante du théorème 3 doit être vérifiée.

Théorème 3 (Condition suffisante d'optimalité locale) *Soit $f : \mathbb{R}^n \to \mathbb{R}$ une application différentiable dans un voisinage ouvert de $\boldsymbol{x}^* \in \mathbb{R}^n$ et deux fois différentiable en $\boldsymbol{x}^*$. Si $\nabla f(\boldsymbol{x}^*) = 0$ et $\nabla^2 f(\boldsymbol{x}^*)$ est définie positive, alors $\boldsymbol{x}^*$ est un* minimum local *de f.*

1.1.4 Optimisation sous contraintes

Dans cette sous-section, le domaine réalisable $D \subset \mathbb{R}^n$ est défini par des contraintes d'égalité et d'inégalité (définition 3) :

$$D = \{\boldsymbol{x} \in \mathbb{R}^n \mid g(\boldsymbol{x}) \leq 0 \wedge h(\boldsymbol{x}) = 0\} \tag{1.8}$$

où $g : \mathbb{R}^n \to \mathbb{R}^m$ et $h : \mathbb{R}^n \to \mathbb{R}^p$ sont continues. Un problème d'optimisation sous contraintes sera désormais exprimé sous la forme standard (*s.c.* signifiant *sous contraintes*) :

$$\begin{array}{lll} (\mathcal{P}) & \min\limits_{\boldsymbol{x}\in\mathbb{R}^n} & f(\boldsymbol{x}) \\ & s.c. & g_i(\boldsymbol{x}) \leq 0, \quad i \in \{1, \ldots, m\} \\ & & h_j(\boldsymbol{x}) = 0, \quad j \in \{1, \ldots, p\} \end{array} \tag{1.9}$$

Définition 3 (Contrainte, relation) *Soient $\mathcal{V} = (x_1, \ldots, x_n)$ un ensemble de variables et D le domaine des variables de $\mathcal{V}$. Une* contrainte *c est une formule logique s'exprimant comme :*

$$c(x_1, \ldots, x_n) \diamond 0 \tag{1.10}$$

où $\diamond \in \{\leq, \geq, =\}$. On note réciproquement $var(c) = \mathcal{V}$ l'ensemble des variables intervenant dans la contrainte c. On appelle relation *ρ_c de c l'ensemble des solutions de c.*

La condition nécessaire du premier ordre d'optimalité locale (théorème 2) n'est plus valable lorsque le problème est contraint. L'exemple 1 montre que le minimum global, lorsqu'il n'est pas un point stationnaire, se situe sur la frontière de la contrainte (la contrainte est dite active, voir définition 4).

Exemple 1 *Soit le problème d'optimisation :*

$$\begin{array}{lll} (\mathcal{P}) & \min\limits_{x\in\mathbb{R}} & f(x) = x^2 \\ & s.c. & x \geq 1 \end{array} \tag{1.11}$$

Sur la figure 1.2, le rectangle mauve décrit le domaine réalisable $\{x \in \mathbb{R} \mid x \geq 1\}$ de $(\mathcal{P})$. Le minimum global de $(\mathcal{P})$ est $x^* = 1$ (représenté par un disque rouge sur la figure), qui n'est pas un point stationnaire : $f'(x^*) = 2 \neq 0$.

Définition 4 (Contrainte active/inactive) *Une contrainte d'inégalité $g \leq 0$ est dite* active *en $\boldsymbol{x} \in \mathbb{R}^n$ lorsque $g(\boldsymbol{x}) = 0$, et* inactive *lorsque $g(\boldsymbol{x}) < 0$.*

Les conditions nécessaires d'optimalité dans le cas contraint (théorème 4 [Karush 39]) reposent notamment sur la séparation entre les contraintes d'inégalité actives et inactives. Une condition supplémentaire, dite de qualification (définition 5), est requise.

Définition 5 (Condition de qualification) *Les contraintes de $(\mathcal{P})$ sont dites* qualifiées en *$\boldsymbol{x} \in \mathbb{R}^n$ si les gradients des contraintes d'inégalité actives et les gradients des contraintes d'égalité sont linéairement indépendants en $\boldsymbol{x}$.*

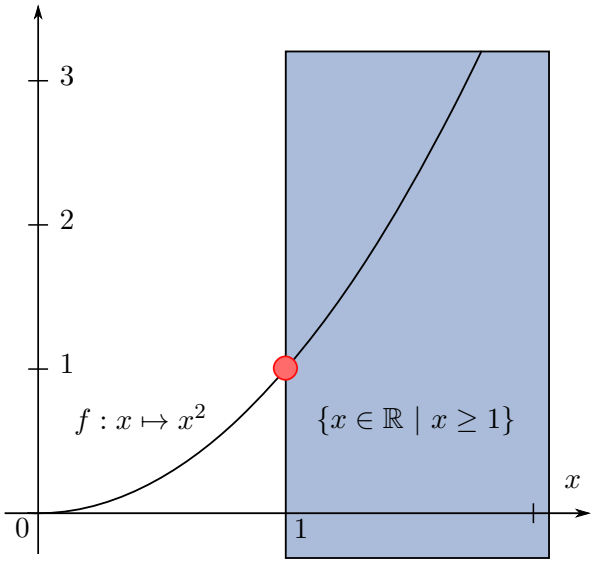


FIGURE 1.2 – Minimum global d'un problème sous contraintes

Théorème 4 (Conditions d'optimalité de Karush-Kuhn-Tucker (KKT)) *On suppose que f, g_i ($i \in \{1, \ldots, m\}$) et h_j ($j \in \{1, \ldots, p\}$) sont continûment différentiables en un point $\boldsymbol{x}^* \in \mathbb{R}^n$, et que les contraintes g_i et h_j sont qualifiées en $\boldsymbol{x}^*$. Si $\boldsymbol{x}^*$ est un minimum local de f, alors il existe les réels λ_i ($i \in \{1, \ldots, m\}$) et μ_j ($j \in \{1, \ldots, p\}$), appelés multiplicateurs de Lagrange, vérifiant les conditions suivantes :*

Stationnarité

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\boldsymbol{x}^*) + \sum_{j=1}^{p} \mu_j \nabla h_j(\boldsymbol{x}^*) = 0 \tag{1.12}$$

Faisabilité primale

$$\begin{aligned} g_i(\boldsymbol{x}^*) &\leq 0, \qquad i \in \{1, \ldots, m\} \\ h_j(\boldsymbol{x}^*) &= 0, \qquad j \in \{1, \ldots, p\} \end{aligned} \tag{1.13}$$

Faisabilité duale

$$\lambda_i \geq 0, \quad i \in \{1, \ldots, m\} \tag{1.14}$$

Complémentarité

$$\lambda_i g_i(\boldsymbol{x}^*) = 0, \quad i \in \{1, \ldots, m\} \tag{1.15}$$

La condition de complémentarité indique que si g_i est inactive en $\boldsymbol{x}^*$ (c'est-à-dire si $g_i(\boldsymbol{x}^*) < 0$), alors $\lambda_i = 0$. Les conditions de KKT peuvent être interprétées comme la recherche d'un minimum local dans laquelle les contraintes d'inégalité actives sont remplacées par des contraintes d'égalité et les contraintes d'inégalité inactives sont ignorées.

L'hypothèse de qualification dans les conditions de KKT est nécessaire pour garantir l'existence des multiplicateurs de Lagrange. Le cas contraire est illustré dans l'exemple 2.

Exemple 2 (Contrainte non qualifiée) *Soit le problème d'optimisation :*

$$\begin{array}{lll} (\mathcal{P}) & \min\limits_{x\in\mathbb{R}} & f(x) = x \\ & s.c. & g(x) = -x^3 \leq 0 \end{array} \tag{1.16}$$

La solution optimale de $(\mathcal{P})$ *est* $x^* = 0$*. Pourtant, il n'existe pas de* $\lambda \in \mathbb{R}$ *vérifiant les conditions de KKT, c'est-à-dire tel que* $f'(x^*) + \lambda g'(x^*) = 0$*. Ceci est dû au fait que* $g'(x^*) = 0$*, c'est-à-dire que* g *n'est pas une contrainte qualifiée en* x^**.*

Notons que tous les points qui satisfont les conditions de KKT ne sont pas des minima locaux, de la même manière que tous les points stationnaires d'un problème d'optimisation sans contraintes ne sont pas minima locaux.

1.2 Techniques de résolution

De nombreuses techniques de résolution permettent de résoudre une grande variété de problèmes d'optimisation présentant une expression analytique particulière. Le tableau 1.1 (non exhaustif) liste les principales méthodes d'optimisation continue. "NC" désigne les problèmes non convexes, "$g \leq 0$" les problèmes contraints et "$n \uparrow$" les problèmes de grande taille. Une méthode déterministe parcourt l'espace de recherche toujours de la même manière, tandis que deux exécutions d'une méthode stochastique peuvent fournir des résultats différents. Notons que les méthodes stochastiques dites globales sont dotées de mécanismes permettant de s'échapper des minima locaux, mais ne garantissent pas pour autant l'optimalité de la solution.

Table 1.1 – Méthodes d'optimisation continue

	Problème			Méthode		
	NC	$g \leq 0$	$n \uparrow$	sans dérivées	déterministe	globale
Programmation linéaire		✓	✓	✓	✓	✓
Points intérieurs		✓	✓	✓	✓	✓
Sous-gradient		✓	✓	✓	✓	✓
Quasi-Newton	✓		✓		✓	
Recuit simulé	✓		✓	✓		✓
Nelder-Mead	✓		✓	✓	✓	
Algorithmes à pop.	✓	✓	✓	✓		✓
Lipschitz	✓	✓	✓	✓	✓	✓
Méthodes d'intervalles	✓	✓		✓	✓	✓

Nous mentionnons dans les sous-sections suivantes l'optimisation linéaire, l'optimisation convexe et l'optimisation non convexe, ainsi que les principales méthodes de résolution associées.

1.2.1 Problèmes linéaires

Résoudre un problème linéaire revient à minimiser une fonction linéaire (définition 6) sur un polytope (un polyèdre convexe) de $\mathbb{R}^n$ défini par des (in)équations linéaires.

Définition 6 (Fonction linéaire) *Soit $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ un vecteur de n variables réelles. On appelle* fonction linéaire *réelle de $\boldsymbol{x}$ une fonction $f : \mathbb{R}^n \to \mathbb{R}$ qui s'exprime comme une combinaison linéaire des variables de $\boldsymbol{x}$:*

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} a_i x_i = \boldsymbol{a}^\top \boldsymbol{x} \tag{1.17}$$

où $\boldsymbol{a}$ est un vecteur de coefficients réels.

Un problème linéaire (fonction objectif et contraintes sont linéaires) s'exprime généralement sous la forme canonique donnée par la définition 1.18. Les deux techniques de résolution les plus répandues sont l'algorithme du simplexe, dont la complexité dans le pire des cas est exponentielle mais est très efficace en pratique, et les méthodes de points intérieurs.

Définition 7 (Problème linéaire) *Soient $\boldsymbol{x}$ un vecteur de n variables réelles, $\boldsymbol{c}$ un vecteur de n réels, $\boldsymbol{b}$ un vecteur de m réels et A une matrice réelle de taille $m \times n$. Le* problème linéaire *associé s'écrit sous forme canonique :*

$$\begin{array}{ll} (\mathcal{P}) \quad \min\limits_{\boldsymbol{x} \in \mathbb{R}^n} & \boldsymbol{c}^\top \boldsymbol{x} \\ \quad s.c. & A\boldsymbol{x} \leq \boldsymbol{b} \end{array} \tag{1.18}$$

1.2.2 Problèmes convexes

Un problème convexe est un problème dont la fonction objectif est convexe (définition 1.19) et dont l'espace réalisable est convexe (définition 9).

Définition 8 (Fonction convexe) *Une fonction f définie sur un intervalle réel I est dite* convexe *lorsque :*

$$\forall (x, y) \in I^2, \quad \forall t \in [0, 1], \qquad f(tx + (1-t)y) \leq tf(x) + (1-t)f(y) \tag{1.19}$$

On dit que f est strictement convexe *lorsque :*

$$\forall (x, y) \in I^2, x \neq y, \quad \forall t \in]0, 1[, \qquad f(tx + (1-t)y) < tf(x) + (1-t)f(y) \tag{1.20}$$

Définition 9 (Ensemble convexe) *Un ensemble C est dit* convexe *lorsque :*

$$\forall (x, y) \in C^2, \quad \forall t \in [0, 1], \qquad tx + (1-t)y \in C \tag{1.21}$$

On montre facilement qu'une solution locale d'un problème d'optimisation strictement convexe est l'unique solution globale du problème.

1.2.3 Problèmes non convexes

De nombreux problèmes pratiques sont non convexes ; la convergence vers un minimum local ne suffit alors pas à garantir son optimalité globale. La présence de multiples optima locaux handicape fortement les méthodes de résolution lorsqu'il s'agit de déterminer si la solution trouvée est globale ou s'il n'existe pas de solution.

Méthodes locales

L'utilisation de méthodes locales, explorant un voisinage d'une solution initiale et fournissant un minimum local, semble souvent un bon compromis entre qualité de résultat et effort de recherche. On distingue deux grandes catégories de méthodes locales :

- les méthodes d'optimisation mathématique sont généralement basées sur des informations du premier ordre. L'algorithme du gradient améliore successivement une solution initiale dans la direction opposée au gradient, afin de faire décroître la valeur de la fonction objectif. La méthode itérative de Newton consiste à linéariser la fonction en l'itéré courant, puis à chercher le zéro de la linéarisation ;
- les méthodes heuristiques sont des méthodes générales de recherche de solutions approchées. La méthode de Nelder-Mead [Nelder 65] fait évoluer un polytope de $n+1$ sommets dans un espace à n dimensions, subissant des transformations géométriques simples au cours des itérations, jusqu'à éventuellement se rapprocher d'un minimum local. La recherche par motif [Hooke 61] maintient de manière similaire $2n+1$ points dans l'espace de recherche.

Méthodes globales

Les méthodes globales recherchent un minimum global sur l'ensemble du domaine. On en distingue deux familles :

- les métaheuristiques sont une famille de méthodes génériques, basées sur des mécanismes tels que la mémoire locale (recherche tabou [Glover 90]), la recherche gloutonne (GRASP [Feo 89]) ou la recherche aléatoire (algorithmes à population, recuit simulé [Kirkpatrick 83]) ;
- les méthodes déterministes explorent l'espace de recherche de manière exhaustive afin d'identifier le minimum global. Elles comptent les algorithmes de branch and bound (voir chapitre 3) et l'optimisation lipschitzienne.

Méthodes fiables

Les méthodes d'optimisation globale déterministes, bien qu'exhaustives, ne permettent pas de garantir l'optimalité de la solution avec une précision numérique donnée. Une comparaison des principaux solveurs d'optimisation globale [Neumaier 05] montre qu'une grande majorité d'entre eux souffre d'approximations numériques dues aux arrondis de l'arithmétique flottante.

Le seul outil actuellement capable de fournir un encadrement rigoureux d'un calcul numérique, même en présence d'arrondis, est l'analyse par intervalles. Un panel de techniques fiables basées sur les intervalles est présenté dans le chapitre 3.

Chapitre 2

Algorithmes à population

Sommaire

2.1 Algorithmes à population 16
2.1.1 Gestion des contraintes 17
2.1.2 Algorithmes utilisés 17
2.2 Algorithme génétique 18
2.2.1 Sélection des parents 18
2.2.2 Croisement et mutation 19
2.2.3 Opérateur de scaling 20
2.2.4 Opérateur de sharing 21
2.2.5 Remplacement de la population 21
2.3 Optimisation par essaims particulaires 22
2.3.1 Formules de mise à jour 22
2.3.2 Voisinage 23
2.4 Algorithme à évolution différentielle 23
2.4.1 Opérateur de croisement quaternaire 24
2.4.2 Choix de l'individu de base 24
2.4.3 Contraintes de bornes 25
2.4.4 Gestion directe des contraintes 25
2.5 Critères d'arrêt 25

Les algorithmes à population sont des marches aléatoires guidées par des heuristiques, qui font évoluer un ensemble de solutions candidates (individus) dans l'espace de recherche, dans le but d'obtenir de bonnes solutions à un problème d'optimisation. Appartenant à la famille des métaheuristiques, ils ne nécessitent aucune hypothèse sur la régularité de la fonction objectif (continuité, différentiabilité) comme peuvent l'exiger certaines techniques d'optimisation mathématique (descente de gradient, quasi-Newton). Seule une procédure d'évaluation de la fonction objectif en un point de l'espace de recherche est requise. Les algorithmes à population sont régis par des mécanismes permettant d'alterner des phases

de *diversification* (exploration de l'espace de recherche) et d'*intensification* (convergence locale dans le voisinage d'une bonne solution).

La section 2.1 présente le schéma général des algorithmes à population. En particulier, nous détaillons les algorithmes génétiques dans la section 2.2, l'optimisation par essaims particulaires dans la section 2.3 et les algorithmes à évolution différentielle dans la section 2.4.

2.1 Algorithmes à population

Les algorithmes à population (algorithme 1) sont munis de mécanismes permettant de s'échapper des minima locaux, bien qu'ils ne fournissent en général aucune garantie d'optimalité globale de la solution. Ils sont par conséquent un outil particulièrement adapté à l'optimisation de problèmes difficiles, multimodaux, boîtes noires (dont l'expression analytique n'est pas connue), bruités, dynamiques, etc. pour lesquels d'autres méthodes d'optimisation échouent à trouver une bonne solution. Ils échantillonnent l'espace de recherche de manière stochastique, guidés par les individus les plus prometteurs (figure 2.1) et tentent de trouver un minimum global parmi une multitude de minima locaux.

Algorithme 1 Algorithme à population

Initialiser la population initiale P
Evaluer les individus
répéter
    Générer une nouvelle population P'
    Déterminer la nouvelle population par sélection dans $P \cup P'$
jusqu'à critère d'arrêt vérifié
renvoyer le meilleur individu de la population

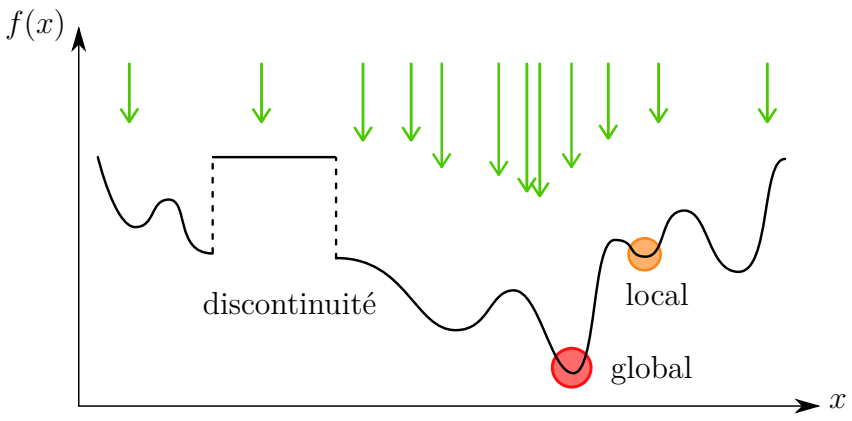


FIGURE 2.1 – Echantillonnage d'un algorithme à population

2.1.1 Gestion des contraintes

L'omniprésence des problèmes d'optimisation sous contraintes a motivé le développement de mécanismes capables de gérer des contraintes linéaires/non linéaires ou d'égalité/d'inégalité [Michalewicz 96b, Price 06, Talbi 09]. Parmi les principales méthodes :

- les stratégies de rejet n'exploitent pas les individus non réalisables, qui sont éliminés. Cette stratégie n'est pertinente que lorsque l'espace non réalisable est petit devant l'espace de recherche ;
- les méthodes de préservation génèrent un nouvel individu réalisable à partir d'un individu réalisable ;
- les méthodes de pénalités (incluant les barrières logarithmiques et l'ajout d'une pénalité mesurant la violation de la contrainte) consistent à agréger les contraintes du problème à la fonction objectif sous la forme d'une somme pondérée de pénalités p_i, de poids w_i :

$$f_p(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} w_i p_i(g_i(\boldsymbol{x})) \tag{2.1}$$

  Les méthodes de pénalités souffrent de plusieurs défauts : si les poids w_i ne sont pas correctement ajustés, la somme pondérée présente le risque d'être dominée par une pénalité, ou la métaheuristique peut rester piégée dans une région non réalisable si les valeurs de la fonction objectif sont beaucoup plus faibles que les violations des contraintes ;
- les méthodes de réparation sont des heuristiques spécifiques au problème, généralement gloutonnes, visant à transformer un individu non réalisable en un individu réalisable ;
- les méthodes de décodage établissent une bijection entre l'ensemble des représentations possibles des individus et l'ensemble des solutions réalisables du problème ;
- les méthodes directes considèrent une relation d'ordre sur les individus : les individus réalisables sont considérés comme mieux évalués que les individus non réalisables. L'exploration est donc guidée vers les zones réalisables de l'espace de recherche.

2.1.2 Algorithmes utilisés

Deux sous-familles principales des algorithmes à population sont les algorithmes évolutionnaires (AE) et les algorithmes à intelligence distribuée :

1. les AE s'inspirent du mécanisme d'évolution d'une population dans son environnement. D'un côté, des opérateurs de variation apportent de la diversité à la population afin de favoriser l'exploration de l'espace de recherche. De l'autre, des opérateurs de sélection et de remplacement intensifient la recherche dans le voisinage d'une solution. Les algorithmes génétiques (AG) et les algorithmes à évolution différentielle (ED) sont respectivement décrits dans les sections 2.2 et 2.4 ;
2. le concept d'intelligence distribuée est né de l'observation de la dynamique collective de colonies d'insectes. L'interaction entre individus, dont les actions simples sont indépendantes de toute règle globale, permet l'émergence de comportements collectifs

cohérents. L'optimisation par essaims particulaires (OEP) est décrite dans la section 2.3.

2.2 Algorithme génétique

Les AG s'inspirent de la théorie darwinienne de la sélection naturelle [Holland 75] : les gènes qui sont le plus adaptés aux besoins d'une espèce dans son environnement ont davantage de chances d'être conservés au sein d'une population au cours du temps. Un AG établit une correspondance entre :

- le génotype (l'ensemble des gènes portés par les chromosomes) d'un individu et les composantes de la solution ;
- le phénotype (caractère observable au niveau macroscopique) et l'évaluation de la solution par la fonction objectif.

En imitant certains processus tels que l'hérédité, la mutation et la sélection naturelle, un AG fait évoluer une population de N individus (un ensemble de solutions candidates) qui est partiellement remplacée à chaque génération (itération). Une implémentation possible est donnée dans l'algorithme 2.

Algorithme 2 Algorithme génétique

```
Initialiser la population initiale
Evaluer les individus
répéter
    Sélectionner les parents
    Croiser les parents et muter les nouveaux individus
    Evaluer les nouveaux individus
    Conserver les meilleurs individus
jusqu'à critère d'arrêt vérifié
renvoyer le meilleur individu de la population
```

Les AG, parmi les plus anciens AE, utilisent à l'origine un codage binaire (des gènes dans $\{0, 1\}$) pour résoudre des problèmes d'optimisation combinatoire. Ils ont depuis été étendus aux problèmes continus en adoptant un codage réel. Les AG ont été appliqués avec succès à une large variété de problèmes difficiles en bioinformatique, économie ou chimie.

2.2.1 Sélection des parents

Les individus les mieux adaptés à leur environnement sont plus enclins à se reproduire et transmettre leur patrimoine génétique à leur progéniture, tandis que les moins adaptés meurent avant la reproduction. Deux schémas de sélection des parents sont généralement considérés : le tirage à la roulette [Goldberg 89] et la *Stochastic remainder without replacement selection* [Goldberg 89].

Roulette

En maximisation, le tirage à la roulette consiste à sélectionner un individu i ($i \in \{1, \ldots, N\}$) comme parent avec une probabilité p_i proportionnelle à son évaluation $f_i \geq 0$:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \in [0, 1] \tag{2.2}$$

Le tirage à la roulette s'interprète comme le tirage d'un numéro sur une roulette de casino dont les encoches seraient espacées de manière non uniforme.

Les individus les mieux évalués ont davantage de chances de se reproduire. Néanmoins, ceci n'exclut pas que des individus d'évaluations plus faibles parviennent à transmettre leur patrimoine génétique et participent à l'amélioration de la solution courante. Lorsque la dimension de la population est faible, un biais de sélection peut toutefois exister en raison du peu de tirages effectués.

Stochastic remainder without replacement selection

La *Stochastic remainder without replacement selection* évite le biais du tirage à la roulette. Pour chaque individu i, on calcule le rapport :

$$r_i = \frac{N f_i}{\sum_{j=1}^{N} f_j} \tag{2.3}$$

L'individu i est répliqué exactement $\lfloor r_i \rfloor$ fois, où $\lfloor \cdot \rfloor$ est la fonction partie entière. Le tirage à la roulette est ensuite appliqué sur les individus affectés de l'évaluation $r_i - \lfloor r_i \rfloor$. Lorsque la taille de la population est faible, ce schéma de sélection donne en général de meilleurs résultats.

2.2.2 Croisement et mutation

Le croisement et la mutation sont des opérateurs participant à la diversification et à l'intensification de la population ; leur contribution varie selon le choix d'implémentation.

Lors d'un croisement, les chromosomes de deux parents échangent une ou plusieurs portions de matériel génétique avec une probabilité de croisement $p_c \in]0, 1[$. La progéniture issue du croisement, généralement deux chromosomes enfants, possède alors des chromosomes brassés (figure 2.2).

Un gène peut être substitué aléatoirement à un autre (muté) durant la phase de reproduction (figure 2.3). L'individu muté possède alors une séquence génétique qui n'est pas exclusivement issue de ses deux parents. Le taux de mutation p_m contrôle le caractère aléatoire de la recherche : il permet d'éviter la convergence vers un minimum local en s'échappant de son voisinage. p_m est généralement choisi faible pour conserver l'évolution naturelle de la population et éviter de transformer l'AG en une simple recherche aléatoire.

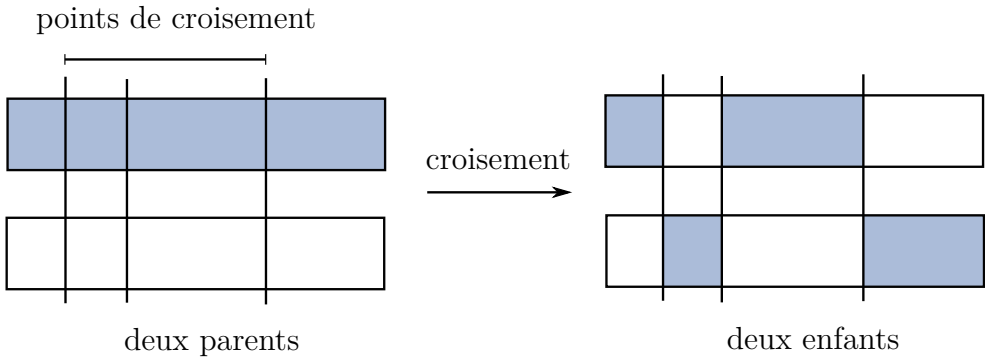


FIGURE 2.2 – Croisement entre deux chromosomes

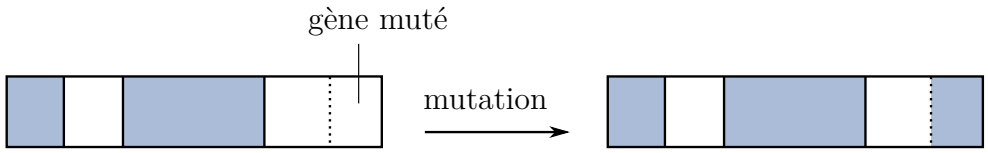


FIGURE 2.3 – Mutation d'un chromosome

2.2.3 Opérateur de scaling

Le scaling ou mise à l'échelle [Michalewicz 96a] consiste à amplifier ou réduire artificiellement les écarts d'évaluations entre les individus afin d'ajuster la pression de sélection. La fonction de scaling appliquée à l'évaluation initiale peut prendre différentes formes, par exemple affine et exponentielle.

Scaling affine

Pour chaque individu i, l'évaluation mise à l'échelle f_i^{sc} est une fonction affine de l'évaluation initiale f_i :

$$f_i^{sc} = af_i + b \tag{2.4}$$

a et b peuvent varier au cours des générations. Le paramètre positif a influe sur la pression de sélection des individus (figure 2.4). La pression est réduite lorsque $a < 1$, ce qui favorise une exploration plus large de l'espace de recherche. Lorsque $a > 1$, les individus les mieux évalués sont favorisés, ce qui privilégie l'intensification au détriment de l'exploration.

Scaling exponentiel

Pour chaque individu i, l'évaluation mise à l'échelle f_i^{sc} est une fonction exponentielle de l'évaluation initiale f_i :

$$f_i^{sc} = (f_i)^p \tag{2.5}$$

Lorsque le réel positif p est proche de 1, le scaling reste relativement neutre. En revanche, l'exploration de l'espace de recherche est privilégiée pour $p < 1$ et est d'autant plus aléatoire

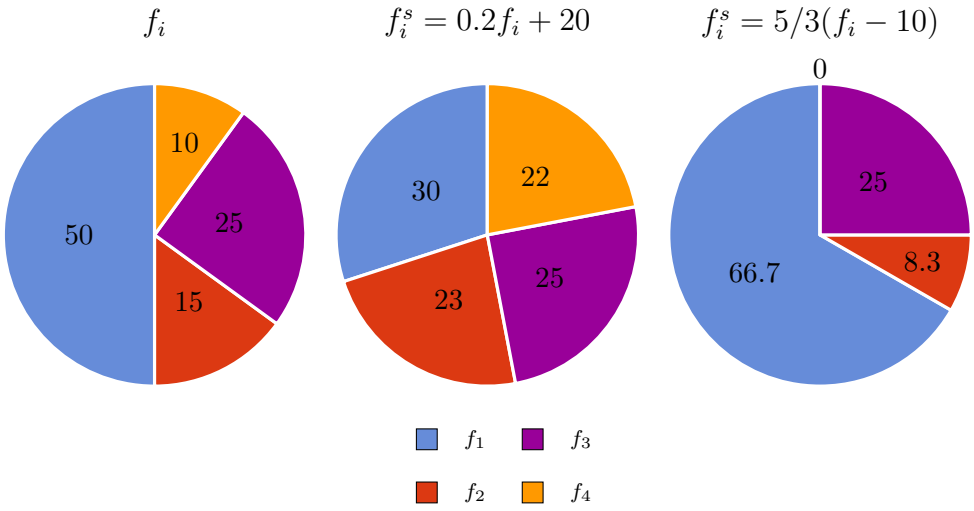


FIGURE 2.4 – Scaling affine

que p tend vers 0. Lorsque $p > 1$, la sélection des individus les mieux évalués est davantage favorisée.

2.2.4 Opérateur de sharing

Le sharing [Goldberg 87] est un opérateur visant à empêcher une convergence trop rapide de la population vers un optimum local. Le sharing pénalise les individus trop proches ; la donnée d'une distance est donc nécessaire pour estimer la densité d'individus dans l'espace de recherche.

Une première approche consiste à diviser (en maximisation) l'évaluation de chaque individu par une estimation de la densité de population dans son voisinage. Sa complexité quadratique peut néanmoins s'avérer coûteuse au regard des autres opérateurs, généralement linéaires. Une deuxième approche, le sharing clusterisé [Yin 93], pénalise les individus regroupés au sein de groupes (clusters) en fonction de leur distance au centre du cluster. A la notion de distance s'ajoute donc la définition de barycentre d'un cluster. Cette variante du sharing est moins gourmande en calculs et s'exécute avec une complexité de $O(N \log N)$.

2.2.5 Remplacement de la population

Afin de déterminer quels seront les individus conservés à la génération suivante, parents et enfants peuvent « s'affronter » dans un tournoi : le mieux évalué entre un parent et son enfant remporte le tournoi et est conservé dans la population.

On parle d'élitisme lorsqu'on conserve systématiquement les k individus les mieux évalués ($k \leq N$) d'une génération à la suivante. De manière générale, il est contre-productif d'éliminer systématiquement les individus les plus mal évalués, potentiellement porteurs de gènes « utiles » à l'élaboration d'une solution satisfaisante. Conserver de « mauvais »

individus contribue donc à la diversification de la population et empêche l'algorithme de converger prématurément.

2.3 Optimisation par essaims particulaires

L'OEP [Kennedy 95] s'inspire du déplacement des groupes d'oiseaux ou des bancs de poissons. Les solutions candidates (appelées particules) se déplacent dans l'espace de recherche en modifiant leur position et leur vitesse selon des règles simples : le mouvement de chaque particule est influencé par sa meilleure position connue et par la meilleure solution connue de l'essaim dans l'espace de recherche (algorithme 3). Les positions des particules améliorées localement permettent à l'essaim de se déplacer globalement vers de bonnes solutions.

Algorithme 3 Optimisation par essaims particulaires

```
Initialiser l'essaim (positions et vitesses des particules)
Evaluer les particules
répéter
    pour chaque particule de l'essaim faire
        Mettre à jour la vitesse de la particule
        Mettre à jour la position de la particule
        Mettre à jour la meilleure position connue de la particule
        Mettre à jour la meilleure position connue de l'essaim
    fin pour
jusqu'à critère d'arrêt vérifié
renvoyer meilleure position connue de l'essaim
```

2.3.1 Formules de mise à jour

La vitesse d'une particule $V^{(k+1)}$ à la génération $k+1$ est fonction de sa position courante $X^{(k)}$, sa vitesse courante $V^{(k)}$, sa meilleure position connue X_l et la meilleure position connue X_g de l'essaim :

$$V^{(k+1)} = \omega V^{(k)} + \alpha^\top (X_l - X^{(k)}) + \beta^\top (X_g - X^{(k)}) \tag{2.6}$$

où ω est un hyperparamètre (facteur d'inertie) décidé par l'utilisateur, et α et β sont des vecteurs tirés dans $[0, 1]^n$ avec une probabilité uniforme. La position de la particule est ensuite simplement mise à jour :

$$X^{(k+1)} = X^{(k)} + V^{(k+1)} \tag{2.7}$$

2.3.2 Voisinage

L'OEP standard est dite « global best », car chaque particule est attirée par la meilleure position connue X_g de l'essaim. Ce phénomène de biais est souvent la source d'une convergence prématurée vers des minima locaux. Une alternative, dite « local best », consiste à guider une particule vers la meilleure position connue d'un sous-ensemble de particules voisines (par exemple, les k plus proches).

2.4 Algorithme à évolution différentielle

L'algorithme à ED est parmi les AE les plus simples et les plus performants [Storn 97]. Initialement conçue pour les problèmes sans contraintes à variables continues, l'ED a été étendue aux problèmes mixtes et aux problèmes contraints. Sa robustesse et son nombre faible d'hyperparamètres en font un outil de choix pour la résolution de problèmes difficiles ; l'algorithme s'est notamment illustré dans des problèmes d'entraînement de réseaux de neurones [Slowik 08], de conception aérodynamique [Rogalsky 00], de décision multicritère, d'approximation polynomiale et d'ordonnancement de tâches.

Contrairement aux AG dont les nouveaux individus proviennent des opérateurs de croisement et de mutation, l'ED n'est pas inspirée de l'évolution naturelle mais génère de nouveaux individus via des opérations géométriques. Basée sur le concept de différence de vecteurs, l'ED combine avec une certaine probabilité les composantes d'individus existants pour former de nouveaux individus (algorithme 4).

Algorithme 4 Evolution différentielle

fonction EvolutionDifférentielle(f : fonction objectif, NP : taille de la population, W : facteur d'amplitude, CR : taux de croisement)
  $P \leftarrow$ population initiale aléatoire
  répéter
    $P' \leftarrow \varnothing$ ▷ population temporaire
    pour $\boldsymbol{x} \in P$ **faire**
      $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \leftarrow$ ChoixParents($\boldsymbol{x}$, P)
      $\boldsymbol{y} \leftarrow$ Croisement($\boldsymbol{x}$, $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$, W, CR) ▷ génère un nouvel individu
      si $f(\boldsymbol{y}) < f(\boldsymbol{x})$ **alors**
        $P' \leftarrow P' \cup \{\boldsymbol{y}\}$ ▷ $\boldsymbol{y}$ remplace $\boldsymbol{x}$
      sinon
        $P' \leftarrow P' \cup \{\boldsymbol{x}\}$ ▷ $\boldsymbol{x}$ est conservé
      fin si
    fin pour
    $P \leftarrow P'$ ▷ la population temporaire remplace la population
  jusqu'à critère d'arrêt vérifié
renvoyer meilleur individu de P
fin fonction

2.4.1 Opérateur de croisement quaternaire

Notons NP la taille de la population, $W > 0$ le facteur d'amplitude et $CR \in [0, 1]$ le taux de croisement. A chaque génération, NP nouveaux individus sont générés : pour chaque individu $\boldsymbol{x} = (x_1, \ldots, x_n)$, trois autres individus $\boldsymbol{u} = (u_1, \ldots, u_n)$ (appelé individu de base), $\boldsymbol{v} = (v_1, \ldots, v_n)$ et $\boldsymbol{w} = (w_1, \ldots, w_n)$, tous différents et différents de $\boldsymbol{x}$, sont aléatoirement choisis dans la population. Les composantes y_i ($i \in \{1, \ldots, n\}$) du nouvel individu $\boldsymbol{y} = (y_1, \ldots, y_n)$ sont alors calculées de la manière suivante :

$$y_i = \begin{cases} u_i + W \times (v_i - w_i) & \text{si } i = R \text{ ou } r_i < CR \\ x_i & \text{sinon} \end{cases} \tag{2.8}$$

où r_i est tiré uniformément dans $[0, 1]$ et R est un indice aléatoire tiré uniformément dans $\{1, \ldots, n\}$ pour tout $\boldsymbol{x}$, garantissant qu'au moins une composante de $\boldsymbol{y}$ diffère de celle de $\boldsymbol{x}$. $\boldsymbol{y}$ remplace alors $\boldsymbol{x}$ dans la population s'il améliore la fonction objectif. L'opérateur de sélection est donc réduit à une sélection élitiste.

La figure 2.5 illustre le croisement entre les individus $\boldsymbol{x}$, $\boldsymbol{u}$ (individu de base), $\boldsymbol{v}$ et $\boldsymbol{w}$ en deux dimensions. La fonction objectif est représentée en courbes de niveaux. La différence $\boldsymbol{v} - \boldsymbol{w}$ donne la direction de déplacement (une approximation de la direction opposée au gradient) dans laquelle $\boldsymbol{u}$ est translaté.

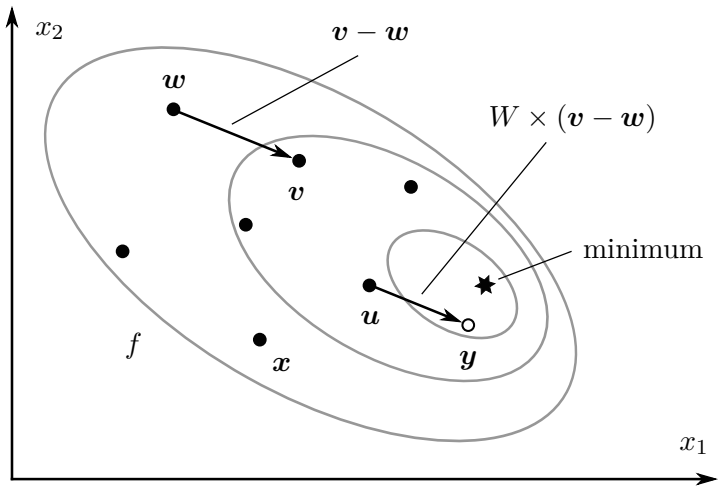


FIGURE 2.5 – Croisement quaternaire de l'évolution différentielle

2.4.2 Choix de l'individu de base

Dans un AG, la probabilité de sélection d'un individu est généralement proportionnelle à son évaluation. Dans l'ED, la sélection de l'individu de base $\boldsymbol{u}$ est équiprobable parmi tous les individus de la population. Deux variantes ont été proposées par [Price 06] pour garantir que tous les individus de la population courante jouent le rôle d'individu de base

une et une seule fois par génération. Les indices des individus de base dans la population peuvent être obtenus :

1. par permutation aléatoire dans $\{1, \ldots, NP\}$;
2. en tirant un offset dans $\{1, \ldots, NP-1\}$ avec une probabilité uniforme et en l'ajoutant (modulo NP) à l'indice de chaque individu $\boldsymbol{x}$.

2.4.3 Contraintes de bornes

Il arrive régulièrement que certaines composantes du nouvel individu $\boldsymbol{y}$ se trouvent en dehors de l'espace de recherche D. Deux alternatives sont alors possibles :

- pénaliser la fonction objectif : un terme constant ou dépendant du nombre et de la magnitude des violations est ajouté à la fonction objectif. Lorsque les individus générés ont une forte tendance à violer les contraintes de bornes, cette approche présente un risque de convergence lente ;
- générer une nouvelle composante à l'intérieur du domaine : saturer la borne du domaine, réinitialiser aléatoirement la composante dans le domaine entier ou sur le segment liant la composante de base u_i à la borne du domaine D_i (méthode du rebond [Price 06]) :

$$y_i = \begin{cases} u_i + \omega(\overline{D_i} - u_i) & \text{si } y_i > \overline{D_i} \\ u_i + \omega(\underline{D_i} - u_i) & \text{si } y_i < \underline{D_i} \end{cases} \tag{2.9}$$

  où ω est tiré avec une probabilité uniforme dans $[0, 1]$.

2.4.4 Gestion directe des contraintes

La gestion directe des contraintes consiste à séparer les évaluations de la fonction objectif et des contraintes du problème : à chaque individu est associé un tableau contenant la valeur de la fonction objectif et l'évaluation de chacune des contraintes. Le choix de l'individu de base $\boldsymbol{u}$ satisfait alors une des règles suivantes :

- $\boldsymbol{u}$ appartient au domaine réalisable, contrairement à $\boldsymbol{x}$;
- $\boldsymbol{u}$ et $\boldsymbol{x}$ appartiennent au domaine réalisable, et $f(\boldsymbol{u}) < f(\boldsymbol{x})$;
- $\boldsymbol{u}$ et $\boldsymbol{x}$ n'appartiennent pas au domaine réalisable, mais $\boldsymbol{u}$ ne viole aucune contrainte davantage que $\boldsymbol{x}$.

2.5 Critères d'arrêt

On distingue deux grandes catégories de critères d'arrêt :

- un critère *statique* est généralement basé sur les ressources matérielles disponibles (temps CPU, nombre d'itérations ou d'évaluations de la fonction objectif) et connues a priori ;

– un critère *dynamique* fait référence à la qualité de la solution (suffisamment proche d'un optimum connu a priori) ou à la fin de convergence (nombre d'itérations consécutives sans améliorer la meilleure solution connue).

Chapitre 3

Analyse par intervalles

Sommaire

3.1 Calcul par intervalles 28
3.1.1 Contrôle des arrondis 28
3.1.2 Arithmétique d'intervalles 29
3.2 Fonctions d'inclusion 31
3.2.1 Dépendance 32
3.2.2 Formes du deuxième ordre 33
3.2.3 Extension par monotonie 36
3.2.4 Arithmétique affine 37
3.3 Algorithme de branch and bound par intervalles 37
3.3.1 Branch and bound 38
3.3.2 Branch and bound par intervalles 39
3.3.3 Heuristiques 40
3.3.4 Techniques d'accélération 41
3.4 Différentiation automatique 43
3.4.1 Mode direct 44
3.4.2 Mode adjoint 44
3.5 Conclusion 45

L'analyse par intervalles est la branche de l'analyse numérique dédiée à l'encadrement des erreurs d'arrondis. Les méthodes d'intervalles permettent de calculer un minorant et un majorant rigoureux d'une fonction sur un intervalle, même en présence d'arrondis. Elles constituent donc une approche priviliégiée pour l'optimisation globale fiable.

Le calcul par intervalles est décrit dans la section 3.1. Le concept de fonction d'inclusion est introduit dans la section 3.2. Les algorithmes de branch and bound par intervalles, dédiés à l'optimisation de problèmes continus, sont présentés dans la section 3.3. La section 3.4 mentionne les différents modes de différentiation automatique.

3.1 Calcul par intervalles

L'arithmétique flottante (AF) est une méthode de représentation approchée des nombres réels sur des machines discrètes [Goldberg 91]. Un nombre flottant x est représenté par son signe, sa mantisse (la partie fractionnaire de x) et son exposant. Les floating point units ou unités de calcul en virgule flottante (FPU) présentes dans les processeurs manipulent des nombres dont la mantisse est de taille fixe, ce qui conduit à des erreurs d'approximation lorsque les réels ne sont pas représentables exactement en machine. Par exemple, la constante π arrondie à 3 décimales est soit 3.141, soit 3.142, et la valeur exacte se situe quelque part dans l'intervalle $[3.141, 3.142]$.

La norme IEEE-754 fixant la représentation des réels et le comportement des opérations de base en virgule flottante a depuis 1985 été adoptée par la plupart des FPU. Elle spécifie quatre modes d'arrondi : au plus proche, vers zéro, vers $+\infty$ et vers $-\infty$. Le format IEEE-754 double précision (64 bits) fournit une précision avancée (15 à 17 décimales). Néanmoins, l'accumulation de nombreux arrondis sur des problèmes numériquement instables est la source de résultats erronés (voir exemple 3).

Exemple 3 (Accumulation des erreurs d'arrondis) *Une accumulation catastrophique d'erreurs d'arrondis a été illustrée par [Rump 88]. Considérons la fonction :*

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + \frac{x}{2y} \tag{3.1}$$

En évaluant $f(77617, 33096)$*, on obtient* 1.172603 *en simple précision et* 1.1726039400531 *en double précision. Pourtant, la valeur exacte est* $-\frac{54767}{66192} = -0.827396$.

3.1.1 Contrôle des arrondis

La thèse de doctorat de [Moore 66] a posé les fondements du calcul par intervalles : l'idée de Moore est de représenter chaque résultat intermédiaire d'un calcul par un intervalle contenant le résultat exact. Un réel x représentable en machine est alors remplacé par l'intervalle dégénéré $[x, x]$, tandis qu'un réel non représentable exactement en virgule flottante (par exemple 0.1) est encadré rigoureusement par un intervalle.

L'arithmétique d'intervalles (AI) étend l'arithmétique réelle aux intervalles de manière rigoureuse. Les fonctions élémentaires ($+$, $-$, $\times$, $/$, log, exp, etc.) sont implémentées en arrondissant les calculs vers l'extérieur : la borne gauche du résultat est calculée avec un mode d'arrondi vers $-\infty$ et la borne droite avec un mode d'arrondi vers $+\infty$. La valeur exacte est alors numériquement garantie d'appartenir à l'intervalle obtenu par AI.

De nombreuses bibliothèques logicielles implémentent l'AI : Profil/BIAS [Knüppel 94] (développée en C++ à l'Université de Technologie de Hamburg), Gaol [Goualard 03] (implémentation C++ d'opérateurs de programmation par contraintes sur intervalles), Boost [Brönnimann 06] (template C++), MPFI [Revol 02] (bibliothèque multi-précision en C et C++), Sun [Microsystems 01] (Fortran 95 et C++) et Filib [Lerch 01]. Nous avons récemment implémenté une bibliothèque de calcul par intervalles dans le langage fonctionnel

OCaml [Alliot 12b]. Par souci de performance, des routines de bas niveau (C et assembleur) permettent un contrôle fin des arrondis.

3.1.2 Arithmétique d'intervalles

Nous adoptons les notations suivantes :
- $\mathbb{R}$ dénote l'ensemble des réels ;
- $\mathbb{F}$ dénote l'ensemble des flottants ;
- un intervalle $X = [\underline{X}, \overline{X}]$ à bornes flottantes définit l'ensemble :

$$X := \{x \in \mathbb{R} \mid \underline{X} \leq x \leq \overline{X}\} \tag{3.2}$$

- un intervalle est dit dégénéré lorsque $\underline{X} = \overline{X}$;
- $\mathbb{I}$ représente l'ensemble des intervalles à bornes flottantes :

$$\mathbb{I} := \{[\underline{X}, \overline{X}] \mid (\underline{X}, \overline{X}) \in \mathbb{F}^2 \wedge \underline{X} \leq \overline{X}\} \tag{3.3}$$

- pour un ensemble $D \subset \mathbb{R}$, $\mathbb{I}(D)$ dénote l'ensemble des intervalles inclus dans D. La définition s'étend au cas multivarié ;
- $int(X)$ désigne l'intérieur d'un intervalle X non dégénéré, c'est-à-dire l'ensemble :

$$int(X) := \{x \in \mathbb{R} \mid \underline{X} < x < \overline{X}\} \tag{3.4}$$

- $m(X) := \frac{1}{2}(\underline{X} + \overline{X})$ désigne le milieu de l'intervalle X ;
- $w(X) := \overline{X} - \underline{X}$ désigne la largeur de l'intervalle X ;
- une boîte $\boldsymbol{X} = (X_1, \ldots, X_n)$ est un produit cartésien d'intervalles ;
- $m(\boldsymbol{X}) := (m(X_1), \ldots, m(X_n))$ désigne le milieu de la boîte $\boldsymbol{X}$;
- $w(\boldsymbol{X}) = \max_{i=1\ldots n} w(X_i)$ désigne la largeur de la boîte $\boldsymbol{X}$;
- $\square(X, Y)$ désigne l'enveloppe convexe de X et Y, soit le plus petit intervalle de $\mathbb{I}$ contenant X et Y, et calculable par la machine.

Par la suite, nous notons en majuscules les quantités intervalles et en gras les vecteurs. Un intervalle est donc noté X, une boîte $\boldsymbol{X}$ et un vecteur de réels $\boldsymbol{x}$.

L'extension à l'AI des opérateurs binaires $\diamond \in \{+, -, \times, /\}$ fournit le plus petit intervalle contenant l'image directe :

$$X \diamond Y = \square\{x \diamond y \mid x \in X \wedge y \in Y\} \tag{3.5}$$

On peut calculer explicitement les bornes gauche et droite du résultat en fonction des bornes des opérandes :

$$\begin{aligned}
[a,b] + [c,d] &= [a+c, b+d] \\
[a,b] - [c,d] &= [a-d, b+c] \\
[a,b] \times [c,d] &= [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)] \\
\frac{1}{[a,b]} &= [\frac{1}{b}, \frac{1}{a}] \text{ si } 0 \notin [a,b] \\
\frac{[a,b]}{[c,d]} &= [a,b] \times \frac{1}{[c,d]} \text{ si } 0 \notin [c,d]
\end{aligned} \tag{3.6}$$

Rappelons que les calculs sur intervalles (par exemple l'addition notée de manière abusive $[a,b]+[c,d]=[a+c,b+d]$) doivent être effectués en arrondissant la borne gauche vers $-\infty$ et la borne droite vers $+\infty$.

La plupart des fonctions élémentaires unaires sont étendues aux intervalles en utilisant leurs propriétés de monotonie :

$$\begin{aligned} \exp([a,b]) &= [\exp(a),\exp(b)] \\ \log([a,b]) &= [\log(a),\log(b)] \qquad (a>0) \\ \sqrt{[a,b]} &= [\sqrt{a},\sqrt{b}] \qquad (a\geq 0) \end{aligned} \tag{3.7}$$

Les fonctions non monotones sur l'intervalle considéré (puissances paires, fonctions trigonométriques) sont néanmoins monotones par morceaux et font l'objet d'une étude au cas par cas.

L'AI possède des propriétés plus faibles que l'arithmétique réelle : la soustraction n'est pas l'opération inverse de l'addition sur $\mathbb{I}$, de même que la division n'est pas l'opération inverse de la multiplication (exemple 4). En outre, la propriété de distributivité de la multiplication par rapport à l'addition n'est pas vérifiée sur $\mathbb{I}$; seule la propriété plus faible de sous-distributivité est valable :

$$\forall (X,Y,Z)\in\mathbb{I}^3,\quad X(Y+Z)\subset XY+XZ \tag{3.8}$$

Exemple 4 (Non-inversibilité de l'addition et de la multiplication) *Soient* $X=[2,4]$ *et* $Y=[3,5]$*. Alors :*

$$\begin{aligned} W=X+Y=[2,4]+[3,5]=[5,9] \qquad & W-X=[5,9]-[2,4]=[1,7]\supsetneq Y \\ Z=X\times Y=[2,4]\times[3,5]=[6,20] \qquad & \frac{Z}{X}=\frac{[6,20]}{[2,4]}=[1.5,10]\supsetneq Y \end{aligned} \tag{3.9}$$

L'AI étendue [Hanson 68, Kahan 68, Hansen 92] généralise l'AI aux intervalles à bornes infinies, ce qui permet notamment de décrire la division par un intervalle contenant zéro :

$$\frac{1}{[a,b]}=\begin{cases} \varnothing & \text{si } a=b=0 \\ [\frac{1}{b},\frac{1}{a}] & \text{si } 0<a \text{ ou } b<0 \\ [\frac{1}{b},+\infty] & \text{si } a=0 \\ [-\infty,\frac{1}{a}] & \text{si } b=0 \\ [-\infty,\frac{1}{a}]\cup[\frac{1}{b},+\infty] & \text{si } a<0<b \end{cases} \tag{3.10}$$

ou les calculs à bornes infinies :

$$\sqrt{[4,+\infty]}=[2,+\infty] \tag{3.11}$$

3.2 Fonctions d'inclusion

Les propriétés de conservativité du calcul par intervalles permettent de construire des extensions aux intervalles (définition 12) des fonctions réelles décomposables (définition 10).

Définition 10 (Fonction décomposable) *Une fonction est* décomposable *(factorable function) lorsqu'elle s'exprime récursivement comme une composition de termes élémentaires (opérateurs, fonctions élémentaires ou variables).*

Définition 11 (Image directe) *Soient $f : D \subset \mathbb{R}^n \to \mathbb{R}$ et $\boldsymbol{X} \in \mathbb{I}(D)$. On note $f(\boldsymbol{X})$ l'*image directe *de f sur $\boldsymbol{X}$:*

$$f(\boldsymbol{X}) := \{f(\boldsymbol{x}) \mid \boldsymbol{x} \in \boldsymbol{X}\} \tag{3.12}$$

Définition 12 (Fonction d'inclusion) *Soient $f : D \subset \mathbb{R}^n \to \mathbb{R}$ et $F : \mathbb{I}(D) \to \mathbb{I}$. On dit que F est une* fonction d'inclusion *(ou* extension aux intervalles*) de f si :*

$$\begin{array}{rll} \forall \boldsymbol{X} \subset \mathbb{I}(D), & f(\boldsymbol{X}) \subset F(\boldsymbol{X}) & \textit{(Conservativité)} \\ \forall (\boldsymbol{X}, \boldsymbol{Y}) \in \mathbb{I}(D)^2, & \boldsymbol{X} \subset \boldsymbol{Y} \implies F(\boldsymbol{X}) \subset F(\boldsymbol{Y}) & \textit{(Isotonicité d'inclusion)} \end{array} \tag{3.13}$$

Les fonctions d'inclusion d'une fonction réelle ne sont pas uniques : on peut généralement construire des fonctions d'inclusion dont les ordres de convergence (définition 13) sont différents, c'est-à-dire dont la surestimation décroît à des vitesses différentes avec la taille de l'intervalle. L'évaluation d'expressions syntaxiquement équivalentes en arithmétique réelle peut fournir des encadrements plus ou moins précis en AI. Ce point est discuté dans la section 3.2.1.

Définition 13 (Ordre de convergence) *Soient $f : D \subset \mathbb{R}^n \to \mathbb{R}$ et F une fonction d'inclusion de f. On dit que F est d'*ordre de convergence α *($\alpha > 0$) lorsque, pour tout $\boldsymbol{X} \in \mathbb{I}(D)$:*

$$w(F(\boldsymbol{X})) - w(f(\boldsymbol{X})) = O(w(\boldsymbol{X})^\alpha) \tag{3.14}$$

$w(F(\boldsymbol{X})) - w(f(\boldsymbol{X}))$ représente l'erreur de surestimation de l'image de f sur $\boldsymbol{X}$.

L'extension aux intervalles la plus simple à construire pour une fonction décomposable est la fonction d'inclusion naturelle (définition 14). Elle a un ordre de convergence linéaire, c'est-à-dire que l'erreur de surestimation varie linéairement avec la taille des domaines des variables.

Définition 14 (Fonction d'inclusion naturelle) *Soit $f : D \subset \mathbb{R}^n \to \mathbb{R}$. Sa* fonction d'inclusion naturelle *$F_N : \mathbb{I}(D) \to \mathbb{I}$ est obtenue en remplaçant dans l'expression de f chacune des variables par son domaine et chacune des opérations élémentaires par son extension aux intervalles.*

3.2.1 Dépendance

La raison principale pour laquelle l'AI surestime, parfois de manière grossière, l'image d'une fonction est connue sous le nom de problème de dépendance : les différentes occurrences des variables sont décorrélées et considérées comme des variables différentes. L'encadrement calculé dépend donc de l'expression syntaxique de la fonction ; ceci explique la propriété de sous-distributivité sur $\mathbb{I}$ (voir 3.8).

Par l'exemple, l'intervalle $X = [a, b]$ soustrait à lui-même donne $Y = X - X = [a-b, b-a]$. Lorsque $a < b$, Y n'est pas réduit à l'intervalle dégénéré $[0, 0]$ (la valeur exacte 0 appartient évidemment à Y). Ici, l'erreur de surestimation (la largeur de l'intervalle Y) est $2(b-a)$, soit deux fois la largeur de l'intervalle X. L'image calculée est en fait $\{x_1 - x_2 \mid x_1 \in X, x_2 \in X\}$. La dépendance (l'égalité) entre x_1 et x_2 n'est pas connue de l'AI, ce qui ajoute un degré de liberté dans la construction de l'ensemble $f(X)$.

Dans son théorème fondamental de l'AI, [Moore 66] a prouvé que sous certaines conditions, l'AI ne surestime pas l'image de f (théorème 5).

Théorème 5 (Image optimale [Moore 66]) *Lorsqu'une fonction f est continue sur une boîte $\boldsymbol{X}$, sa fonction d'inclusion naturelle fournit l'image optimale, aux arrondis près, lorsque les variables ont au plus une occurrence dans l'expression de f :*

$$F_N(\boldsymbol{X}) = f(\boldsymbol{X}) \tag{3.15}$$

Le contre-exemple suivant met en évidence l'importance de l'hypothèse de continuité du théorème de Moore.

Exemple 5 (Fonction non continue sur un intervalle) *Soient $f(x) = (\frac{1}{x})^2$ et $X = [-1, 1]$. L'image directe de X par f est $f(X) = [1, +\infty]$. Malgré la présence d'une seule occurrence de la variable x dans l'expression de f, l'AI calcule $\frac{1}{X} = [-\infty, +\infty]$, puis $(\frac{1}{X})^2 = [-\infty, +\infty]^2 = [0, +\infty] \supset f(X)$. La surestimation est provoquée par la discontinuité de f sur X.*

Une méthode immédiate pour réduire la surestimation de l'AI est de réécrire une expression dans laquelles les variables apparaissent plusieurs fois, pour tenter de réduire leur nombre d'occurrences à un (exemple 6).

Exemple 6 (Réécriture d'une expression) *Soit $f(x) = \frac{x-y}{x+y}$, $x \in X = [6, 8]$ et $y \in Y = [2, 4]$. L'image directe de f est $f(X, Y) = [\frac{1}{5}, \frac{3}{5}]$. L'extension naturelle de f fournit l'encadrement :*

$$F_N(X, Y) = \frac{X - Y}{X + Y} = \frac{[6, 8] - [2, 4]}{[6, 8] + [2, 4]} = \frac{[2, 6]}{[8, 12]} = [\frac{1}{6}, \frac{3}{4}] \supset f(X, Y) \tag{3.16}$$

Réarrangeons l'expression de f, que nous notons g :

$$g(x, y) = \frac{x - y}{x + y} = \frac{x + y - 2y}{x + y} = 1 - \frac{2y}{x + y} = 1 - \frac{2}{1 + \frac{x}{y}} \tag{3.17}$$

Alors :

$$G_N(X,Y) = 1 - \frac{2}{1+\frac{X}{Y}} = 1 - \frac{2}{1+\frac{[6,8]}{[2,4]}} = [\frac{1}{5}, \frac{3}{5}] = g(X,Y) \tag{3.18}$$

Réécrire l'expression de f de manière à obtenir une seule occurrence de x et y garantit d'obtenir, aux arrondis près, l'image exacte.

Généralement, les fonctions traitées dans les problèmes d'optimisation globale sont complexes et les variables apparaissent plusieurs fois dans les expressions de la fonction objectif et des contraintes. Il n'est alors pas toujours possible de factoriser les expressions syntaxiques et les inclusions calculées par l'extension naturelle sont généralement grossières, même lorsque les intervalles sont de largeur faible. Ceci rend délicate l'utilisation des méthodes d'intervalles pour la résolution de problèmes difficiles. Néanmoins, de nombreux auteurs ont cherché à s'affranchir du problème de dépendance inhérent au calcul par intervalles.

La première méthode pour réduire la surestimation est d'évaluer la fonction d'inclusion sur des intervalles plus étroits. En effet, la surestimation calculée par une extension aux intervalles – isotone pour l'inclusion – décroît avec la largeur des domaines des variables. Parfois, évaluer une fonction d'inclusion F sur une partition $\{X_i\}_{i=1\ldots n}$ de X améliore strictement l'encadrement de l'image de f. Ce principe est exploité dans les algorithmes de branch and bound (séparation et évaluation), dans lesquels alternent des phases d'évaluation par AI et des phases de partitionnement du domaine (voir section 3.3).

Une deuxième méthode requiert l'utilisation de fonctions d'inclusion d'ordre supérieur à 1. Rappelons que pour une fonction d'inclusion d'ordre α, la surestimation de l'image de f sur X décroît avec $w(X)^\alpha$. L'encadrement calculé par des extensions aux intervalles d'ordre supérieur tend à être de meilleure qualité sur des intervalles de faible diamètre. Le calcul de formes d'ordre supérieur est décrit dans la sous-section 3.2.2.

Une troisième méthode consiste à vérifier si la fonction est monotone par rapport à certaines de ses variables sur la boîte courante, et à réduire le calcul de l'image de f à des évaluations ponctuelles aux bornes de la boîte. Indubitablement la plus puissante, la détection de monotonie s'affranchit totalement du problème de dépendance associé à ces variables, mais n'est pas toujours exploitable sur des boîtes de diamètre important. La méthode est décrite dans la sous-section 3.2.3.

3.2.2 Formes du deuxième ordre

Nous introduisons dans cette section une famille d'extensions aux intervalles possédant la propriété de convergence quadratique (voir définition 13).

Soient $f : D \subset \mathbb{R} \to \mathbb{R}$, $X \in \mathbb{I}(D)$ et $c \in X$ (par exemple $c = m(X)$). Le théorème de Taylor indique que, lorsque f est dérivable en c jusqu'à l'ordre $m-1$, on peut écrire pour

tout $x \in X$:

$$\begin{aligned} f(x) &= \sum_{k=0}^{m-1} \frac{f^{(k)}(c)}{k!}(x-c)^k + \frac{f^{(m)}(\xi)}{m!}(x-c)^m \\ &\in \sum_{k=0}^{m-1} \frac{f^{(k)}(c)}{k!}(x-c)^k + \frac{F^{(m)}(X)}{m!}(x-c)^m \end{aligned} \tag{3.19}$$

où ξ est strictement compris entre c et x, et $F^{(m)}$ est une fonction d'inclusion de $f^{(m)}$. L'inclusion vérifiée pour tout x permet de définir une extension aux intervalles, dite de Taylor. Les formes de Taylor essentiellement utilisées en analyse par intervalles sont les approximations linéaires ($m = 1$) et quadratiques ($m = 2$). Dans la suite, nous nous concentrons sur la forme linéaire :

$$F_T(X, c) := f(c) + F'(X)(X - c) \tag{3.20}$$

aussi connue sous le nom de *mean value form* (en référence au *mean value theorem*, le théorème des accroissements finis [Jeffreys 99]). F_T est de convergence quadratique (théorème 6) et isotone pour l'inclusion (théorème 7) lorsque $c = m(X)$.

Définition 15 (Fonction lipschitzienne [Moore 66]) *Soient $f : D \subset \mathbb{R}^n \to \mathbb{R}$ et $F : \mathbb{I}(D) \to \mathbb{I}$ une fonction d'inclusion de f. F est une* fonction lipschitzienne *s'il existe un réel positif K tel que :*

$$\forall \boldsymbol{X} \in \mathbb{I}(D), \quad w(F(\boldsymbol{X})) \leq K w(\boldsymbol{X}) \tag{3.21}$$

Théorème 6 (Convergence quadratique [Krawczyk 82]) *Si F' est lipschitzienne, la forme de Taylor $F_T(X, m(X))$ a une convergence quadratique.*

Théorème 7 (Isotonie d'inclusion [Caprani 80]) *Si F' est isotone pour l'inclusion, alors la forme de Taylor $F_T(X, m(X))$ est isotone pour l'inclusion.*

Remarque 2 (Erreurs d'arrondis) *En pratique, l'évaluation $f(c)$ est sujette à des erreurs d'arrondis et doit être remplacée par $F(c) := F([c, c])$.*

Visuellement, un cône simple ou double (en forme de papillon) contenant l'ensemble des dérivées possibles de f sur X est déployé depuis le point $(c, F(c))$ et englobe la courbe de f (figure 3.1).

La forme de Taylor s'étend facilement à une fonction multivariée $f : D \subset \mathbb{R}^n \to \mathbb{R}$:

$$f(\boldsymbol{X}) \subset F(\boldsymbol{c}) + \sum_{i=1}^{n} \frac{\partial F}{\partial x_i}(\boldsymbol{X}) \cdot (X_i - x_i) \tag{3.22}$$

où $\boldsymbol{X} = (X_1, \ldots, X_n) \in \mathbb{I}(D)$, $\boldsymbol{c} = (c_1, \ldots, c_n) \in \boldsymbol{X}$ et $\frac{\partial F}{\partial x_i}$ est une fonction d'inclusion de la $i$ème dérivée partielle de f. Les n dérivées partielles peuvent être évaluées simultanément en utilisant des techniques peu coûteuses de différentiation automatique (voir section 3.4). [Hansen 68] a proposé une variante récursive dans laquelle le développement en série de Taylor est réalisé variable après variable.

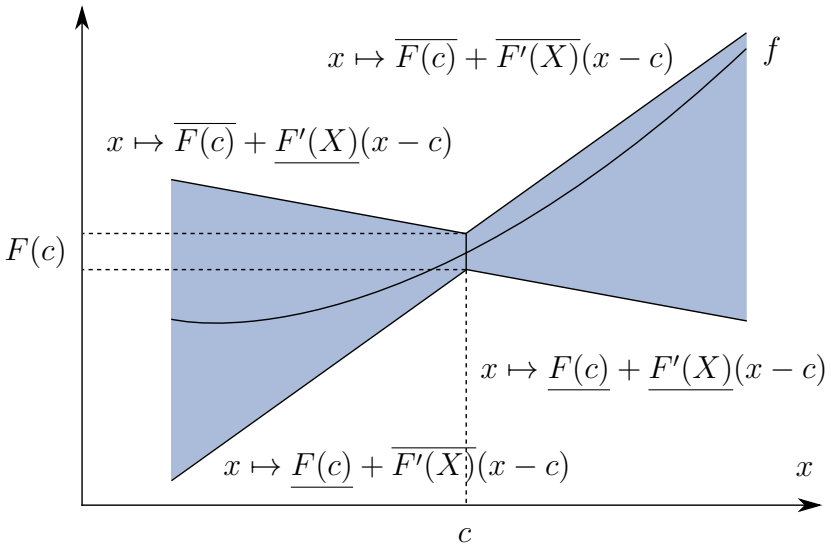


FIGURE 3.1 – Forme de Taylor

[Baumann 88] a explicité l'expression analytique du centre optimal c_B^- (respectivement c_B^+) qui maximise le minorant (respectivement minimise le majorant) pour la forme de Taylor linéaire (figure 3.2) :

$$c_B^- := \begin{cases} \underline{X} & \text{si } 0 \leq F'(X) \\ \overline{X} & \text{si } 0 \geq F'(X) \\ \frac{U\underline{X} - L\overline{X}}{U-L} & \text{sinon} \end{cases} \qquad c_B^+ := \begin{cases} \overline{X} & \text{si } 0 \leq F'(X) \\ \underline{X} & \text{si } 0 \geq F'(X) \\ \frac{U\overline{X} - L\underline{X}}{U-L} & \text{sinon} \end{cases} \tag{3.23}$$

où $[L, U] := F'(X)$. La preuve est donnée dans l'annexe A.1.1.

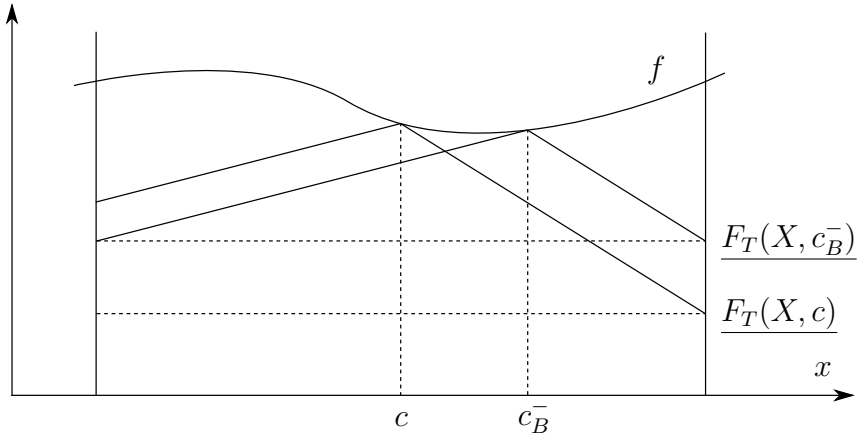


FIGURE 3.2 – Comparaison des minorants donnés par $F_T(X, c)$ et $F_T(X, c_B^-)$

Les formes alternatives centrée, bicentrée et kite, d'ordre de convergence 2, sont mentionnées dans les annexes A.1.2 et A.1.3.

3.2.3 Extension par monotonie

L'extension par monotonie consiste à exploiter la monotonie locale de la fonction par rapport à certaines de ses variables (définition 16) afin de s'affranchir partiellement du problème de dépendance et de calculer des encadrements plus précis que par les formes de premier et deuxième ordres.

Définition 16 (Variable localement monotone) *Soient $f : D \subset \mathbb{R}^n \to \mathbb{R}$, F une extension aux intervalles de f et $\boldsymbol{X} \in \mathbb{I}(D)$ une boîte. On dit que f est* localement croissante *(respectivement* décroissante*) par rapport à la variable x_i lorsque $\frac{\partial f}{\partial x_i}(\boldsymbol{X})$ est positive (respectivement négative). Par extension, on appellera* variable localement monotone *une variable par rapport à laquelle f est monotone sur $\boldsymbol{X}$.*

Exemple 7 (Monotonie locale) *Soient $f(x) = x + 2\cos(x)$ et $X = [3, 6]$. Calculons la dérivée de f par rapport à x : $f'(x) = 1 - 2\sin(x)$. Son évaluation naturelle sur X produit (avec une précision de 3 décimales) $F'_N(X) = 1 - 2\sin([3, 6]) = 1 - 2[-1, 0.142] = [0.717, 3] \geq 0$. x est donc localement croissante sur X.*

L'extension par monotonie F_M (définition 17) calcule un encadrement de l'image de f plus précis que F_N lorsque certaines variables multi-occurrentes sont détectées monotones sur la boîte $\boldsymbol{X}$:

$$f(\boldsymbol{X}) \subset F_M(\boldsymbol{X}) \subset F_N(\boldsymbol{X}) \tag{3.24}$$

Définition 17 (Extension par monotonie) *Soit $f : D \subset \mathbb{R}^n \to \mathbb{R}$ et $\boldsymbol{X} = (X_1, \ldots, X_n)$ une boîte. On note $\boldsymbol{X}^- = (X_1^-, \ldots, X_n^-)$ et $\boldsymbol{X}^+ = (X_1^+, \ldots, X_n^+)$ les boîtes définies pour tout $i \in \{1, \ldots, n\}$ par :*

$$X_i^- := \begin{cases} \underline{X_i} & \text{si } x_i \text{ croissante} \\ \overline{X_i} & \text{si } x_i \text{ décroissante} \\ X_i & \text{sinon} \end{cases} \qquad X_i^+ := \begin{cases} \overline{X_i} & \text{si } x_i \text{ croissante} \\ \underline{X_i} & \text{si } x_i \text{ décroissante} \\ X_i & \text{sinon} \end{cases} \tag{3.25}$$

*L'*extension par monotonie F_M *de f est définie par :*

$$F_M(\boldsymbol{X}) := [\underline{F_N(\boldsymbol{X}^-)}, \overline{F_N(\boldsymbol{X}^+)}] \tag{3.26}$$

Puisque l'extension par monotonie remplace toutes les variables monotones par une de leurs bornes, le problème de dépendance associé à ces variables disparaît dans l'évaluation de F_M (voir exemple 8). Par conséquent, lorsque toutes les variables multi-occurrentes de f sont détectées monotones, F_M produit l'image exacte (aux arrondis près) de f sur $\boldsymbol{X}$.

Exemple 8 (Extension par monotonie) *Soient $f(x, y, z) = -x^2 + xy + yz - 3z$ et la boîte $\boldsymbol{X} = ([6, 8], [2, 4], [7, 15])$. L'évaluation naturelle produit $F_N(\boldsymbol{X}) = -[6, 8]^2 + [6, 8] \times$*

$[2,4]+[2,4]\times[7,15]-3[7,15]=[-83,35]$. *Les dérivées partielles de f par rapport à x, y et z sont :*

$$\frac{\partial f}{\partial x}(x,y,z) = -2x+y \in [-14,-8] \leq 0 \tag{3.27}$$

$$\frac{\partial f}{\partial y}(x,y,z) = x+z \in [13,23] \geq 0 \tag{3.28}$$

$$\frac{\partial f}{\partial z}(x,y,z) = y-3 \in [-1,1] \tag{3.29}$$

f est décroissante par rapport à x et croissante par rapport à y. En revanche, z n'est pas détectée monotone. L'évaluation par monotonie produit :

$$F_M(\boldsymbol{X}) = [\underline{F(8,2,[7,15])}, \overline{F(6,4,[7,15])}] = [-79,27] \subset F_N(\boldsymbol{X}) \tag{3.30}$$

Afin d'améliorer l'encadrement de f sur $\boldsymbol{X}$, [Araya 10] propose de calculer les dérivées partielles de f séparément sur les boîtes $\boldsymbol{X}^-$ et $\boldsymbol{X}^+$ dans lesquelles les variables déjà détectées monotones ont été remplacées par une de leurs bornes. Cette nouvelle extension F_{Mrec}, appelée *extension récursive par monotonie*, fournit un encadrement toujours au moins aussi bon que l'extension par monotonie F_M :

$$f(\boldsymbol{X}) \subset F_{Mrec}(\boldsymbol{X}) \subset F_M(\boldsymbol{X}) \subset F_N(\boldsymbol{X}) \tag{3.31}$$

3.2.4 Arithmétique affine

L'arithmétique affine (AA) est une méthode d'analyse numérique alternative à l'AI permettant de construire un encadrement d'une fonction sur une boîte. Elle consiste à représenter chaque quantité du calcul par une combinaison linéaire de variables symboliques : les dépendances linéaires entre les variables sont mémorisées (le calcul $X+2Y-3X$ n'est en conséquence pas sujet au problème de dépendance) et les opérations non-linéaires sont linéarisées en introduisant un terme d'erreur. Davantage de détails sont fournis dans l'annexe A.2.

L'efficacité de l'AA dans les algorithmes d'optimisation globale n'est plus à démontrer [Messine 97, Ninin 10b], notamment parce qu'elle exhibe une convergence quadratique. Néanmoins, cette technique demeure complexe à implémenter et coûteuse en termes d'opérations sur les intervalles. Pour ces raisons, l'AA n'est pas utilisée dans nos travaux.

3.3 Algorithme de branch and bound par intervalles

Initialement destiné à contenir les erreurs d'arrondis dans les calculs numériques, le calcul par intervalles a vu le nombre de ses applications exploser ces dernières décennies. Celles-ci incluent notamment l'optimisation globale, l'optimisation robuste, la satisfaction de contraintes, la recherche de racines et l'intégration numérique.

Les capacités de l'AI à raisonner de manière ensembliste en font un outil de choix pour l'optimisation globale de problèmes continus. La méthode générique de branch and bound est introduite dans la sous-section 3.3.1, puis étendue aux intervalles dans la sous-section 3.3.2. Les sous-sections 3.3.3 et 3.3.4 détaillent plusieurs heuristiques de recherche et d'accélération.

3.3.1 Branch and bound

L'algorithme de branch and bound (BB) [Lawler 66], ou séparation et évaluation, est une méthode générique d'optimisation globale combinatoire (discrète) de problèmes sous contraintes :

$$\begin{array}{lll} (\mathcal{P}) & \min\limits_{\boldsymbol{x}\in D} & f(\boldsymbol{x}) \\ & s.c. & g_i(\boldsymbol{x}) \leq 0 \quad (i \in \{1,\ldots,m\}) \\ & & h_j(\boldsymbol{x}) = 0 \quad (j \in \{1,\ldots,p\}) \end{array} \tag{3.32}$$

où D est un ensemble discret.

Remarque 3 *Généralement, on considère qu'une contrainte d'égalité h_j est satisfaite si la contrainte relaxée $-\varepsilon_= \leq h_j \leq \varepsilon_=$ (avec $\varepsilon_=$ arbitrairement petit) est satisfaite.*

Il n'est pas toujours possible d'énumérer tous les éléments de D, soit parce qu'il n'existe pas d'algorithme simple pour le faire, soit parce que la cardinalité de D rend l'énumération impossible en pratique. Un algorithme de BB partitionne l'espace de recherche D en sous-espaces S_k et construit un arbre de recherche dont les feuilles sont des solutions ponctuelles. Les sous-espaces ne contenant pas de solutions optimales sont éliminés et l'algorithme est appliqué récursivement sur les S_k restants. La meilleure solution des sous-problèmes générés est alors solution du problème initial.

Bien que la complexité algorithmique soit exponentielle dans le pire des cas, l'élimination de sous-espaces en début de recherche évite souvent l'énumération systématique de toutes les solutions. L'algorithme de BB converge généralement en temps fini (théorème 8), mais pas nécessairement raisonnable.

Théorème 8 (Convergence d'un BB) *Si le partitionnement de l'espace de recherche est un processus fini, alors l'algorithme de BB termine en temps fini.*

Sur chaque sous-espace S_k sont estimés des minorants $g_i^-(S_k)$, $h_j^-(S_k)$, $f^-(S_k)$ et des majorants $g_i^+(S_k)$, $h_j^+(S_k)$, $f^+(S_k)$ respectivement des contraintes g_i, h_j et de la fonction objectif f. Chaque sous-espace S_k est alors étiqueté suivant les règles de la définition 18. $\tilde{f}$ dénote le meilleur majorant connu du minimum global f^*.

Définition 18 (Classification des sous-espaces) *Soit S_k un sous-espace de l'espace de recherche.*

– S_k *est* réalisable *si tous les points de* S_k *satisfont* g_i *:*

$$\forall i \in \{1, \ldots, m\}, \quad g_i^+(S_k) \leq 0 \tag{3.33}$$

et si tous les points de S_k *satisfont* h_j *:*

$$\forall j \in \{1, \ldots, p\}, \quad -\varepsilon_= \leq h_i^-(S_k) \wedge h_i^+(S_k) \leq \varepsilon_= \tag{3.34}$$

$\tilde{f}$ *est alors mis à jour avec la valeur de* $f^+(S_k)$ *;*

– S_k *est* non réalisable *si tous les points de* S_k *violent une contrainte* g_i *:*

$$\exists i \in \{1, \ldots, m\}, \quad 0 < g_i^-(S_k) \tag{3.35}$$

ou si tous les points de S_k *violent une contrainte* h_j *:*

$$\exists j \in \{1, \ldots, p\}, \quad \varepsilon_= < h_j^-(S_k) \vee h_j^+(S_k) < -\varepsilon_= \tag{3.36}$$

– S_k *est* sous-optimal *si* S_k *est réalisable et si tout point de* S_k *a une évaluation plus mauvaise que* $\tilde{f}$ *:*

$$\tilde{f} < f^-(S_k) \tag{3.37}$$

– S_k *est* indécidable *s'il n'est ni réalisable, ni non réalisable, ni sous-optimal : il est alors possible que* S_k *contienne un minimiseur global.*

Durant l'exploration de l'espace de recherche, les sous-espaces non réalisables et sous-optimaux sont éliminés avec la garantie qu'aucune solution optimale ne s'y trouve, et ne sont pas davantage explorés. Les sous-espaces restants sont généralement stockés dans une structure de données adaptée : l'ordre dans lequel sont traités ces sous-espaces détermine la nature de l'exploration de l'espace de recherche. Plusieurs heuristiques sont mentionnées dans la section 3.3.3. Les sous-espaces réalisables et indécidables sont récursivement traités jusqu'à élimination. La solution optimale de $(\mathcal{P})$ est obtenue lorsque tous les sous-espaces de l'espace de recherche ont été traités.

Une extension connue sous le nom de branch and bound spatial [Ryoo 95] permet de résoudre des problèmes continus et mixtes (faisant intervenir des variables continues et discrètes). Un minorant de la fonction objectif est calculé par relaxation convexe sur chaque sous-espace. On peut montrer qu'un tel algorithme converge en temps fini (théorème 8), puisque les flottants ont une représentation finie sur une machine. Les propriétés conservatives de l'AI permettent également de construire de manière automatique un minorant et un majorant rigoureux d'une fonction décomposable sur une boîte. L'intégration de techniques d'intervalles dans un algorithme de BB est décrite dans la sous-section suivante.

3.3.2 Branch and bound par intervalles

Les premiers algorithmes de branch and bound par intervalles (BBI) sont dus à [Moore 76] et [Skelboe 74] : l'espace de recherche est partitionné en boîtes disjointes sur

lesquelles les fonctions d'inclusion de la fonction objectif et des contraintes sont évaluées. Les boîtes $\boldsymbol{X}$ en attente sont insérées dans une file de priorité $\mathcal{L}$ et rangées dans l'ordre croissant des minorants $\underline{F(\boldsymbol{X})}$. Une boîte extraite de $\mathcal{L}$ est bissectée (partitionnée en deux sous-boîtes) orthogonalement à la dimension la plus large. Pour une précision ε donnée, l'algorithme fournit une solution $\tilde{\boldsymbol{x}}$ (d'évaluation $\overline{F(\tilde{\boldsymbol{x}})} = \tilde{f}$) qui vérifie $\tilde{f} - f^* < \varepsilon$, même en présence d'erreurs d'arrondis. Une boîte $\boldsymbol{X}$ est éliminée lorsqu'il n'est pas possible d'améliorer $\tilde{f}$ de plus de ε, c'est-à-dire lorsque $\tilde{f} - \varepsilon < \underline{F(\boldsymbol{X})}$.

Un BBI générique est décrit dans l'algorithme 5. Certaines implémentations intègrent des techniques d'accélération (voir sous-section 3.3.4) permettant d'éliminer ou réduire les boîtes extraites de $\mathcal{L}$ sans perdre de solutions. Si une boîte ne peut être éliminée, elle est partitionnée (voir sous-section 3.3.3) et les sous-boîtes sont insérées dans $\mathcal{L}$.

Algorithme 5 Branch and bound par intervalles

fonction BBI($\boldsymbol{X}_0$: boîte initiale, F : fonction objectif, $\mathcal{C}$: système de contraintes)
  $\tilde{f} \leftarrow +\infty$ ▷ Meilleur majorant connu
  $\mathcal{L} \leftarrow \{\boldsymbol{X}_0\}$ ▷ File de priorité
  tant que $\mathcal{L} \neq \varnothing$ **faire**
    Extraction d'une boîte $\boldsymbol{X}$ de $\mathcal{L}$ ▷ section 3.3.3
    Techniques d'accélération ▷ section 3.3.4
    Evaluation des contraintes de $\mathcal{C}$
    si $\boldsymbol{X}$ ne peut pas être éliminée **alors**
      Mise à jour du meilleur majorant du minimum global
      Partitionnement de $\boldsymbol{X}$ en $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k\}$ ▷ section 3.3.3
      Insertion de $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k\}$ dans $\mathcal{L}$
    fin si
  fin tant que
  renvoyer $\tilde{f}$
fin fonction

Remarque 4 *Une boîte étant un vecteur d'intervalles fermés, deux sous-boîtes issues du partitionnement d'une boîte partagent une face.*

3.3.3 Heuristiques

Les heuristiques d'un BBI jouent un rôle prépondérant dans la vitesse de convergence de l'algorithme, et dépendent en général du problème traité. On distingue essentiellement deux types d'heuristiques : les stratégies d'exploration de l'espace de recherche et les méthodes de partitionnement des boîtes.

Exploration de l'espace de recherche

L'ordre d'insertion des boîtes restantes dans la file de priorité $\mathcal{L}$ détermine la stratégie d'exploration de l'espace de recherche. Les heuristiques de recherche les plus utilisées sont :

– « best-first search » : la boîte $\boldsymbol{X}$ pour laquelle $\underline{F_N(\boldsymbol{X})}$ est la plus faible est extraite de $\mathcal{L}$. Cette stratégie concentre l'exploration dans les sous-espaces les plus « prometteurs » ;
– « largest first » : la boîte de largeur maximale est extraite de $\mathcal{L}$. Cette stratégie revient à explorer les boîtes les plus anciennes et correspond à un parcours en largeur de l'arbre de recherche ;
– « depth-first search » : cette stratégie correspond à un parcours en profondeur de l'arbre de recherche.

Partitionnement des boîtes

Historiquement, deux heuristiques de partitionnement des boîtes se sont distinguées :
– la variable dont le domaine est le plus large est bissectée. Cette stratégie tend à conserver des boîtes de forme hypercubique ;
– les variables sont bissectées l'une après l'autre (schéma de type round robin) à chaque point de choix.

Ces dernières années, l'heuristique Smear [Csendes 97] s'est progressivement imposée comme une alternative compétitive aux deux heuristiques précédentes : la variable x_i pour laquelle la quantité intervalle $\frac{\partial F}{\partial x_i}(\boldsymbol{X})(X_i - x_i)$ est la plus large sur la boîte $\boldsymbol{X}$ est bissectée. Cette stratégie revient à sélectionner la variable pour laquelle la forme de Taylor varie le plus.

3.3.4 Techniques d'accélération

Calcul de majorant

Le majorant de la fonction objectif $\overline{F(\boldsymbol{X})}$ calculé par AI sur une boîte réalisable $\boldsymbol{X}$ est garanti d'être un majorant du minimum global f^*, mais n'est pas nécessairement atteignable et peut constituer un majorant grossier. [Ichida 79] a proposé d'évaluer systématiquement le milieu de chaque boîte $\boldsymbol{X}$ pour obtenir un majorant de f^* (exemple 9). Si $m(\boldsymbol{X})$ est un point réalisable, le meilleur majorant connu $\tilde{f}$ de f^* peut être mis à jour en utilisant l'évaluation $\overline{F(m(\boldsymbol{X}))}$. Certaines implémentations font appel à des solveurs d'optimisation locale afin d'améliorer le meilleur majorant connu.

Remarque 5 *Lorsque $\boldsymbol{X}$ est une boîte réalisable, il n'est pas nécessaire de vérifier si $m(\boldsymbol{X})$ est réalisable, puisque tout point de $\boldsymbol{X}$ satisfait les contraintes. En revanche, si $\boldsymbol{X}$ est indécidable, les contraintes sont évaluées sur $m(\boldsymbol{X})$ en utilisant l'AI.*

Exemple 9 (Branch and bound par intervalles) *Soit $f(x) = x^2\cos(x) + x$. Cherchons le minimum global de f sur l'intervalle $X = [-5, 3]$ avec une précision ε (figure 3.3). L'image exacte de f sur X est $f(X) = [-15.311, 2.092]$.*

1. *le meilleur majorant connu $\tilde{f}$ du minimum global est initialisé à $+\infty$;*
2. *X est bissecté en $X_1 = [-5, -1]$ et $X_2 = [-1, 3]$;*
3. *traitement de X_1 : un minorant est $\underline{F_N(X_1)} = -30$. $\tilde{f}$ est mis à jour par l'évaluation du point milieu : $\tilde{f} \leftarrow \overline{F(-3)} = -11.91$. X_1 est bissecté en $X_3 = [-5, -3]$ et $X_4 = [-3, -1]$;*

4. *traitement de X_2 : un minorant est $\underline{F_N(X_2)} = -9.91 \geq \tilde{f}$. X_2 ne peut pas contenir de minimum global et est éliminé ;*
5. *traitement de X_4 : un minorant est $\underline{F_N(X_4)} = -11.91 \geq \tilde{f}$. X_4 ne peut pas contenir de minimum global et est éliminé ;*
6. *traitement de X_3 : un minorant est $\underline{F_N(X_3)} = -30$. L'évaluation du point milieu améliore $\tilde{f}$: $\tilde{f} \leftarrow \overline{F(-4)} = -14.458$. X_3 est bissecté en $X_5 = [-5, -4]$ et $X_6 = [-4, -3]$;*
7. *traitement de X_6 : un minorant est $\underline{F_N(X_6)} = -20$. L'évaluation du point milieu améliore $\tilde{f}$: $\tilde{f} \leftarrow \overline{F(-4)} = -14.972$;*
8. *traitement de X_5 : un minorant est $\underline{F_N(X_5)} = -21.341$.*

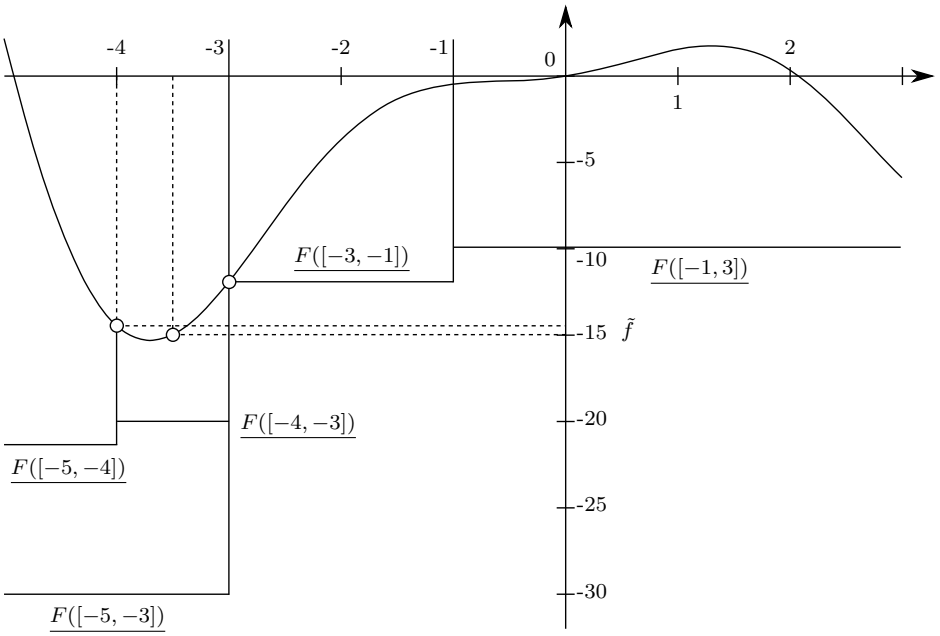


FIGURE 3.3 – Recherche du minimum global de $f(x) = x^2\cos(x) + x$ sur $X = [-5, 3]$ par un algorithme de branch and bound par intervalles

A ce stade, les minorants calculés sur les intervalles X_5 et X_6 ne permettent pas de les éliminer. Il est en revanche garanti qu'un minimiseur global de f sur X appartient à $X_5 \cup X_6 = [-5, -3]$. Les deux intervalles sont bissectés et récursivement traités pour mettre à jour $\tilde{f}$ et raffiner le calcul des minorants. L'algorithme termine lorsque tous les sous-intervalles X_k vérifient la condition de coupe $\tilde{f} - \varepsilon \leq \underline{F_N(X_k)}$.

Calcul de minorants et réduction des boîtes

[Kearfott 93, Du 94] ont montré que le temps de convergence d'un algorithme de BBI dépend fortement du comportement de la fonction au voisinage des minima et de l'ordre de

convergence de l'extension aux intervalles utilisée. L'AA ainsi que les techniques présentées dans les sections 3.2.2 (extensions de deuxième ordre) et 3.2.3 (extension par monotonie) sont des outils puissants pour le calcul de minorants de bonne qualité.

Les techniques de réduction de boîtes sont issues des communautés :

- d'AI : [Sotiropoulos 05] réduit les bornes de la boîte en combinant une forme de Taylor avec la contrainte $f \leq \tilde{f}$. [Yamamura 98] remplace les termes non-linéaires des contraintes par leur évaluation en AI, puis réduit les bornes des variables en $2n$ appels au simplexe ;
- de programmation par contraintes sur intervalles (PPCI) : les techniques de propagation de contraintes, inspirées de la programmation par contraintes [Mackworth 77], réduisent les bornes d'une boîte par rapport aux contraintes du problème. Ces techniques, appelées algorithmes de filtrage ou contracteurs, sont présentées dans le chapitre 4.

Lorsque le problème est contraint, le minorant de la fonction objectif obtenu par AI est en général grossier, car il ne tient pas compte du domaine réalisable délimité par les contraintes. Les contracteurs basés sur la programmation linéaire linéarisent le problème (fonction objectif et contraintes), puis fournissent un minorant de la fonction objectif sur le polytope des contraintes et (ou) réduisent les domaines des variables. Lorsque le problème n'est pas contraint, les conditions nécessaires d'optimalité locale sur une boîte (théorème 2, page 7) peuvent être vérifiées par des tests de monotonie et de convexité. La contrainte $\nabla f = 0$ peut notamment être traitée par des contracteurs afin d'éliminer des points non stationnaires [Van Hentenryck 97].

3.4 Différentiation automatique

Les logiciels de calcul formel, implémentant la différentiation symbolique, sont capables de générer les formules des dérivées partielles d'une fonction à partir de son expression analytique. Deux problèmes majeurs sont pourtant à déplorer : la taille de ces expressions augmente rapidement avec celle de la fonction, et les solveurs peinent généralement à fournir une expression unique des dérivées partielles lorsque la fonction fait intervenir des branchements (instructions « if then else »).

La différentiation automatique (DA) est une technique d'évaluation *ponctuelle* des dérivées partielles d'une fonction décrite par une composition de fonctions élémentaires (par exemple, un programme informatique). L'expression de la fonction est représentable par un graphe de calcul, dont chaque nœud représente une expression intermédiaire et chaque arc une fonction élémentaire. La DA s'appuie sur la règle de dérivation des fonctions composées (théorème 9) appliquée à des valeurs numériques.

Théorème 9 (Dérivation des fonctions composées) *Soient $x = x(u, v)$ et $y = y(u, v)$ deux applications dérivables au point (u, v). Supposons que $z = f(x, y)$ est dérivable au*

point $(x(u,v), y(u,v))$*. Alors les dérivées partielles de* $f(x(u,v), y(u,v))$ *sont données par :*

$$\begin{aligned} \frac{\partial z}{\partial u} &= \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u} \\ \frac{\partial z}{\partial v} &= \frac{\partial z}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial v} \end{aligned} \tag{3.38}$$

Deux modes de DA ont été développés : la DA en mode direct (1965-1970) évalue la fonction et ses dérivées partielles en parallèle, et la DA en mode adjoint (1976-1980) évalue d'abord la fonction, puis les dérivées partielles dans le sens inverse.

3.4.1 Mode direct

Le mode direct évalue l'expression 3.38 de droite à gauche : les dérivées partielles directionnelles sont calculées en partant des feuilles du graphe de calcul (les variables) et en propageant vers le haut les évaluations intermédiaires de la fonction et de son gradient. Notons que le processus n'est pas limité à l'évaluation des dérivées premières, mais peut être étendu au calcul des dérivées d'ordre supérieur. Le calcul des n dérivées partielles de la fonction f par le mode direct a une complexité de $O(ne)$, où e est le nombre d'opérations effectuées lors de l'évaluation de f. La complexité peut être réduite à $O(e)$ en privilégiant le mode adjoint.

3.4.2 Mode adjoint

Le mode adjoint évalue l'expression dans l'ordre inverse du mode direct, et exploite le fait que les termes $\frac{\partial z}{\partial x}$ et $\frac{\partial z}{\partial y}$ de l'équation 3.38 sont communs aux deux dérivées partielles $\frac{\partial z}{\partial u}$ et $\frac{\partial z}{\partial v}$. Une phase montante d'évaluation associe à chaque nœud de l'arbre syntaxique le résultat intermédiaire. Une phase descendante calcule la dérivée partielle de chaque nœud par rapport à ses nœuds fils (exemple 10). La dérivée partielle de f par rapport à une variable x_i est alors obtenue en sommant les dérivées partielles $\frac{\partial f}{\partial x_i}$ par rapport à chacune des occurrences de x_i (les feuilles de l'arbre). Les n dérivées partielles sont obtenues en $O(e)$.

Exemple 10 (Différentiation automatique en mode adjoint) *Soit* f *la fonction à deux variables définie par :*

$$f(x,y) = \cos^2(x) - xy \tag{3.39}$$

L'expression de f *s'écrit comme la composition d'opérations élémentaires :*

$$\begin{aligned} t_1 &= x & t_4 &= t_3^2 \\ t_2 &= y & t_5 &= t_1 t_2 \\ t_3 &= \cos(t_1) & t_6 &= t_4 - t_5 \end{aligned} \tag{3.40}$$

Le calcul du gradient en mode adjoint se déroule comme suit :

$$
\begin{aligned}
\frac{\partial t_6}{\partial t_6} &= 1 \qquad \frac{\partial t_6}{\partial t_5} = -1 \qquad \frac{\partial t_6}{\partial t_4} = 1 \\
\frac{\partial t_6}{\partial t_3} &= \frac{\partial t_6}{\partial t_4} \cdot \frac{\partial t_4}{\partial t_3} = 2t_3 \\
\frac{\partial f}{\partial y} := \frac{\partial t_6}{\partial t_2} &= \frac{\partial t_6}{\partial t_5} \cdot \frac{\partial t_5}{\partial t_2} = -t_1 \\
\frac{\partial f}{\partial x} := \frac{\partial t_6}{\partial t_1} &= \frac{\partial t_6}{\partial t_5} \cdot \frac{\partial t_5}{\partial t_1} + \frac{\partial t_6}{\partial t_3} \cdot \frac{\partial t_3}{\partial t_1} = -t_2 - 2t_3 \sin(t_1)
\end{aligned}
\tag{3.41}
$$

où t_2 et t_3 sont les résultats intermédiaires calculés durant la phase d'évaluation.

TABLE 3.1 – Différentiation automatique en mode adjoint

Nœuds	en $(1, 3)$	en $([0.9, 1], [2.9, 3.1])$
t_1	1	[0.9, 1]
t_2	3	[2.9, 3.1]
t_3	0.54	[0.54, 0.622]
t_4	0.292	[0.291, 0.39]
t_5	3	[2.61, 3.1]
t_6	-2.71	[-2.81, -2.22]
$\frac{\partial t_6}{\partial t_3}$	1.08	[1.08, 1.244]
$\frac{\partial f}{\partial x}$	-3.91	[-4.15, -3.74]
$\frac{\partial f}{\partial y}$	-1	[-1, -0.9]

Les étapes du calcul de gradient de f en le point $(1, 3)$ (en AF) et sur la boîte $([0.9, 1], [2.9, 3.1])$ (en AI) sont détaillées dans le tableau 3.1.

3.5 Conclusion

Les techniques d'intervalles constituent la seule approche pour l'optimisation globale robuste aux erreurs d'arrondis, et ont été utilisées avec succès pour la résolution de problèmes d'optimisation continue. Cependant, le problème de dépendance inhérent à l'AI handicape fortement les méthodes de résolution et limite la taille des instances résolubles. Certaines approches complémentaires (fonctions d'inclusion d'ordre supérieur, arithmétique affine, propagation de contraintes) sont désormais incontournables. Dans la suite de ce document, nous nous concentrons essentiellement sur les techniques de propagation de contraintes.

Chapitre 4

Contracteurs

Sommaire

4.1 Opérateurs de cohérence partielle 47
4.2 Cohérences locales 49
4.2.1 Cohérence 2B 49
4.2.2 Evaluation-propagation 50
4.2.3 Contracteurs exploitant la monotonie 51
4.2.4 Cohérence de boîte 52
4.2.5 Algorithme de Newton par intervalles 53
4.2.6 Point fixe 55
4.3 Cohérences fortes 56
4.3.1 Cohérence 3B 56
4.3.2 Cohérence CID 57
4.4 Cohérences globales 59
4.4.1 Algorithme de Newton 59
4.4.2 Convexification 60
4.5 Contraction et différentiation automatique 61

Les algorithmes de BBI sont depuis quelques années systématiquement dotés de procédures de contraction, dans le but de réduire les boîtes en raisonnant de manière locale (à l'échelle d'une contrainte individuell) ou globale (toutes les contraintes prises en compte en même temps). Ces algorithmes, appelés branch and contract par intervalles (BCI), alternent des phases de contraction (et évaluation) et bissection. Ce chapitre compare différentes cohérences partielles et globales, et les algorithmes de filtrage associés issus de la communauté d'AI et de la communauté de PPCI.

4.1 Opérateurs de cohérence partielle

Tout problème d'optimisation numérique peut être reformulé en un problème numérique de satisfaction de contraintes (NCSP) (définition 19), en maintenant une contrainte

dynamique $f \leq \tilde{f}$ sur le meilleur majorant connu du minimum global.

Définition 19 (Problème de satisfaction de contraintes) *Un problème de satisfaction de contraintes (CSP) est la donnée d'un triplet $P = (\mathcal{V}, \mathcal{C}, D)$, où $\mathcal{V}$ représente l'ensemble des variables du problème, $\mathcal{C}$ l'ensemble des contraintes et D le domaine initial des variables. Résoudre le problème P revient à trouver une instanciation (ou toutes les instanciations) de $\mathcal{V}$ dans D satisfaisant $\mathcal{C}$. On appelle NCSP un problème $P = (\mathcal{V}, \mathcal{C}, D)$ pour lequel D est un sous-ensemble de $\mathbb{R}^n$. D est alors noté sous la forme d'une boîte $\boldsymbol{X}$.*

Résoudre un NCSP est étroitement lié à la notion de cohérence partielle, une propriété locale permettant d'éliminer (filtrer) certaines valeurs du domaine qui n'appartiennent pas à l'ensemble des solutions d'une contrainte (on parle alors d'incohérence). Les valeurs incohérentes sont éliminées par des d'algorithmes de filtrage/contraction (ou contracteurs, définition 20).

Définition 20 (Contracteur [Chabert 09a]) *Soient $\boldsymbol{X} \in \mathbb{I}^n$ une boîte, c une contrainte réelle et ρ_c la relation définie par c. On appelle* contracteur *une fonction OC (*outer contractor*) qui satisfait la propriété de correction (figure 4.1), c'est-à-dire qui réduit $\boldsymbol{X}$ en éliminant des valeurs incohérentes par rapport à c :*

$$\boldsymbol{X} \cap \rho_c \subset OC(\boldsymbol{X}, c) \tag{4.1}$$

Un contracteur qui calcule une cohérence partielle ϕ est appelé opérateur de cohérence ϕ et est noté OC_ϕ.

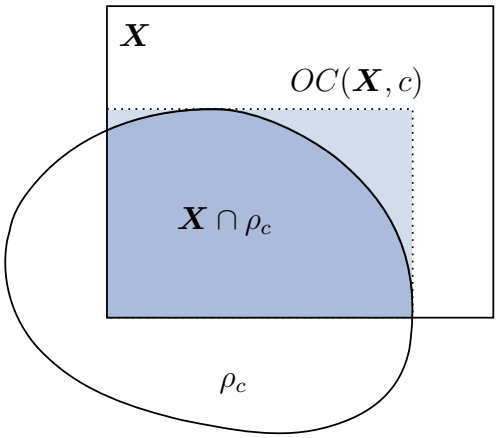


FIGURE 4.1 – Contraction d'une boîte $\boldsymbol{X}$ par rapport à une contrainte c

Les contracteurs peuvent avoir les propriétés suivantes :

$$\text{Monotone} : \boldsymbol{X} \subset \boldsymbol{Y} \Rightarrow OC(\boldsymbol{X}, c) \subset OC(\boldsymbol{Y}, c) \tag{4.2}$$

$$\text{Convergent} : \boldsymbol{X}_k \underset{k\to+\infty}{\longrightarrow} \boldsymbol{x} \Rightarrow OC(\boldsymbol{X}_k, c) \underset{k\to+\infty}{\longrightarrow} \boldsymbol{x} \cap \rho_c \tag{4.3}$$

$$\text{Idempotent} : \forall \boldsymbol{X} \in \mathbb{I}^n, OC(OC(\boldsymbol{X}, c), c) = OC(\boldsymbol{X}, c) \tag{4.4}$$

$$\text{Minimal} : \forall \boldsymbol{X} \in \mathbb{I}^n, OC(\boldsymbol{X}, c) = \square(\boldsymbol{X} \cap \rho_c) \tag{4.5}$$

$$\text{Mince} : \forall \boldsymbol{x} \in \mathbb{R}^n, OC(\boldsymbol{x}, c) = \boldsymbol{x} \cap \rho_c \tag{4.6}$$

La programmation sur les contracteurs [Chabert 09a] consiste à définir des opérations (intersection, union, composition, répétition) permettant de construire des contracteurs plus complexes (définition 21).

Définition 21 (Intersection et composition de contracteurs) *Soient $\boldsymbol{X}$ une boîte, c_1 et c_2 deux contraintes et OC_1 et OC_2 deux contracteurs respectivement associés à c_1 et c_2. Les opérations d'intersection et de composition de OC_1 et OC_2 sont définies par :*

$$\textit{Intersection} : OC_1(\boldsymbol{X}, c_1) \cap OC_2(\boldsymbol{X}, c_2) \tag{4.7}$$

$$\textit{Composition} : OC_2(OC_1(\boldsymbol{X}, c_1), c_2) \tag{4.8}$$

La composition de contracteurs possède un pouvoir filtrant plus élevé qu'une simple intersection – la boîte en entrée du deuxième contracteur est la boîte contractée par le premier contracteur – et est donc toujours privilégiée en pratique.

La section 4.2 introduit deux types de cohérences partielles dérivées de la cohérence d'arêtes sur les problèmes discrets : la cohérence 2B (*2B consistency* ou *hull consistency*) et la cohérence de boîte (*box consistency*). Les cohérences partielles 3B et CID, plus fortes que les cohérences 2B et de boîte, sont introduites dans la section 4.3. Des algorithmes de cohérence globale, basés sur une linéarisation des contraintes, sont présentés dans la section 4.4.

4.2 Cohérences locales

4.2.1 Cohérence 2B

La cohérence 2B maintient une propriété de cohérence d'arêtes pour chaque borne des variables intervenant dans une contrainte (définition 22). Géométriquement parlant, chaque face d'une boîte 2B-cohérente par rapport à un système de contraintes $\mathcal{C}$ intersecte toutes les contraintes de $\mathcal{C}$.

Définition 22 (Cohérence 2B) *Soit c une contrainte n-aire. Une boîte $\boldsymbol{X}$ est dite 2B-cohérente par rapport à c lorsque :*

$$\forall i \in \{1, \ldots, n\}, \quad X_i = \square\{x_i \in X_i \mid \exists x_1 \in X_1, \ldots, \exists x_n \in X_n, c(x_1, \ldots, x_i, \ldots, x_n)\} \tag{4.9}$$

$\boldsymbol{X}$ est dite 2B-cohérente par rapport à un système de contraintes $\mathcal{C}$ si $\boldsymbol{X}$ est 2B-cohérente par rapport à chacune des contraintes de $\mathcal{C}$.

4.2.2 Evaluation-propagation

L'algorithme d'évaluation-propagation [Messine 97], également connu sous les noms HC4Revise [Benhamou 99] ou FBBT [Belotti 09], est un algorithme inspiré des travaux de [Cleary 87] sur l'AI relationnelle. Applicable à tout problème explicite, il calcule une approximation de la cohérence 2B. Il consiste à propager une contrainte en effectuant un double parcours de son arbre syntaxique, dans le but de contracter chaque occurrence des variables (exemple 11).

Une première phase montante d'évaluation calcule en chacun des nœuds de l'arbre le résultat intermédiaire par AI. Le résultat attaché à la racine de l'arbre, correspondant à l'évaluation de la contrainte, est ensuite intersecté avec l'intervalle imposé par la contrainte. La seconde phase propage ce résultat de haut en bas en projetant (inversant) les opérations en chaque nœud [Goualard 08]. Si une incohérence est détectée, la boîte ne peut satisfaire la contrainte et est éliminée. Sinon, les variables (feuilles de l'arbre) sont éventuellement contractées.

Exemple 11 (Algorithme HC4Revise) *Soient :*
- $2x = z - y^2$ *une contrainte d'égalité ;*
- *les domaines* $X = [0, 20]$, $Y = [-10, 10]$ *et* $Z = [0, 16]$.

La décomposition de la contrainte en opérations élémentaires s'écrit :

$$\begin{array}{ll} n_1 := 2x & n_3 := z - n_2 \\ n_2 := y^2 & n_1 = n_3 \end{array} \tag{4.10}$$

La phase d'évaluation de la contrainte (figure 4.2) évalue les nœuds de l'arbre syntaxique :

$$\begin{aligned} N_1 &= 2X = 2 \times [0, 20] = [0, 40] \\ N_2 &= Y^2 = [-10, 10]^2 = [0, 100] \\ N_3 &= Z - N_2 = [0, 16] - [0, 100] = [-100, 16] \end{aligned} \tag{4.11}$$

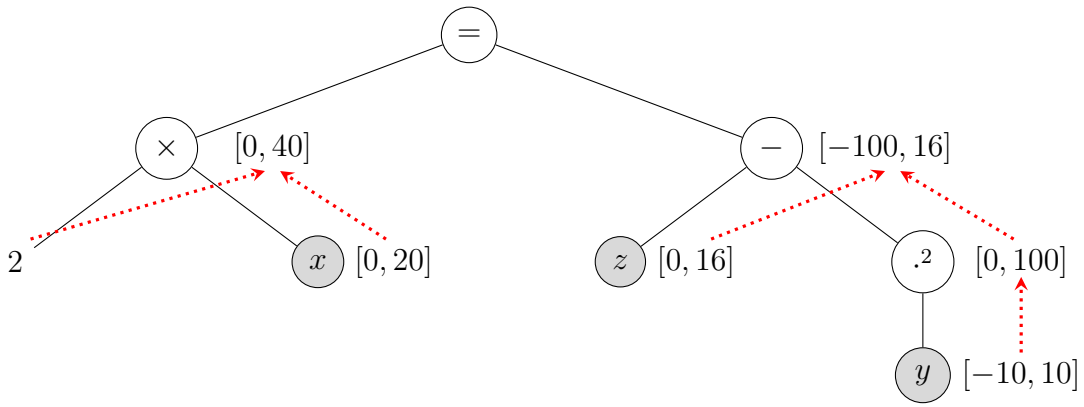


FIGURE 4.2 – HC4Revise : phase montante d'évaluation

La phase de propagation de la contrainte (figure 4.3) intersecte l'évaluation du membre gauche de l'équation ($n_1 = [0, 40]$) et celle du membre droit ($n_3 = [-100, 16]$), puis propage la contrainte en chaque nœud de l'arbre syntaxique en utilisant les fonctions de projection des opérations élémentaires :

$$\begin{aligned}
N_1' &= N_3' = N_1 \cap N_3 = [0, 40] \cap [-100, 16] = [0, 16] \\
X' &= X \cap \frac{N_1'}{2} = [0, 20] \cap [0, 8] = [0, 8] \\
Z' &= Z \cap (N_2 + N_3') = [0, 16] \cap ([0, 100] + [0, 16]) = [0, 16] \\
N_2' &= N_2 \cap (Z' - N_3') = [0, 100] \cap ([0, 16] - [0, 16]) = [0, 16] \\
Y' &= \square \left(Y \cap (-\sqrt{N_2'}), Y \cap \sqrt{N_2'} \right) = \square([-4, 0], [0, 4]) = [-4, 4]
\end{aligned} \tag{4.12}$$

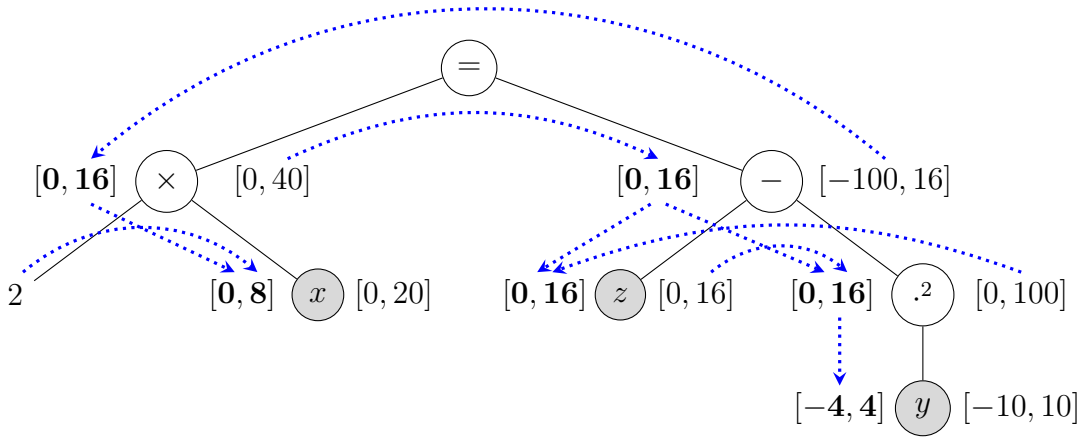


FIGURE 4.3 – HC4Revise : phase descendante de projection

La boîte initiale $([0, 20], [-10, 10], [0, 16])$ *a été réduite à* $([0, 8], [-4, 4], [0, 16])$ *sans perte de solutions de la contrainte.*

4.2.3 Contracteurs exploitant la monotonie

L'algorithme adaptatif Mohc (MOnotonic Hull Consistency) [Araya 10] exploite la monotonie locale des variables pour contracter plus efficacement une boîte par rapport à un système d'équations où les variables apparaissent plusieurs fois. L'algorithme exploite le fait que l'image d'une fonction f peut être encadrée par deux fonctions d'expressions connues (voir définition 17) :

$$f(\boldsymbol{X}) \subset F_M(\boldsymbol{X}) := [\underline{F(\boldsymbol{X}^-)}, \overline{F(\boldsymbol{X}^+)}] \tag{4.13}$$

Par conséquent, l'équation $f = 0$ est décomposable en deux contraintes $F(\boldsymbol{X}^-) \leq 0$ et $F(\boldsymbol{X}^+) \geq 0$. Notons $\mathcal{V}$ l'ensemble des variables de f, $\mathcal{V}_m \subset \mathcal{V}$ l'ensemble des variables multi-occurrentes détectées monotones sur $\boldsymbol{X}$ et $\mathcal{V}_{nm} \subset \mathcal{V}$ le reste des variables.

Mohc est un algorithme de point fixe (voir sous-section 4.2.6) dont la procédure de révision MohcRevise (algorithme 6) invoque :

- HC4Revise($F(\boldsymbol{X}^-) \leq 0$) pour contracter les variables de $\mathcal{V}_{nm}$;
- HC4Revise($F(\boldsymbol{X}^+) \geq 0$) pour contracter les variables de $\mathcal{V}_{nm}$;
- une version de l'algorithme de Newton par intervalles (voir sous-section 4.2.5), appelée MonotonicBoxNarrow, pour contracter les variables de $\mathcal{V}_m$.

Algorithme 6 MohcRevise

fonction MohcRevise($f = 0$: contrainte d'égalité, **in-out** $\boldsymbol{X}$: boîte, $\mathcal{V}$: variables)
  HC4Revise($F(\boldsymbol{X}) = 0$)
  si OccurrencesMultiples($\mathcal{V}$) **alors**
    HC4Revise($F(\boldsymbol{X}^-) \leq 0$)
    HC4Revise($F(\boldsymbol{X}^+) \geq 0$)
    pour x_i monotone **faire**
      $X_i \leftarrow$ MonotonicBoxNarrow($F, \boldsymbol{X}, i$)
    fin pour
  fin si
fin fonction

L'algorithme Octum, développé indépendamment par [Chabert 09b], est identique à MonotonicBoxNarrow lorsque la fonction est monotone par rapport à chacune de ses variables.

4.2.4 Cohérence de boîte

La cohérence de boîte [Benhamou 94, Collavizza 99a] définit une cohérence plus grossière que la cohérence 2B, puisque les variables existentiellement quantifiées de la définition 22 sont remplacées par des variables universellement quantifiées (définition 23).

Les algorithmes de cohérence de boîte produisent en revanche un bien meilleur filtrage lorsque les contraintes contiennent plusieurs occurrences des variables. En particulier, ils sont optimaux lorsque $c : X \subset \mathbb{R} \rightarrow \mathbb{R}$ est continue sur X par rapport à une unique variable x à occurrences multiples. Les algorithmes de cohérence 2B sont quant à eux très efficaces lorsque les variables n'ont qu'une seule occurrence.

Définition 23 (Cohérence de boîte) *Soient c une contrainte n-aire et C une extension aux intervalles de c. Une boîte $\boldsymbol{X}$ est dite boîte-cohérente par rapport à c lorsque :*

$$\forall i \in \{1, \ldots, n\}, \quad X_i = \square\{x_i \in X_i \mid C(X_1, \ldots, [x_i, x_i], \ldots, X_n)\} \tag{4.14}$$

$\boldsymbol{X}$ est dite boîte-cohérente par rapport à un système de contraintes $\mathcal{C}$ si $\boldsymbol{X}$ est boîte-cohérente par rapport à chacune des contraintes de $\mathcal{C}$.

4.2.5 Algorithme de Newton par intervalles

L'algorithme de Newton par intervalles est une extension aux intervalles de la méthode de Newton, permettant de trouver et borner rigoureusement *tous les zéros* d'une fonction sur un intervalle initial X. Il calcule une approximation de la cohérence de boîte.

Soit z un zéro d'une fonction f continue sur l'intervalle X et dérivable sur $int(X)$. Pour tout $c \in X$, le théorème des valeurs intermédiaires indique qu'il existe ξ strictement compris entre z et c tel que :

$$0 = f(z) = f(c) + f'(\xi)(z - c) \tag{4.15}$$

La valeur inconnue $f'(\xi)$ peut être bornée rigoureusement par $F'(X)$, où F' est une fonction d'inclusion de f'. On en déduit l'inclusion :

$$z \in N_f(X, c) := c - \frac{f(c)}{F'(X)} \tag{4.16}$$

où $N_f(X, c)$ désigne l'opérateur de Newton. Construisons par récurrence la suite :

$$\begin{cases} X_0 & = X \\ X_{k+1} & = X_k \cap N_f(X_k, c_k), \quad \forall k \geq 0 \end{cases} \tag{4.17}$$

Alors tous les zéros de f sont bornés avec une précision arbitraire. De plus, $X_{k+1} = \varnothing$ implique qu'il n'existe aucun zéro de f dans X_k, et $X_{k+1} \subset int(X_k)$ prouve l'existence d'un unique zéro dans X_{k+1} [Neumaier 90].

Il est possible que les zéros multiples de f soient séparés automatiquement grâce à l'AI étendue (exemple 12). Dans le cas général cependant, en raison de la surestimation du calcul de $F'(X)$, un point fixe $X_{k+1} = X_k$ est souvent atteint. [Hansen 92] préconise alors de bissecter X_k, puis d'itérer sur chacun des deux sous-intervalles.

Remarque 6 *En pratique, $f(c)$ peut être sujette à des arrondis numériques. Pour assurer la conservativité des calculs, il est impératif de lui substituer $F(c)$.*

Lorsqu'il existe un unique zéro z de f dans X, et sous les conditions $c = m(X)$ et f monotone sur X, l'algorithme de Newton par intervalles converge q-quadratiquement (définition 24) vers z, c'est-à-dire que le nombre de décimales correctes double à chaque itération asymptotiquement [Hansen 92].

Définition 24 (Convergence q-quadratique) *Soient $\{u_k\}_{k\geq 0}$ une suite et u^* un réel. On dit que u converge q-quadratiquement vers u^* s'il existe un réel C positif tel que :*

$$\forall k \geq 0, \quad \|u_{k+1} - u^*\| \leq C\|u_k - u^*\|^2 \tag{4.18}$$

Exemple 12 (Newton par intervalles) *Soit $f(x) = x^2 - 2$. Cherchons les zéros de f sur l'intervalle $X_0 = [-3, 2]$. Notons $c_0 = m(X_0) = -0.5$ et $F'(X) = 2X$. Alors :*

$$\begin{aligned} N_f(X_0, c_0) &= c_0 - \frac{f(c_0)}{F'(X_0)} = c_0 - \frac{c_0^2 - 2}{2X} = -0.5 - \frac{(-0.5)^2 - 2}{2[-3, 2]} = -0.5 - \frac{-1.75}{[-6, 4]} \\ &= (-0.5 + [-\infty, -\frac{7}{24}]) \cup (-0.5 + [\frac{7}{16}, +\infty]) \\ &= [-\infty, -\frac{19}{24}] \cup [-\frac{1}{16}, +\infty] \end{aligned} \tag{4.19}$$

La contraction de X_0 est alors constituée de deux sous-intervalles, représentés sur la figure 4.4 comme l'intersection entre l'axe des abscisses et les deux demi-cônes :

$$X_0 \cap N_f(X_0, c_0) = [-3, -\frac{19}{24}] \cup [-\frac{1}{16}, 2] \tag{4.20}$$

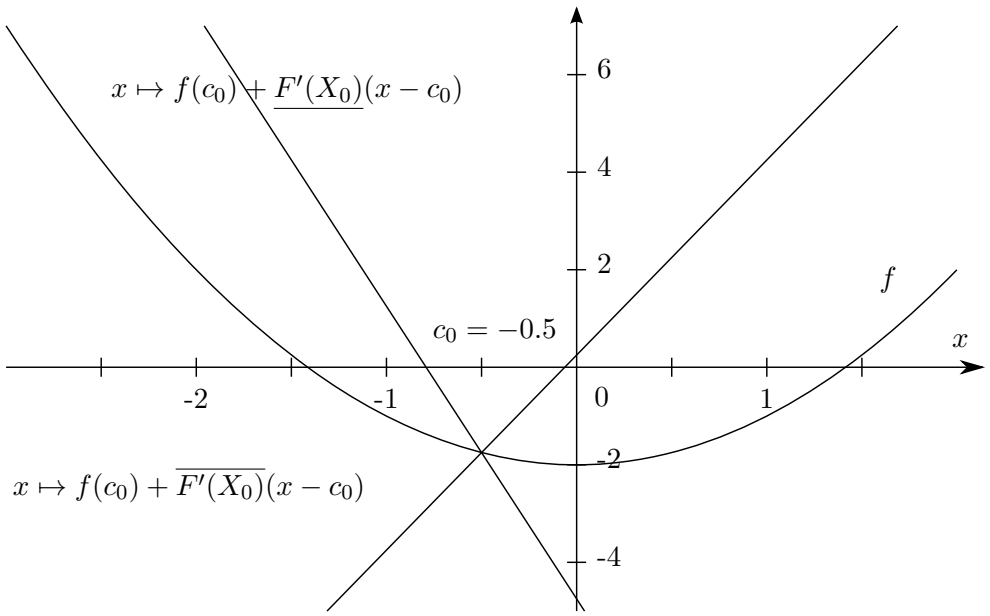


FIGURE 4.4 – Recherche des zéros de $f : x \mapsto x^2 - 2$ dans $X_0 = [-3, 2]$ par l'algorithme de Newton par intervalles

Notons $X_1 = [-3, -\frac{19}{24}]$ et $X_5 = [-\frac{1}{16}, 2]$, et appliquons-leur récursivement l'algorithme (tableau 4.1). Dans un souci de concision, les calculs sont limités à une précision 10^{-4}.

Les deux zéros de f sur X_0 sont bornés après 8 itérations, respectivement par les intervalles $[-1.414213693, -1.414213455]$ (de largeur $2.4 \cdot 10^{-7}$) et $[1.414213562, 1.414213562]$ (de largeur $6.1 \cdot 10^{-11}$). Leur existence est garantie lors des itérations 1 et 6 (en gras) : $X_2 := X_1 \cap N_f(X_1, c_1) \subset int(X_1)$ et $X_7 := X_6 \cap N_f(X_6, c_6) \subset int(X_6)$.

TABLE 4.1 – Newton par intervalles

k	X_k	c_k	$F'(X_k)$	$N_f(X_k, c_k)$	$X_k \cap N_f(X_k, c_k)$
1	$[-3, -0.7916]$	-1.8958	$[-6, -1.5833]$	$[-1.6302, -0.8889]$	$X_2 := [\mathbf{-1.6302}, \mathbf{-0.8889}]$
2	$[-1.6302, -0.8889]$	-1.2596	$[-3.2603, -1.7779]$	$[-1.4922, -1.3863]$	$X_3 := [-1.4922, -1.3863]$
3	$[-1.4922, -1.3863]$	-1.4393	$[-2.9843, -2.7727]$	$[-1.4154, -1.4134]$	$X_4 := [-1.4154, -1.4134]$
4	$[-1.4154, -1.4134]$	-1.4144	$[-2.8307, -2.8269]$	$[-1.4143, -1.4142]$	$[-1.4143, -1.4142]$
5	$[-0.0625, 2]$	0.9688	$[-0.125, 4]$	$[-\infty, -7.5234] \cup [1.2341, +\infty]$	$X_6 := [1.2341, 2]$
6	$[1.2341, 2]$	1.6171	$[2.4682, 4]$	$[1.3679, 1.4634]$	$X_7 := [\mathbf{1.3679}, \mathbf{1.4634}]$
7	$[1.3679, 1.4634]$	1.4156	$[2.7358, 2.9267]$	$[1.4141, 1.4143]$	$X_8 := [1.4141, 1.4143]$
8	$[1.4141, 1.4143]$	1.4142	$[2.8283, 2.8286]$	$[1.4142, 1.4143]$	$[1.4142, 1.4143]$

4.2.6 Point fixe

Afin de traiter un système de contraintes $\mathcal{C}$, un algorithme de point fixe est généralement intégré dans les solveurs : un point fixe (algorithme 7) est une boucle de propagation idempotente qui contracte une boîte $\boldsymbol{X}$ par rapport à $\mathcal{C}$. Un contracteur OC, appelé procédure de révision, manipule une contrainte du système à la fois. On note alors, par extension, $OC(\boldsymbol{X}, \mathcal{C})$ l'application PointFixe$(\boldsymbol{X}, \mathcal{C}, OC)$.

Une liste de contraintes $\mathcal{Q}$ contient initialement les contraintes de $\mathcal{C}$. A chaque itération, une contrainte c_i est extraite de $\mathcal{Q}$. Lorsque $\boldsymbol{X}$ est contractée par rapport à la contrainte c_i, les contraintes qui font intervenir les variables contractées dans $\boldsymbol{X}$ sont « réveillées » et insérées dans $\mathcal{Q}$. Sinon, c_i est supprimée de $\mathcal{Q}$ et la contrainte suivante est traitée par la procédure de révision.

Algorithme 7 Boucle de propagation (point fixe)

fonction POINTFIXE$(\boldsymbol{X}, \mathcal{C}, OC)$
  $\mathcal{Q} \leftarrow \mathcal{C}$
  répéter
    Choisir une contrainte c_i dans $\mathcal{Q}$
    $\boldsymbol{X}' \leftarrow OC(\boldsymbol{X}, c_i)$ ▷ invocation du contracteur OC
    si $\boldsymbol{X}' \neq \boldsymbol{X}$ **alors**
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{c_j \mid c_j \in \mathcal{C} \wedge \exists x_k \in var(c_j), X'_k \neq X_k\}$
      $\boldsymbol{X} \leftarrow \boldsymbol{X}'$
    fin si
    $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{c_i\}$
  jusqu'à $\mathcal{Q} = \varnothing$
fin fonction

Remarque 7 *Pour éviter les phénomènes de convergence lente, une contrainte c_j n'est en pratique insérée dans la liste de contraintes $\mathcal{Q}$ que lorsqu'elle fait intervenir des variables qui ont été* suffisamment *contractées, par exemple lorsque le ratio entre la taille du domaine*

contracté et la taille du domaine initial est inférieur à un seuil η :

$$\exists x_k \in var(c_j), \quad w(X'_k) < \eta w(X_k) \tag{4.21}$$

4.3 Cohérences fortes

Les cohérences 2B et de boîte sont dites faibles, car elles ne raisonnent que sur les bornes des variables à l'échelle d'une contrainte individuelle. En conséquence, elles conservent souvent des valeurs qui ne sont pas solutions du système de contraintes et le filtrage résultant est souvent de piètre qualité. Les cohérences partielles plus fortes 3B et CID, au pouvoir filtrant plus élevé, invoquent les opérateurs de cohérence 2B et de boîte pour « rogner » les domaines des variables. Le rognage consiste à assigner temporairement une petite portion de son domaine (une bandelette) à une variable, puis à éliminer cette bandelette du domaine de la variable par réfutation lorsque cela est possible. Contrairement à un algorithme de bissection, de complexité exponentielle, le rognage est un algorithme de réfutation polynomial. En revanche, le rognage n'est pas incrémental, dans le sens où une contraction sur le domaine d'une variable nécessite de relancer le filtrage sur toutes les autres variables.

Le rognage 3B (sous-section 4.3.1) consiste à éliminer les bandelettes extérieures sur lesquelles une incohérence est détectée. Le processus de réfutation est interrompu lorsqu'aucune borne ne peut être réduite. Le rognage CID (sous-section 4.3.2) traite l'intégralité des bandelettes afin d'éliminer les valeurs incohérentes communes aux bandelettes.

4.3.1 Cohérence 3B

La cohérence 3B (définition 25) [Lhomme 93] est une relaxation de la *singleton arc consistency* basée sur le rognage, dont le sous-contracteur (le contracteur de réfutation des bandelettes) est un opérateur de cohérence 2B. Elle consiste à vérifier que les bornes des variables sont cohérentes par rapport au système de contraintes. La cohérence 3B s'étend récursivement à la cohérence kB ($k > 2$) en invoquant un opérateur de cohérence $(k-1)$B sur les bandelettes.

Définition 25 (Cohérence 3B(s_{3B})) *Soient :*
- $P = (\mathcal{V}, \mathcal{C}, \boldsymbol{X})$ *un NCSP ;*
- OC_{2B} *un opérateur de cohérence 2B ;*
- s_{3B} *le nombre de bandelettes ;*
- $\boldsymbol{X_i^1}$ *la première bandelette du domaine de la variable x_i, c'est-à-dire la sous-boîte de $\boldsymbol{X}$ dont la $i$ème composante est $[\underline{X_i}, \underline{X_i} + \frac{w(X_i)}{s_{3B}}]$;*
- $\boldsymbol{X_i^{s_{3B}}}$ *la dernière bandelette du domaine de la variable x_i, c'est-à-dire la sous-boîte de $\boldsymbol{X}$ dont la $i$ème composante est $[\overline{X_i} - \frac{w(X_i)}{s_{3B}}, \overline{X_i}]$.*

$\boldsymbol{X}$ est dite 3B(s_{3B})-cohérente par rapport à $c \in \mathcal{C}$ lorsque :

$$\forall i \in \{1, \ldots, n\}, \quad OC_{2B}(\boldsymbol{X_i^1}, c) \neq \varnothing \wedge OC_{2B}(\boldsymbol{X_i^{s_{3B}}}, c) \neq \varnothing \tag{4.22}$$

$\boldsymbol{X}$ est dite 3B(s_{3B})-cohérente par rapport à $\mathcal{C}$ si elle est 3B(s_{3B})-cohérente par rapport à chacune des contraintes de $\mathcal{C}$.

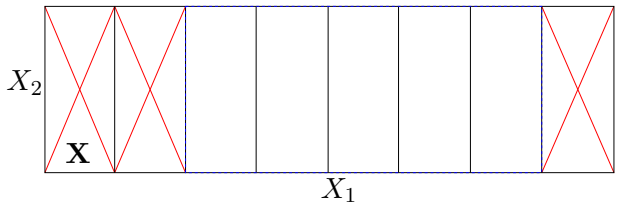


FIGURE 4.5 – Rognage 3B pour $s_{3B} = 8$

L'algorithme calculant la cohérence 3B procède à un rognage des domaines des variables à leurs bornes (figure 4.5). Une variable x_k est temporairement instanciée à un sous-intervalle (bandelette) de son domaine X_k, puis un sous-contracteur associé au système de contraintes est invoqué sur le sous-problème. Si une incohérence est détectée, la bandelette est éliminée du domaine X_k de la variable (algorithme 8).

Algorithme 8 Rognage 3B

fonction VARIABLE3B(**in-out** $\boldsymbol{X}$: boîte, i : composante, $\mathcal{C}$: système de contraintes, OC_{2B} : opérateur de cohérence 2B, s_{3B} : nombre de bandelettes)
  Réduire la borne gauche de X_i par OC_{2B}
  Réduire la borne droite de X_i par OC_{2B}
fin fonction
fonction ROGNAGE3B(**in-out** $\boldsymbol{X}$: boîte, $\mathcal{C}$: système de contraintes, OC_{2B} : opérateur de cohérence 2B, s_{3B} : nombre de bandelettes)
  pour $i \in \{1, \ldots, n\}$ **faire** ▷ boucle sur les variables
    VARIABLE3B($\boldsymbol{X}, i, \mathcal{C}, OC_{2B}, s_{3B}$)
  fin pour
fin fonction
fonction 3B(**in-out** $\boldsymbol{X}$: boîte, $\mathcal{C}$: système de contraintes, OC_{2B} : opérateur de cohérence 2B, s_{3B} : nombre de bandelettes)
  POINTFIXE($\boldsymbol{X}$, $\mathcal{C}$, ROGNAGE3B($\boldsymbol{X}, \mathcal{C}, OC_{2B}, s_{3B}$))
fin fonction

4.3.2 Cohérence CID

La disjonction constructive sur les CSP consiste à manipuler une disjonction de contraintes $c_1 \vee \ldots \vee c_m$ en prenant alternativement comme point de choix chacune des contraintes c_i indépendamment et en supprimant les valeurs incohérentes pour toutes les contraintes. La disjonction constructive sur intervalles, ou constructive interval disjunction (CID), consiste à éliminer les valeurs incohérentes communes à toutes les bandelettes via l'opérateur d'enveloppe convexe (définition 26).

Définition 26 (Cohérence CID(s_{CID})) *Soient :*
- $P = (\mathcal{V}, \mathcal{C}, \boldsymbol{X})$ *un NCSP ;*
- OC *un opérateur de contraction calculant une cohérence partielle ;*
- s_{CID} *le nombre de bandelettes ;*
- $\boldsymbol{X_i^k}$ *la* k*ième bandelette du domaine de la variable* x_i*, c'est-à-dire la sous-boîte de* $\boldsymbol{X}$ *dont la* i*ème composante est* $[\underline{X_i} + (k-1)\frac{w(X_i)}{s_{CID}}, \underline{X_i} + k\frac{w(X_i)}{s_{CID}}]$.

La variable $x_i \in \mathcal{V}$ *est* CID(s_{CID})-cohérente *par rapport à* P *et* OC *lorsque :*

$$\boldsymbol{X} = \bigsqcup_{k=1}^{s_{CID}} OC(\boldsymbol{X_i^k}, \mathcal{C}) \tag{4.23}$$

P *est* CID(s_{CID})-cohérent *si toutes les variables de* $\mathcal{V}$ *sont CID(s_{CID})-cohérentes.*

L'algorithme 3B réfute des valeurs incohérentes aux extrémités du domaine de la variable x_i, mais l'effort fourni pour contracter les autres variables x_j ($j \neq i$) est perdu. L'avantage substantiel de l'algorithme CID [Trombettoni 07] par rapport à 3B est de conserver en partie l'information de contraction sur les autres variables : l'enveloppe convexe des s_{CID} bandelettes contractées remplace alors la boîte initiale (algorithme 9).

La cohérence CID(s_{CID}) est difficile à obtenir car la convergence vers un point fixe est lente. En pratique, on cherche à obtenir un quasi point fixe avec une précision η : on considère qu'une variable x_i est CID(s_{CID}, η)-cohérente lorsque l'enveloppe convexe des bandelettes ne contracte aucune variable de plus de η. Néanmoins, les résultats expérimentaux suggèrent que la boucle de point fixe apporte peu de contraction, et que l'utilisation de 2 bandelettes ($s_{CID} = 2$) donne généralement de bons résultats.

Remarque 8 *La boucle de VariableCID peut être interrompue lorsque l'enveloppe convexe courante* $\boldsymbol{X}_\square$ *est égale à* $\boldsymbol{X}$ *sur toutes les dimensions sauf* X_i*. Aucune contraction sur* X_j *($j \neq i$) n'est alors possible.*

La figure 4.6 illustre la contraction de quatre bandelettes construites en découpant le domaine de x_1. Contrairement à un simple rognage de type 3B, l'union des quatre bandelettes contractées ($\boldsymbol{X_1'}, \boldsymbol{X_2'}, \boldsymbol{X_3'}, \boldsymbol{X_4'}$) permet de conserver de la contraction sur la variable x_2.

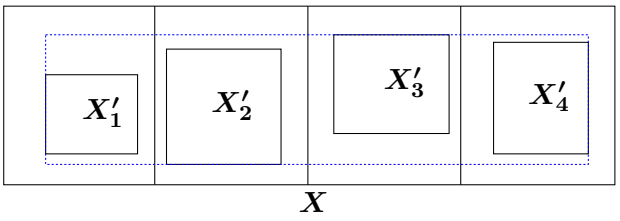


FIGURE 4.6 – Rognage CID pour $s_{CID} = 4$

L'algorithme hybride 3BCID [Trombettoni 07] combine le rognage CID et le rognage 3B, et remédie à la différence d'ordre de grandeur des deux paramètres s_{3B} et s_{CID} (en général, $s_{CID} \ll s_{3B}$). Il possède un pouvoir filtrant plus élevé que les rognages 3B et CID seuls.

Algorithme 9 Rognage CID

fonction VariableCID(**in-out** $\boldsymbol{X}$: boîte, i : composante, $\mathcal{C}$: système de contraintes, OC : contracteur, s_{CID} : nombre de bandelettes)
  $\boldsymbol{X}_\square \leftarrow \varnothing$
  pour $k \in \{1, \ldots, s_{CID}\}$ **faire** ▷ boucle sur l'ensemble des bandelettes
    $\boldsymbol{B}_k \leftarrow$ Bandelette$(\boldsymbol{X}, i, k, s_{CID})$
    $\boldsymbol{X}_\square \leftarrow \square(\boldsymbol{X}_\square, OC(\boldsymbol{B}_k, \mathcal{C}))$ ▷ enveloppe convexe
  fin pour
  $\boldsymbol{X} \leftarrow \boldsymbol{X}_\square$
fin fonction
fonction RognageCID(**in-out** $\boldsymbol{X}$: boîte, $\mathcal{C}$: système de contraintes, OC : contracteur, s_{CID} : nombre de bandelettes)
  pour $i \in \{1, \ldots, n\}$ **faire** ▷ boucle sur les variables
    VariableCID$(\boldsymbol{X}, i, \mathcal{C}, OC, s_{CID})$
  fin pour
fin fonction
fonction CID(**in-out** $\boldsymbol{X}$: boîte, $\mathcal{C}$: système de contraintes, OC : contracteur, s_{CID} : nombre de bandelettes)
  PointFixe$(\boldsymbol{X}, \mathcal{C},$ RognageCID$(\boldsymbol{X}, \mathcal{C}, OC, s_{CID}))$
fin fonction

4.4 Cohérences globales

Certains algorithmes de contraction exploitent les contraintes non pas tour à tour, mais simultanément ; ils procèdent généralement à une convexification du problème :

- les problèmes d'optimisation sous contraintes d'égalité (système carré) sont résolus par une version multivariée de l'algorithme de Newton (sous-section 4.4.1) ;
- les problèmes sous contraintes d'(in)égalité sont relaxés, puis un algorithme de programmation linéaire est invoqué sur le polytope des contraintes (sous-section 4.4.2).

4.4.1 Algorithme de Newton

L'algorithme de Newton par intervalles (sous-section 4.2.5) s'étend aux systèmes à n équations et n inconnues. Soient $g : \mathbb{R}^n \to \mathbb{R}^n$ un vecteur de contraintes différentiables, une boîte $\boldsymbol{X}$ et $\boldsymbol{x} \in \boldsymbol{X}$. Linéariser g au point $\boldsymbol{x}$ conduit à résoudre l'équation :

$$g(\boldsymbol{x}) + J_g(\boldsymbol{X})(\boldsymbol{X} - \boldsymbol{x}) = 0 \tag{4.24}$$

où J_g est une extension aux intervalles de la matrice jacobienne de g (la matrice des dérivées partielles du premier ordre de g). En posant $\boldsymbol{Y} = \boldsymbol{X} - \boldsymbol{x}$, l'équation 4.24 devient :

$$J_g(\boldsymbol{X})\boldsymbol{Y} = -g(\boldsymbol{x}) \tag{4.25}$$

Le système linéaire $A\boldsymbol{Y} = \boldsymbol{b}$ correspondant est alors résolu par l'algorithme de Gauss-Seidel sur intervalles.

4.4.2 Convexification

[Mentzer 91] minore la fonction objectif et les contraintes sur une boîte $\boldsymbol{X}$ en prenant un coin de la boîte comme point de développement en série de Taylor (figure 4.7).

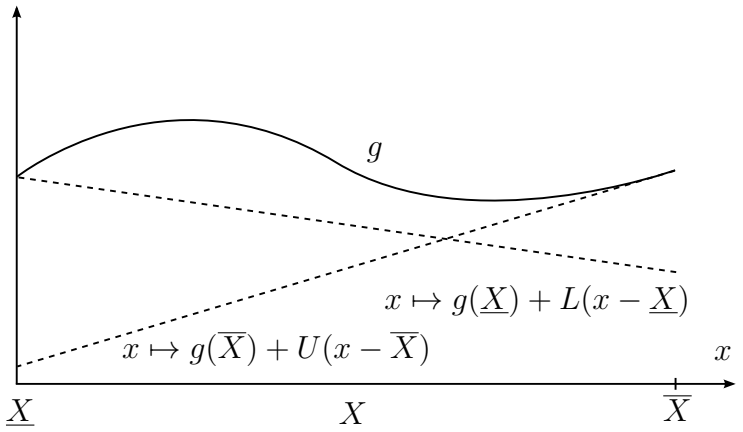


FIGURE 4.7 – Minorations linéaires de g par deux formes de Taylor $G_T(X, \underline{X})$ et $G_T(X, \overline{X})$

En notant $\frac{\partial G}{\partial x_i}(\boldsymbol{X})$ un encadrement de la $i$ème dérivée partielle de g sur $\boldsymbol{X}$ et en prenant (par exemple) le coin gauche comme point de développement, on obtient :

$$\forall \boldsymbol{x} \in \boldsymbol{X}, \quad g(\underline{\boldsymbol{X}}) + \sum_{i=1}^{n} \underline{\frac{\partial G}{\partial x_i}(\boldsymbol{X})} \cdot (x_i - \underline{X_i}) \leq g(\boldsymbol{x}) \tag{4.26}$$

Le problème relaxé $\mathcal{P}_{lb}$ (équation 4.27) est résolu par l'algorithme du simplexe. S'il est détecté non réalisable, il n'existe pas de solution au problème initial (le polytope délimité par les contraintes relaxées contient l'ensemble des solutions du problème initial). Sinon, la solution optimale de $\mathcal{P}_{lb}$ est un minorant du problème initial sur la boîte courante.

$$\begin{aligned} (\mathcal{P}_{lb}) \quad \min \quad & \sum_{i=1}^{n} \underline{\frac{\partial F}{\partial x_i}(\boldsymbol{X})} \cdot x_i \\ \text{s.c.} \quad & g_j(\underline{\boldsymbol{X}}) + \sum_{i=1}^{n} \underline{\frac{\partial G_j}{\partial x_i}(\boldsymbol{X})} \cdot (x_i - \underline{X_i}) \leq 0, \quad \forall j \in \{1, \ldots, m\} \\ & \underline{X_i} \leq x_i \leq \overline{X_i}, \quad \forall i \in \{1, \ldots, n\} \end{aligned} \tag{4.27}$$

[Yamamura 98, Yamamura 02] résout les NCSP en linéarisant les termes non-linéaires des contraintes et en invoquant l'algorithme du simplexe sur le système relaxé, avec une fonction objectif quelconque. Si aucune solution n'est trouvée, le système initial n'a pas de solution. Sinon, $2n$ appels supplémentaires au simplexe maximisent la borne gauche et minimisent la borne droite du domaine de chaque variable.

L'algorithme X-Newton [Araya 12] exploite les deux approches précédentes pour déterminer un minorant d'un problème d'optimisation sous contraintes et contracter les variables en $2n + 1$ appels au simplexe. Chaque inégalité est minorée par deux hyperplans, obtenus

en sélectionnant aléatoirement un coin de la boîte et son coin opposé. [Ninin 10b] relaxe le problème initial en utilisant l'AA, et invoque une seule fois l'algorithme du simplexe pour calculer un minorant du problème initial.

4.5 Contraction et différentiation automatique

Les similarités entre le double parcours de l'arbre syntaxique effectué par l'algorithme HC4Revise et par la DA en mode ajoint ont été pointées du doigt par [Schichl 05]. L'évaluation de l'arbre (parcours montant) est en effet commune aux deux algorithmes, et peut être effectuée une seule fois. [Schichl 05] pousse le concept plus loin en proposant d'effectuer le parcours descendant de la DA *après* le parcours descendant de propagation de la contrainte pour profiter de la contraction des nœuds *intermédiaires*.

L'exemple 13 illustre le gain de précision obtenu lorsque la DA de l'expression de la contrainte $x + (x + y)^2 - 1 = 0$ est effectuée après invocation d'un algorithme de type HC4Revise. L'évaluation classique des dérivées partielles sur la boîte $(X, Y) = ([0, 1], [0, 1])$ fournit $\frac{\partial G}{\partial x}(X, Y) = [1, 5]$ et $\frac{\partial G}{\partial y}(X, Y) = [0, 4]$. En profitant du parcours descendant de HC4Revise, l'évaluation des dérivées partielles est réduite à $\frac{\partial G}{\partial x}(X, Y) = [1, 3]$ et $\frac{\partial G}{\partial y}(X, Y) = [0, 2]$. Ici, exploiter la contrainte $x + y \in [0, 1]$, dite *contrainte implicite*, permet de calculer un encadrement plus étroit des dérivées partielles et d'améliorer les performances des techniques utilisant les dérivées (forme de Taylor, Mohc).

Exemple 13 *Soient :*
- $g(x, y) = x + (x + y)^2 - 1 = 0$ *une contrainte d'égalité ;*
- *les domaines* $X = [0, 5]$ *et* $Y = [0, 5]$.

L'évaluation du terme de gauche $x + (x + y)^2 - 1$ *en utilisant l'AI se décompose en :*

$$\begin{aligned} N_1 &= X + Y \in [0, 10] \\ N_2 &= N_1^2 \in [0, 100] \\ N_3 &= N_2 + X \in [0, 105] \\ N_4 &= N_3 - 1 \in [-1, 104] \end{aligned} \tag{4.28}$$

Intersectons N_4 *avec le terme de droite :* $N_4' = N_4 \cap [0, 0] = [0, 0]$ *et propageons la contrainte de haut en bas :*

$$\begin{aligned} N_3' &= N_3 \cap (1 + N_4') = [0, 105] \cap [1, 1] = [1, 1] \\ N_2' &= N_2 \cap (N_3' - X) = [0, 100] \cap ([1, 1] - [0, 5]) = [0, 100] \cap [-4, 1] = [0, 1] \\ N_1' &= N_1 \cap \square \left((-\sqrt{N_2'}), \sqrt{N_2}\right) = [0, 10] \cap [-1, 1] = [0, 1] \\ X' &= X \cap (N_3' - N_2') = [0, 5] \cap (1 - [0, 1]) = [0, 5] \cap [0, 1] = [0, 1] \\ X' &= X' \cap (N_1' - Y) = [0, 1] \cap ([0, 1] - [0, 5]) = [0, 1] \cap [-5, 1] = [0, 1] \\ Y' &= Y \cap (N_1' - X') = [0, 5] \cap ([0, 1] - [0, 1]) = [0, 5] \cap [-1, 1] = [0, 1] \end{aligned} \tag{4.29}$$

Les domaines de x et y sont tous deux contractés à $X' = Y' = [0,1]$. Le domaine du nœud $N_1 = X+Y$ a été contracté à $N_1' = [0,1]$, alors que l'évaluation directe de la somme $X'+Y'$ produit $[0,1]+[0,1] = [0,2] \supset [0,1] = N_1'$.

La figure 4.8 illustre l'encadrement par la boîte $([0,1],[0,1])$ de la relation $x+y \in [0,1]$ obtenue après contraction. Le domaine exact, en mauve, est un polytope dont les arêtes ne sont pas parallèles aux axes, et n'est donc pas représentable exactement par une boîte. L'approche de [Schichl 05] exploite l'information disponible à partir des équations $\{x+y = [0,1], x = [0,1], y = [0,1]\}$, alors qu'une étape standard de DA évaluerait la somme $X+Y$ à $[0,1]+[0,1] = [0,2]$.

Finalement, l'évaluation des dérivées partielles sur le polytope donne :

$$\frac{\partial G}{\partial x}(X,Y) = 1 + 2(X+Y) = 1 + 2\boldsymbol{N_1'} = 1 + 2[0,1] = [1,3] \subset [1,5] \tag{4.30}$$

$$\frac{\partial G}{\partial y}(X,Y) = 2(X+Y) = 2\boldsymbol{N_1'} = 2[0,1] = [0,2] \subset [0,4] \tag{4.31}$$

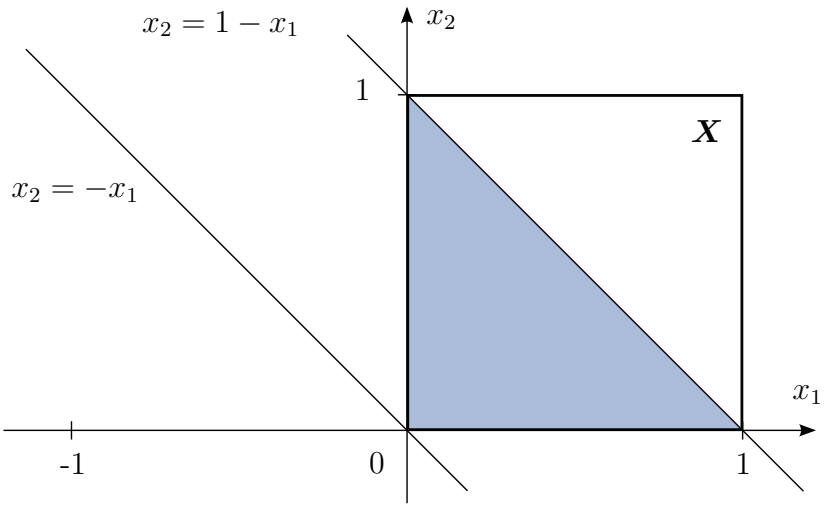


FIGURE 4.8 – Polytope de la contrainte encadré par une boîte

Nous suggérons dans l'exemple 14 que la technique de [Schichl 05] permet parfois d'obtenir un meilleur minorant du problème par les techniques de programmation linéaire.

Exemple 14 *Soit le problème d'optimisation sous contraintes :*

$$\begin{array}{ll} \min\limits_{(x,y)} & f(x,y) = -x - 2y \\ s.c. & g(x,y) = x + (x+y)^2 - 1 = 0 \\ & x \in X = [0,5] \\ & y \in Y = [0,5] \end{array} \tag{4.32}$$

défini sur la boîte $\boldsymbol{X} = (X, Y) = ([0,5],[0,5])$. *Une exécution de HC4Revise par rapport à la contrainte* $g = 0$ *permet de réduire le domaine des variables à* $X = Y = [0,1]$ *(voir exemple 13).*

Déterminer un minorant du problème par des techniques de programmation linéaire nécessite de calculer une relaxation linéaire de g. *Puisque la contrainte d'égalité* $g = 0$ *est équivalente au système de contraintes d'inégalité* $\{g \leq 0, -g \leq 0\}$, *on peut encadrer* g *par deux hyperplans (formes de Taylor en coin), l'un minorant* g *et l'autre minorant* $-g$ *:*

$$\begin{array}{ll} \forall \boldsymbol{x} \in \boldsymbol{X}, & g(\underline{X},\underline{Y}) + L_1(x - \underline{X}) + L_2(y - \underline{Y}) \quad \leq g(\boldsymbol{x}) \\ \forall \boldsymbol{x} \in \boldsymbol{X}, & -g(\underline{X},\underline{Y}) - U_1(x - \underline{X}) - U_2(y - \underline{Y}) \quad \leq -g(\boldsymbol{x}) \end{array} \tag{4.33}$$

où les dérivées partielles de g *sur* $\boldsymbol{X}$ *sont calculées en exploitant le parcours descendant de HC4Revise :*

$$\begin{aligned} \frac{\partial g}{\partial x}(x,y) &= 1 + 2(x+y) \in 1 + 2[0,1] = [1,3] =: [L_1, U_1] \\ \frac{\partial g}{\partial y}(x,y) &= 2(x+y) \in 2[0,1] = [0,2] =: [L_2, U_2] \end{aligned} \tag{4.34}$$

La contrainte d'égalité relaxée s'écrit alors sous la forme de deux inégalités :

$$\begin{array}{ll} \forall \boldsymbol{x} \in \boldsymbol{X}, & x \leq 1 \\ \forall \boldsymbol{x} \in \boldsymbol{X}, & -3x - 2y \leq -1 \end{array} \tag{4.35}$$

Le polytope correspondant est représenté sur la figure 4.9. La solution optimale du problème relaxé, calculée par l'algorithme du simplexe, est le point $(1,1)$, *d'évaluation* $f(1,1) = -3$.

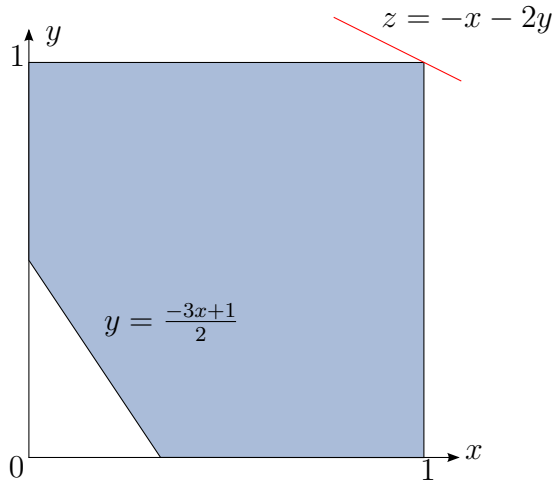


FIGURE 4.9 – Polytope du problème relaxé

L'information fournie par le domaine $[0,1]$ *du nœud* $x + y$ *n'est pourtant pas totalement exploitée : l'ajout de la contrainte implicite* $x + y \in [0,1]$ *au système de contraintes initial*

peut permettre d'obtenir un meilleur minorant du problème initial. Si la contrainte $0 \leq x+y$ *est redondante, la contrainte* $x + y \leq 1$ *conduit à une réduction significative du polytope (figure 4.10).*

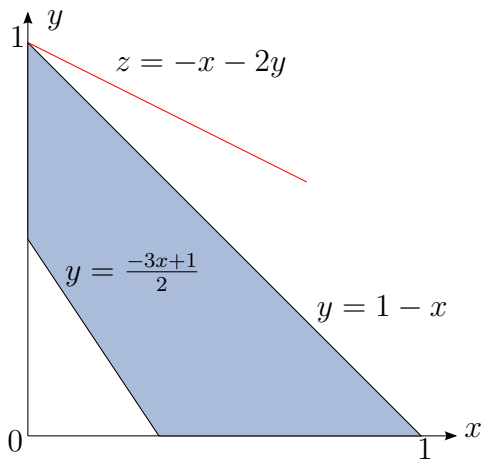


FIGURE 4.10 – Réduction du polytope par l'ajout d'une contrainte implicite

Le simplexe trouve comme nouvelle solution optimale le point $(0, 1)$*, dont l'évaluation est meilleure que la solution précédente :* $f(0, 1) = -2 > -3$.

Les méthodes de programmation linéaire, consistant à calculer une relaxation linéaire du problème puis à résoudre le problème relaxé, constituent un outil puissant pour calculer des minorants du problème *en tenant compte des contraintes*. Nous avons montré que parfois, les algorithmes de propagation de contraintes (par exemple de type HC4Revise) travaillant sur l'arbre syntaxique des fonctions permettent de réduire les domaines des nœuds intermédiaires. Ces nouvelles contraintes (que nous appelons implicites) portant sur les variables du problème peuvent permettre de réduire le polytope du problème relaxé lorsqu'elles sont ajoutées au système de contraintes initial. En ce sens, cette technique est proche de la méthode de [Lebbah 05a] basée sur une décomposition des contraintes en contraintes élémentaires et une relaxation convexe sur chacun des nœuds intermédiaires.

Chapitre 5

Charibde

Sommaire

5.1 Hybridation de techniques d'optimisation 66
5.1.1 Méthodes intégratives 67
5.1.2 Méthodes coopératives 67
5.1.3 Charibde : une approche coopérative 68
5.2 Branch and contract par intervalles 70
5.2.1 Contracteur pour l'optimisation sans contraintes 71
5.2.2 Contracteur pour l'optimisation sous contraintes 73
5.2.3 MaxDist : une heuristique de recherche géométrique 76
5.3 Evolution différentielle 78
5.3.1 Gestion rigoureuse de la fonction objectif 79
5.3.2 Gestion rigoureuse des contraintes 81
5.3.3 Evaluation par comparaison 81
5.3.4 Exploitation du domaine sur intervalles 84
5.4 Comparaison de solveurs sur la base COCONUT 92
5.4.1 Base de comparaison 92
5.4.2 Comparaison des résultats 93
5.5 Nouveaux minima globaux de problèmes difficiles 96
5.5.1 Fonctions de test 96
5.5.2 Résultats expérimentaux 98
5.5.3 Bénéfices de l'hybridation 101
5.5.4 Comparaison de solveurs 107
5.6 Conclusion 109

Les méthodes d'optimisation globale précédemment évoquées semblent traiter de classes de problèmes a priori différentes : les méthodes par intervalles fournissent une preuve numérique d'optimalité, mais sont limitées à des instances de quelques dizaines de variables, tandis que les AE excellent sur les problèmes multimodaux de grande taille pour lesquels les méthodes conventionnelles peinent à converger.

Dans ce chapitre, nous discutons de schémas de synergie visant à réconcilier les deux approches afin de profiter de leurs avantages :

1. explorer rapidement l'espace de recherche ;
2. éviter de rester piégé dans des minima locaux ;
3. éliminer les sous-espaces sous-optimaux ou non réalisables ;
4. certifier l'optimalité de la solution.

et traiter des problèmes jusqu'ici jugés comme insolubles.

La section 5.1 décrit les méthodes hybrides existantes et situe le contexte de l'étude. Nous introduisons Charibde, un algorithme hybride coopératif combinant une méthode déterministe et une méthode stochastique. Les sections 5.2 et 5.3 détaillent l'implémentation des composantes de Charibde. Dans la section 5.4, nous comparons Charibde à des solveurs de pointe sur une base de problèmes contraints difficiles. De nouveaux résultats optimaux sont fournis pour cinq problèmes multimodaux dans la section 5.5.

5.1 Hybridation de techniques d'optimisation

L'hybridation de techniques d'optimisation consiste à répartir différentes tâches de recherche à plusieurs méthodes (généralement deux). Nous traitons par la suite des hybridations hétérogènes, c'est-à-dire hybridant une métaheuristique et un autre algorithme. La taxinomie des métaheuristiques hybrides a été abordée par plusieurs auteurs [Talbi 02, Puchinger 05, Alba 05, Raidl 06, Jourdan 09] ; nous présentons ici une classification hiérarchique récapitulant les hybridations possibles entre techniques d'optimisation discrète ou continue.

[Puchinger 05] retient trois critères à étudier : la nature de l'hybridation (avec une métaheuristique, un méthode de l'intelligence artificielle ou de la recherche opérationnelle), le niveau d'hybridation (échange de composants) et l'ordre d'exécution (exécutions séquentielles ou parallèles). Il en résulte la classification suivante :

1. les méthodes intégratives ou bas niveau : un opérateur donné de la métaheuristique est remplacé par un autre algorithme (recherche locale, méthodes exactes) suivant un schéma maître-esclave. On distingue :
   - le cas où la métaheuristique est maître ;
   - le cas où la métaheuristique est esclave.
2. les méthodes coopératives ou haut niveau : les algorithmes ne sont pas inclus l'un dans l'autre, mais échangent des informations. On distingue :
   - les exécutions séquentielles (HRH) : l'une des deux méthodes est exécutée avant l'autre (similaire à un préprocessing) ;
   - les exécutions parallèles ou entrecroisées (HCH) : les deux méthodes sont indépendantes.

On parle d'hybridation *globale* lorsque les méthodes explorent l'intégralité de l'espace de recherche et d'hybridation *partielle* lorsqu'elles se limitent à un sous-espace. L'hybridation

est dite *générale* si les algorithmes travaillent tous à résoudre le même problème, et *spécialisée* s'ils travaillent sur des problèmes différents.

Les sous-sections 5.1.1 et 5.1.2 présentent certaines hybridations de la littérature, respectivement intégratives et coopératives.

5.1.1 Méthodes intégratives

Les méthodes intégratives incorporent un algorithme à un autre de manière subordonnée (relation maître-esclave), en remplaçant une fonction particulière par une autre.

Métaheuristique esclave

[Zhang 07] intègrent un AG dans un algorithme de BBI. L'AG fournit la direction suivant laquelle les boîtes sont partitionnées, et un individu est généré dans chaque sous-boîte. Lorsqu'une boîte est éliminée, les individus appartenant à cette boîte sont éliminés de la population courante. La meilleure évaluation met à jour le meilleur majorant du minimum global à chaque génération.

Métaheuristique maître

Les algorithmes mémétiques [Moscato 04] sont des algorithmes évolutionnaires utilisant différents types de recherche locale (descente de gradient, recherche taboue, Nelder-Mead, recuit simulé) pour améliorer localement des individus obtenus par des opérateurs de diversification.

Dans [Cotta 03], l'opérateur de croisement est remplacé par un algorithme de BB qui considère tous les descendants pouvant être générés exclusivement à partir des parents, et sélectionne la meilleure combinaison possible.

5.1.2 Méthodes coopératives

Les méthodes coopératives se distinguent par la granularité de l'hybridation, l'implantation matérielle, la mémoire (partagée ou distribuée) et la synchronisation des processus.

Exécutions séquentielles

[Feltl 04] résolvent un problème discret d'affectation généralisé avec un AG. Une première phase consiste à résoudre une relaxation linéaire du problème avec CPLEX, puis à arrondir la solution optimale pour générer des individus prometteurs. Des opérateurs de réparation stochastiques permettent de construire des individus réalisables.

[Sotiropoulos 97] combinent un algorithme de BBI et un AG. Le BBI génère une liste $\mathcal{L}$ de boîtes indécidables, de taille au plus δ. La population de l'AG est alors initialisée en générant un individu dans chacune des boîtes de $\mathcal{L}$. Afin de conserver une taille de population inférieure à 50 individus, des tests numériques ont suggéré que $\delta \in [10^{-3}, 10^{-1}]$.

Les auteurs n'utilisent pas le test du point milieu : le meilleur majorant $\tilde{f}$ du minimum global f^* est mis à jour uniquement en considérant la borne (grossière) $\overline{F(\boldsymbol{X})}$.

Exécutions parallèles

[Gallardo 07] (hybridation d'un BB et d'un algorithme mémétique) et [Blum 11] (hybridation d'une recherche en faisceau et d'un algorithme mémétique) décrivent des stratégies parallèles similaires : la méthode exacte identifie des régions prometteuses de l'espace de recherche, explorées ensuite par la métaheuristique. L'algorithme mémétique fournit un majorant du minimum global à l'algorithme arborescent afin d'élaguer l'espace de recherche, et est guidé en retour vers des régions prometteuses de l'espace de recherche.

[Cotta 95] évoque la possibilité de combiner en parallèle un BB et un AG sur le problème du voyageur de commerce. L'AG fournit au BB un majorant du minimum global, afin d'éliminer les sous-problèmes sous-optimaux. Le BB injecte dans la population de l'AG les circuits qu'il considère prometteurs. Les auteurs critiquent néanmoins cette approche : échanger des informations entre des processus s'exécutant à des vitesses différentes est une tâche difficile. En particulier, injecter des solutions prometteuses en début de convergence de l'AG risque de faire apparaître des « superindividus » et de réduire la diversité de la population. Deux approches alternatives sont considérées : une hybridation intégrative (remplacer le croisement de l'AG par une recherche arborescente) ou un schéma maître-esclaves (composé d'un BB et de m AG en parallèle).

[Alliot 12a] hybrident un AG et un BBI exécutés dans des processus indépendants et qui communiquent via une mémoire partagée. L'AG explore rapidement le domaine à la recherche d'une bonne solution. Celle-ci est mise à disposition du BBI dans la mémoire partagée afin d'améliorer le meilleur majorant connu du minimum global. Le BBI élague l'espace de recherche en éliminant plus efficacement des sous-espaces ne pouvant contenir de solution optimale. Une solution ponctuelle améliorant le meilleur majorant connu est stockée en mémoire partagée par le BBI, puis intégrée à la population de l'AG pour empêcher la convergence prématurée de la population vers des minima locaux. Un troisième processus, déclenché périodiquement, projette les individus hors domaine (n'appartenant pas à une boîte du BBI) dans la boîte la plus proche. L'algorithme adopte un cadre très général : peu de conditions sont requises sur la fonction à optimiser (continuité, différentiabilité et décomposabilité ne sont pas nécessaires) ; la fonction objectif doit être calculable, c'est-à-dire que l'on doit disposer d'une procédure de calcul d'inclusion (éventuellement elle-même boîte noire). Les auteurs présentent de nouveaux résultats optimaux pour la fonction Michalewicz ($n = 12$) et la fonction Griewank ($n = 6$) à laquelle est appliquée une rotation.

5.1.3 Charibde : une approche coopérative

Nos travaux exploitent le schéma coopératif introduit dans [Alliot 12a]. Néanmoins, l'efficacité (et la fiabilité) de leur approche reste très limitée :

- les techniques d'intervalles utilisées restent naïves et peu compétitives avec les solveurs de pointe ;

– la projection des individus hors domaine est peu efficace ;
– l'AG n'est pas fiabilisé, c'est-à-dire que l'évaluation du meilleur individu transmise au BBI peut être sujette à des erreurs d'arrondis ;
– les boîtes dont le diamètre est inférieur à un seuil ε_x sont éliminées sans être davantage explorées : des solutions peuvent être perdues.

Notre algorithme hybride **Charibde** (Cooperative Hybrid Algorithm using Reliable Interval-Based methods and Differential Evolution) [Vanaret 13] combine un algorithme à ED et un algorithme de BCI. Bien qu'intégrant des composants stochastiques, Charibde est un solveur *totalement fiable*.

L'ED, privilégiée à l'AG en raison de ses performances convaincantes sur les problèmes continus et son faible nombre d'hyperparamètres, communique avec un BCI bénéficiant des dernières contributions de la communauté de PPCI. Une heuristique d'exploration avancée réduit périodiquement le domaine de l'ED. Bornes, solutions et espace de recherche sont échangés par passage de messages MPI (figure 5.1).

MPI (The Message Passing Interface) est une norme permettant d'exécuter des programmes parallèles sur des ordinateurs multiprocesseurs via une bibliothèque de fonctions (dans les langages C, C++ et Fortran). Basée sur des points de synchronisation, MPI met à disposition des fonctions d'envoi et de réception bloquants (mise en attente du processus tant que le message n'a pas été envoyé ou reçu) ou immédiats (retour immédiat au déroulement du programme après l'envoi ou la réception du message), ainsi que des fonctions de test de disponibilité d'un message.

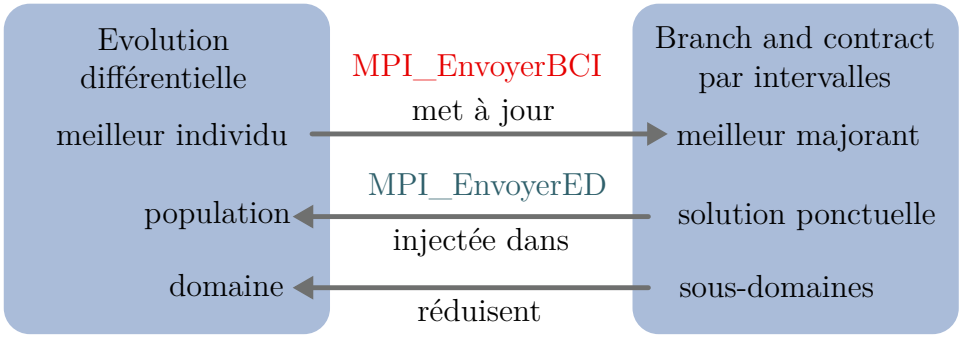


FIGURE 5.1 – Schéma de coopération de Charibde

Les sections 5.2 et 5.3 détaillent respectivement les implémentations de l'algorithme de BCI et de l'algorithme à ED de Charibde. Charibde est comparé à des solveurs performants sur des problèmes de test difficiles dans la section 5.4. De nouveaux minima globaux certifiés par Charibde pour cinq problèmes multimodaux difficiles sont fournis avec les solutions correspondantes dans la section 5.5.

5.2 Branch and contract par intervalles

L'algorithme 10 décrit le BCI implémenté en OCaml dans Charibde. Il intègre des routines de communication avec l'ED par passage de messages MPI : à chaque itération (avant l'extraction d'une boîte), le BCI vérifie avec un test non bloquant si une solution de l'ED est disponible. Si c'est le cas, son évaluation met à jour le meilleur majorant connu $\tilde{f}$. Lorsque l'évaluation d'une solution ponctuelle (par exemple, le centre d'une boîte) améliore $\tilde{f}$, elle est transmise à l'ED et intégrée à sa population.

La file de priorité $\mathcal{L}$ est implémentée par un maximier, c'est-à-dire un arbre complet à gauche (un tas) tel que la priorité d'un nœud est supérieure ou égale à la priorité de chacun de ses fils. L'extraction et l'insertion d'une boîte se font en temps logarithmique.

Algorithme 10 Charibde : algorithme de branch and contract par intervalles

fonction BCI($\boldsymbol{X_0}$: boîte initiale, F : fonction objectif, $\mathcal{C}$: système de contraintes)
  $\tilde{f} \leftarrow +\infty$ ▷ meilleur majorant connu
  $\mathcal{L} \leftarrow \{\boldsymbol{X}_0\}$ ▷ file de priorité
  tant que $\mathcal{L} \neq \varnothing$ **faire**
    $(\boldsymbol{x}_{ED}, f_{ED}) \leftarrow$ MPI_RecevoirED()
    $\tilde{f} \leftarrow \min(\tilde{f}, f_{ED})$
    Extraction d'une boîte $\boldsymbol{X}$ de $\mathcal{L}$
    Contraction de $\boldsymbol{X}$ par rapport à $\tilde{f}$ et aux contraintes ▷ algorithme 11 ou 15
    si $\boldsymbol{X}$ ne peut pas être éliminée **alors**
      si $f(m(\boldsymbol{X})) < \tilde{f}$ **alors** ▷ mise à jour du meilleur majorant
        $\tilde{f} \leftarrow f(m(\boldsymbol{X}))$
        MPI_EnvoyerED($m(\boldsymbol{X}), f(m(\boldsymbol{X}))$)
      fin si
      Découpe de $\boldsymbol{X}$ en $\{\boldsymbol{X}_1, \boldsymbol{X}_2\}$
      Insertion de $\{\boldsymbol{X}_1, \boldsymbol{X}_2\}$ dans $\mathcal{L}$
    fin si
  fin tant que
  renvoyer $(\tilde{f}, \tilde{\boldsymbol{x}})$
fin fonction

A chaque extraction de boîte, Charibde fait appel à une procédure de contraction dont cette section détaille les différentes briques : le contracteur pour les problèmes non contraints (ou contraints par des bornes) est décrit dans la sous-section 5.2.1, et le contracteur pour les problèmes contraints est décrit dans la sous-section 5.2.2. Si une incohérence par rapport aux contraintes est détectée dans un contracteur, un mécanisme d'exception interrompt le traitement de la boîte. Ces procédures de contraction sont applicables aux fonctions décomposables (d'expression analytique connue) dont on peut calculer un encadrement du gradient.

5.2.1 Contracteur pour l'optimisation sans contraintes

Cette sous-section s'inscrit dans le cadre des problèmes d'optimisation sans contraintes (ou sous contraintes de bornes), dont nous rappelons la forme standard :

$$(\mathcal{P}) \quad \min_{\boldsymbol{x} \in \boldsymbol{X} \subset \mathbb{R}^n} \quad f(\boldsymbol{x}) \tag{5.1}$$

Le contracteur implémenté dans Charibde est détaillé dans l'algorithme 11.

Algorithme 11 Contracteur pour l'optimisation sans contraintes

fonction CONTRACTION(**in-out** $\boldsymbol{X}$: boîte, F : fonction objectif, $\tilde{f}$: meilleur majorant)
  $F_{\boldsymbol{X}} \leftarrow$ HC4REVISE$(F(\boldsymbol{X}) \leq \tilde{f})$ ▷ évaluation de f/contraction
  $lb \leftarrow \underline{F_{\boldsymbol{X}}}$ ▷ minorant par évaluation naturelle
  $\boldsymbol{G} \leftarrow \nabla F(\boldsymbol{X})$ ▷ gradient par DA
  si évaluation au deuxième ordre **alors**
    $lb \leftarrow$ DEUXIÈMEORDRE$(\boldsymbol{X}, F, lb, \tilde{f}, \boldsymbol{G})$ ▷ algorithme 12
  fin si
  TESTMONOTONIE$(\boldsymbol{X}, F, \boldsymbol{G})$ ▷ algorithme 14
  renvoyer lb
fin fonction

Calcul de minorants

Le calcul de minorants de bonne qualité est d'une importance cruciale dans un algorithme de BB afin d'élaguer les sous-espaces non réalisables ou sous-optimaux. Les méthodes d'intervalles les plus simples (évaluation naturelle) fournissent souvent un encadrement grossier de l'image d'une fonction sur une boîte. Les formes de deuxième ordre offrent une alternative relativement peu coûteuse à la forme naturelle pour le calcul de minorants. Rappelons que l'ordre de convergence d'une fonction d'inclusion F indique la vitesse à laquelle F approche l'image exacte de la fonction ; les formes de deuxième ordre deviennent en général plus précises que la forme naturelle lorsque la largeur de l'intervalle approche zéro.

L'algorithme 12 décrit le calcul d'un minorant par la forme optimale de Baumann (voir équation 3.23). Intersecter l'évaluation naturelle et l'évaluation de deuxième ordre revient à conserver le minorant le plus élevé. Dans notre implémentation, l'évaluation $f_{\boldsymbol{c}_B} := \overline{F(\boldsymbol{c}_B)}$ du centre optimal de Baumann $\boldsymbol{c}_B$, utilisée dans le calcul du minorant lb_B, permet de mettre à jour le meilleur majorant connu $\tilde{f}$ lorsque $\boldsymbol{c}_B$ est un point réalisable.

Test de monotonie

L'algorithme 13 décrit le test de monotonie implémenté dans Charibde pour les problèmes non contraints ou sous contraintes de bornes : lorsque certaines composantes X_i de la boîte courante $\boldsymbol{X}$ sont au bord du domaine initial $\boldsymbol{D}$ (c'est-à-dire lorsque $\underline{X_i} = \underline{D_i}$ ou $\overline{X_i} = \overline{D_i}$),

Algorithme 12 Calcul d'un minorant par une forme de Baumann optimale

fonction DEUXIÈMEORDRE($\boldsymbol{X}$: boîte, F : fonction objectif, lb : minorant, **in-out** $\tilde{f}$: meilleur majorant, $\boldsymbol{G}$: gradient de F sur $\boldsymbol{X}$)
  $(lb_B, \boldsymbol{c}_B, f_{\boldsymbol{c}_B}) \leftarrow$ BAUMANN($\boldsymbol{X}, F, \boldsymbol{G}$) ▷ forme de Baumann
  si $\tilde{f} < lb_B$ **alors**
    éliminer $\boldsymbol{X}$
  sinon
    $lb \leftarrow \max(lb, lb_B)$ ▷ mise à jour du minorant
    si $f_{\boldsymbol{c}_B} < \tilde{f}$ et $\boldsymbol{c}_B$ réalisable **alors**
      $\tilde{f} \leftarrow f_{\boldsymbol{c}_B}$ ▷ mise à jour du majorant
    fin si
  fin si
  renvoyer lb
fin fonction

il est parfois possible d'éliminer $\boldsymbol{X}$ ou de réduire les domaines des variables à une de leurs bornes. En effet, les composantes d'un minimum local sur la boîte $\boldsymbol{X}$ sont soit stationnaires, soit sur un bord du domaine :

- lorsque $\underline{X_i} = \underline{D_i}$ et
  - f est croissante par rapport à x_i sur $\boldsymbol{X}$, X_i peut être réduit à sa borne gauche ;
  - f est strictement décroissante par rapport à x_i sur $\boldsymbol{X}$, $\boldsymbol{X}$ peut être éliminée ;
- lorsque $\overline{X_i} = \overline{D_i}$ et
  - f est strictement croissante par rapport à x_i sur $\boldsymbol{X}$, $\boldsymbol{X}$ peut être éliminée ;
  - f est décroissante par rapport à x_i sur $\boldsymbol{X}$, X_i peut être réduit à sa borne droite.

Si une composante X_i de $\boldsymbol{X}$ est intérieure au domaine (c'est-à-dire si $X_i \subset int(D_i)$), le minimum de f sur $\boldsymbol{X}$ correspond à un point où $\frac{\partial F}{\partial x_i}(\boldsymbol{X})$ s'annule. Si la dérivée partielle par rapport à x_i (calculée dans l'algorithme 11) ne contient pas 0, $\boldsymbol{X}$ ne peut contenir de minimum global. Lorsque $0 \in \frac{\partial F}{\partial x_i}(\boldsymbol{X})$, un algorithme de cohérence 2B, peu coûteux, est invoqué sur la contrainte $\frac{\partial F}{\partial x_i}(\boldsymbol{X}) = 0$ afin d'éliminer des points non stationnaires. L'algorithme 13 est invoqué comme procédure de révision d'un point fixe (algorithme 14).

Notons que la propagation de contraintes sur les dérivées partielles (par rapport aux contraintes $\frac{\partial F}{\partial x_i} = 0$) semble être une pratique peu répandue dans la communauté d'optimisation globale. Elle apporte pourtant une contraction substantielle de l'espace de recherche.

Algorithme 13 Test de monotonie pour les problèmes sous contraintes de bornes

fonction MinimumLocal(**in-out** $\boldsymbol{X}$: boîte, F : fonction objectif, $\boldsymbol{G} = (G_1, \ldots, G_n)$: gradient de F sur $\boldsymbol{X}$)
  pour $i \in \{1, \ldots, n\}$ **faire**
    si X_i au bord du domaine **alors**
      si G_i est de signe constant **alors**
        réduire X_i à une borne ou éliminer $\boldsymbol{X}$
      fin si
    sinon si $0 \notin G_i$ **alors**
      éliminer $\boldsymbol{X}$
    sinon
      HC4Revise($\frac{\partial F}{\partial x_i}(\boldsymbol{X}) = 0$)
    fin si
  fin pour
fin fonction

Algorithme 14 Boucle de contraction pour les problèmes sous contraintes de bornes

fonction TestMonotonie(**in-out** $\boldsymbol{X}$: boîte, F : fonction objectif, $\boldsymbol{G}$: gradient de F sur $\boldsymbol{X}$)
  PointFixe(MinimumLocal($\boldsymbol{X}, F, \boldsymbol{G}$)) ▷ algorithmes 7 et 13
fin fonction

5.2.2 Contracteur pour l'optimisation sous contraintes

Cette sous-section s'inscrit dans le cadre des problèmes d'optimisation sous contraintes, dont nous rappelons la forme standard :

$$(\mathcal{P}) \quad \begin{array}{lll} \min\limits_{\boldsymbol{x} \in \mathbb{R}^n} & f(\boldsymbol{x}) & \\ s.c. & g_i(\boldsymbol{x}) \leq 0, & i \in \{1, \ldots, m\} \\ & h_j(\boldsymbol{x}) = 0, & j \in \{1, \ldots, p\} \end{array} \tag{5.2}$$

Lorsque le problème est sujet à des contraintes d'égalité h_j ($j \in \{1, \ldots, p\}$), deux approches différentes sont implémentées par les solveurs sur intervalles actuels :

- GlobSol [Kearfott 96b] et Icos [Lebbah 05b] convergent vers une boîte $\boldsymbol{X}$ de faible diamètre, garantie de contenir un vecteur réel $\boldsymbol{x}$ qui minimise f et vérifie les contraintes :

$$\begin{cases} g_i(\boldsymbol{x}) \leq 0, & i \in \{1, \ldots, m\} \\ h_j(\boldsymbol{x}) = 0, & j \in \{1, \ldots, p\} \end{cases} \tag{5.3}$$

  L'existence de $\boldsymbol{x}$ dans $\boldsymbol{X}$ est prouvée numériquement par un algorithme de Newton par intervalles multivarié ;
- IBBA [Ninin 10b], Ibex [Trombettoni 11] et Charibde traitent un problème relaxé où chaque contrainte d'égalité $h_j(\boldsymbol{x}) = 0$ ($j \in \{1, \ldots, p\}$) est remplacée par deux

inégalités :

$$-\varepsilon_{=} \leq h_j(\boldsymbol{x}) \leq \varepsilon_{=} \tag{5.4}$$

$\varepsilon_{=}$ peut être choisi arbitrairement petit.

Le contracteur implémenté dans Charibde est détaillé dans l'algorithme 15. Différentes techniques de calcul de minorants, actives par défaut dans Charibde et décrites dans les sous-sections suivantes, peuvent être désactivées par l'utilisateur.

Algorithme 15 Contracteur pour l'optimisation sous contraintes

fonction CONTRACTION(**in-out** $\boldsymbol{X}$: boîte, F : fonction objectif, **in-out** $\tilde{f}$: meilleur majorant, **in-out** $\mathcal{C}$: système de contraintes)
  $lb \leftarrow -\infty$ ▷ minorant
  répéter
    $\boldsymbol{X}' \leftarrow \boldsymbol{X}$
    $F_{\boldsymbol{X}} \leftarrow$ HC4REVISE$(F(\boldsymbol{X}) \leq \tilde{f})$ ▷ évaluation de f/contraction
    $lb \leftarrow \underline{F_{\boldsymbol{X}}}$ ▷ minorant par évaluation naturelle
    $\boldsymbol{G} \leftarrow \nabla F(\boldsymbol{X})$ ▷ gradient par DA
    si évaluation au deuxième ordre **alors**
      $lb \leftarrow$ DEUXIÈMEORDRE$(\boldsymbol{X}, F, lb, \tilde{f}, \boldsymbol{G})$ ▷ algorithme 12
    fin si
    si évaluation par monotonie **alors**
      $lb \leftarrow$ CONTRACTIONPARMONOTONIE$(\boldsymbol{X}, F, lb, \tilde{f}, \mathcal{C})$ ▷ algorithme 17
    fin si
    $\mathcal{C} \leftarrow$ HC4$(\boldsymbol{X}, \mathcal{C})$ ou MOHC$(\boldsymbol{X}, \mathcal{C})$ ▷ point fixe de contraction
    si convexification **alors**
      $lb \leftarrow$ CONVEXIFICATION$(\boldsymbol{X}, F, lb, \tilde{f}, \boldsymbol{G}, \mathcal{C})$ ▷ algorithme 16
    fin si
  jusqu'à $\boldsymbol{X} = \varnothing$ ou $gain(\boldsymbol{X}, \boldsymbol{X'}) < \eta$
  renvoyer lb
fin fonction

Contracteur basé sur la programmation linéaire

A l'instar de X-Newton [Araya 12], notre contracteur linéaire est basé sur une relaxation de la fonction objectif et des contraintes par une forme de Taylor en coin (voir sous-section 4.4) utilisant la variante récursive de [Hansen 68]. Par défaut dans Charibde, l'algorithme du simplexe est invoqué une seule fois pour obtenir un minorant du problème (algorithme 16) ; $2n$ appels supplémentaires peuvent également contracter les domaines des variables.

Contrairement à la relaxation par AA ou l'algorithme de Newton multivarié, à la convergence quadratique, l'approche adoptée converge linéairement vers la solution du problème initial. Néanmoins, l'AA est coûteuse en évaluations par intervalles et complexe à implémenter, et l'algorithme de Newton manipule un système carré d'équations qui nécessite parfois d'être préconditionné.

Algorithme 16 Calcul d'un minorant et contraction par un contracteur linéaire

fonction CONVEXIFICATION(**in-out** $\boldsymbol{X}$: boîte, F : fonction objectif, lb : minorant, $\tilde{f}$: meilleur majorant, $\boldsymbol{G}$: gradient de F sur $\boldsymbol{X}$, $\mathcal{C}$: système de contraintes)
  $polytope \leftarrow$ TAYLORENCOIN($\boldsymbol{X}, \mathcal{C}$) ▷ relaxation linéaire des contraintes
  $lb_{lp} \leftarrow$ SIMPLEXE($\boldsymbol{X}, G, polytope$)
  si $\tilde{f} < lb_{lp}$ **alors**
    éliminer $\boldsymbol{X}$
  sinon
    $lb \leftarrow \max(lb, lb_{lp})$
    si X-Newton **alors**
      X-NEWTON($\boldsymbol{X}, polytope$)
    fin si
  fin si
  renvoyer lb
fin fonction

Charibde fait appel au binding ocaml-glpk [Mimram 04] vers GLPK (GNU Linear Programming Kit), une bibliothèque de résolution de problèmes linéaires de grande taille et de problèmes mixtes en AF. Une étape de postprocessing [Neumaier 04] (voir annexe A.3) est appliquée après chaque exécution du simplexe afin de déterminer un minorant fiable grâce à l'AI.

Il est suggéré dans [Araya 12] de calculer le minorant lb_{lp} en résolvant le programme linéaire dans son intégralité, puis de résoudre les $2n$ appels suivants en prenant comme base réalisable initiale la solution de l'appel précédent (réduisant donc le simplexe à sa phase II). Malheureusement, ocaml-glpk n'implémente pas cette fonctionnalité, ce qui rend en pratique la contraction des variables coûteuse dans Charibde. Nous montrons toutefois dans la section 5.4 que la plupart des tests numériques de Charibde sont réalisés sans les $2n$ appels de X-Newton.

Une variante de MohcRevise dédiée à l'optimisation

Le contracteur principal de Charibde pour les problèmes sous contraintes (algorithme 15) invoque Mohc sur le système de contraintes $\mathcal{C}$. Un appel supplémentaire à MohcRevise permet de traiter la contrainte $f \leq \tilde{f}$ lorsque l'expression de f contient plusieurs occurrences des variables. Précisons que dans le cas d'une contrainte d'inégalité, seuls un appel à MinRevise et un appel à Left/RightNarrowFmin sont requis.

L'appel à MohcRevise est suivi d'une procédure spécifique à Charibde, visant à extraire un majorant du minimum global, ou de contracter la boîte courante $\boldsymbol{X}$ en une sous-boîte réalisable sans éliminer le minimum sur $\boldsymbol{X}$. Rappelons dans un premier temps que $\boldsymbol{X}^-$ (voir définition 17, page 36) est une sous-boîte de $\boldsymbol{X}$ dans laquelle les variables multi-occurrentes et détectées monotones sont remplacées par une de leurs bornes (la borne gauche pour les variables croissantes, la borne droite pour les variables décroissantes). Par construction,

Algorithme 17 Calcul d'un minorant par monotonie et contraction

fonction ContractionParMonotonie(**in-out** $\boldsymbol{X}$: boîte, F : fonction objectif, lb : minorant, **in-out** $\tilde{f}$: meilleur majorant, **in-out** $\mathcal{C}$: système de contraintes)
  $(lb_M, \boldsymbol{X}^-) \leftarrow$ MohcRevise$(F(\boldsymbol{X}) \leq \tilde{f})$ ▷ évaluation par monotonie/contraction
  si $\tilde{f} < lb_M$ **alors**
    éliminer $\boldsymbol{X}$
  sinon
    $lb \leftarrow \max(lb, lb_M)$ ▷ mise à jour du minorant
    si $\boldsymbol{X}^-$ est réalisable **alors** ▷ évaluation des contraintes par AI
      $\boldsymbol{X} \leftarrow \boldsymbol{X}^-$
      $\mathcal{C} \leftarrow \varnothing$
    sinon si $m(\boldsymbol{X}^-)$ est réalisable **alors**
      $\tilde{f} \leftarrow \min(\tilde{f}, \overline{F(m(\boldsymbol{X}^-))})$ ▷ mise à jour du majorant
    fin si
  fin si
  renvoyer lb
fin fonction

$\boldsymbol{X}^-$ est une sous-boîte de $\boldsymbol{X}$ contenant le minimum non contraint de f sur $\boldsymbol{X}$, c'est-à-dire la solution du problème :

$$\min_{\boldsymbol{x} \in \boldsymbol{X}} f(\boldsymbol{x}) \tag{5.5}$$

Par conséquent, si $\boldsymbol{X}^-$ est une boîte réalisable, elle contient également le minimum contraint de f sur $\boldsymbol{X}$. Dans le cas contraire, les contraintes sont évaluées sur un point de $\boldsymbol{X}^-$ (par exemple le centre de la boîte) afin de mettre à jour le meilleur majorant connu $\tilde{f}$. La procédure complète est donnée dans l'algorithme 17.

5.2.3 MaxDist : une heuristique de recherche géométrique

L'ordre d'extraction des boîtes restantes dans la file de priorité $\mathcal{L}$ détermine la stratégie d'exploration de l'espace de recherche (voir sous-section 3.3.3). Des tests numériques montrent que la stratégie « best-first search » est peu cohérente lorsque la fonction objectif f contient de nombreuses occurrences des variables : la borne gauche de l'évaluation de f sur une boîte de largeur importante donne en général peu d'indications sur la véritable image de f. La stratégie « largest first » consiste à explorer l'espace de recherche en largeur, sans privilégier les sous-espaces prometteurs. La stratégie « depth-first search » tend à explorer rapidement le voisinage des minima locaux, sans toutefois s'en échapper de manière efficace.

Nous proposons une nouvelle heuristique de recherche appelée MaxDist. L'idée sous-jacente est de parvenir à élaguer le voisinage du minimum global $\boldsymbol{x}^*$, non connu a priori, en disposant du meilleur majorant $\tilde{f}$ possible. Il s'agit généralement d'une tâche difficile, en raison de la similarité des valeurs de f dans un voisinage de $\boldsymbol{x}^*$. Afin de repousser au plus

tard cette exploration coûteuse, nous proposons d'extraire systématiquement de $\mathcal{L}$ la boîte la plus éloignée (pour une certaine métrique) de la solution courante $\tilde{\boldsymbol{x}}$ dans l'espace de recherche. Nous avons constaté que :

1. la convergence locale de l'ED dans un voisinage de $\tilde{\boldsymbol{x}}$ est performante ;
2. lorsque la population de l'ED s'agglutine autour d'une solution, il est difficile de s'en extraire et d'explorer d'autres zones de l'espace de recherche.

Deux possibilités (non connues a priori) s'offrent alors :

- soit $\boldsymbol{x}^*$ se trouve dans un voisinage de $\tilde{\boldsymbol{x}}$: on peut alors espérer que $\boldsymbol{x}^*$ soit rapidement atteint par l'ED ;
- soit $\boldsymbol{x}^*$ est éloigné de $\tilde{\boldsymbol{x}}$ dans l'espace de recherche : il n'est alors pas nécessaire d'explorer davantage le voisinage de $\tilde{\boldsymbol{x}}$. Le BCI doit alors se concentrer sur les zones de l'espace de recherche qui sont (éventuellement) difficilement accessibles à la population de l'ED, c'est-à-dire éloignées de la solution courante dans l'espace de recherche.

Notre heuristique MaxDist a donc la particularité d'explorer des zones de l'espace de recherche inacessibles pour l'ED, éventuellement plus prometteuses.

La fonction de distance entre un point $\boldsymbol{x}$ et une boîte $\boldsymbol{X}$ est détaillée dans l'algorithme 18 : elle consiste simplement à sommer l'écart entre $\boldsymbol{x}$ et la plus proche borne de $\boldsymbol{X}$ sur chaque composante. MaxDist est une heuristique adaptative : lorsque la meilleure solution courante $\tilde{\boldsymbol{x}}$ est mise à jour, $\mathcal{L}$ est réordonnée en recalculant la priorité de chacune des boîtes.

Algorithme 18 Distance entre un point et une boîte

fonction DISTANCE($\boldsymbol{x} = (x_1, \ldots, x_n)$: individu, $\boldsymbol{X} = (X_1, \ldots, X_n)$: boîte)
  $d \leftarrow 0$
  pour $j = 1$ à n **faire**
    si $\overline{X_j} < x_j$ **alors**
      $d \leftarrow d + (x_j - \overline{X_j})^2$
    sinon si $x_j < \underline{X_j}$ **alors**
      $d \leftarrow d + (\underline{X_j} - x_j)^2$
    fin si
  fin pour
  renvoyer d
fin fonction

Dans le tableau 5.1, nous comparons notre heuristique MaxDist et les heuristiques « best-first search » (notée MinLb) et « largest first » (notée MaxSize) sur huit problèmes d'optimisation tirés de la base COCONUT [1]. Le temps de calcul de Charibde (en secondes) et la taille maximale de la file de priorité $\mathcal{L}$, notée $|\mathcal{L}|_{max}$, sont donnés pour chaque problème. Ces résultats préliminaires montrent que MaxDist est compétitive par rapport aux heuristiques standards : les huit problèmes de test sont résolus avec MaxDist en un

1. La base de 993 problèmes COCONUT est disponible à l'adresse `http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`

temps cumulé de 88.36s, soit un gain relatif de 10.1% par rapport à MaxSize (98.27s) et un gain relatif de 27.4% par rapport à MinLb (121.66s).

TABLE 5.1 – Comparaison des heuristiques de recherche de Charibde

	MinLb		MaxSize		MaxDist	
Problème	Temps (s)	$\|\mathcal{L}\|_{max}$	Temps (s)	$\|\mathcal{L}\|_{max}$	Temps (s)	$\|\mathcal{L}\|_{max}$
avgasa	6.26	4301	6.12	39864	**5.7**	23
ex2_1_7	47.9	29530	28.3	78784	**26**	19
ex2_1_9	44	111158	43.6	54587	**37.5**	134
ex7_3_5	12	34430	8.9	14064	**8.42**	87
ex6_2_6	2.13	3800	2.15	2236	**1.96**	45
ex6_2_8	3.2	6377	3.13	4316	**3.03**	20
ex6_2_9	3.7	5428	3.66	2924	**3.47**	28
ex6_2_11	2.47	4928	2.41	2556	**2.28**	38
Somme	121.66		98.27		**88.36**	

Outre les performances en temps de calcul, la stratégie MaxDist affiche une taille maximale de $\mathcal{L}$ remarquablement basse (entre 19 et 134 boîtes) par rapport aux autres stratégies (entre 3800 et 111158 pour MinLb, entre 2236 et 78784 pour MaxSize). Ceci n'a qu'une influence limitée sur le temps d'insertion des boîtes dans la file $\mathcal{L}$ (en temps logarithmique dans un maximier), mais offre des perspectives prometteuses pour la collaboration entre BCI et ED : la file de priorité $\mathcal{L}$ de taille modérée peut être transmise à faible coût à l'ED afin de mettre à jour sa population (restart de la population sur un domaine contracté, évaluations paresseuses des individus). Ces techniques sont introduites dans la sous-section 5.3.4.

5.3 Evolution différentielle

L'ED apparaît comme une méthode de choix pour améliorer le meilleur majorant connu $\tilde{f}$ du minimum global f^* et intensifier les coupes de l'espace de recherche : les AE sont dotés de mécanismes permettant de s'échapper des minima locaux et sont capables de générer des solutions réalisables sans connaissance a priori de la topologie du problème. L'ED s'est montrée compétitive sur les problèmes continus et possède peu d'hyperparamètres, rendant le réglage de Charibde moins fastidieux.

Lorsque l'ED améliore la meilleure évaluation connue, l'individu et son évaluation sont transmises au BCI : le premier contribue au calcul de la priorité MaxDist (décrit dans la sous-section 5.2.3), et la seconde à la mise à jour du meilleur majorant connu $\tilde{f}$. En retour, l'algorithme reçoit une solution ponctuelle de la part du BCI lorsque celui-ci parvient à améliorer $\tilde{f}$ (algorithme 19). Afin de ne pas remplacer entièrement la population lorsque de nombreuses solutions y sont succesivement injectées, nous choisissons de remplacer systématiquement le même individu de la population.

Algorithme 19 Charibde : algorithme à évolution différentielle

fonction ED(NP : taille de la population, W : facteur d'amplitude, CR : taux de croisement, $\boldsymbol{D}$: espace de recherche)
  $P \leftarrow$ population initiale, générée aléatoirement dans $\boldsymbol{D}$
  $\tilde{f} \leftarrow +\infty$
  répéter
    $(\boldsymbol{x}, f_{\boldsymbol{x}}) \leftarrow$ MPI_RecevoirBCI()
    ajout de $\boldsymbol{x}$ à P
    $\tilde{f} \leftarrow f_{\boldsymbol{x}}$
    Génération de la population temporaire P' à partir de P
    $P \leftarrow P'$
    $(\boldsymbol{x}_{best}, f_{best}) \leftarrow$ MeilleureEvaluation(P)
    si $f_{best} < \tilde{f}$ **alors**
      $\tilde{f} \leftarrow f_{best}$
      MPI_EnvoyerBCI($\boldsymbol{x}_{best}, f_{best}$)
    fin si
    jusqu'à critère d'arrêt vérifié
    renvoyer meilleur individu de P
  fin fonction

Il convient d'être particulièrement attentif à la manière dont coopèrent les processus ED et BCI : l'algorithme de BCI est intrinsèquement fiable et garantit l'optimalité de la solution, même en présence d'arrondis. Il est donc nécessaire de fiabiliser les échanges MPI de l'ED vers le BCI. Le traitement de la fonction objectif est détaillé dans la sous-section 5.3.1 et le traitement des contraintes est expliqué dans la sous-section 5.3.2. La sous-section 5.3.3 introduit une fonction d'évaluation d'un nouvel individu basée sur la comparaison avec son parent. Les techniques permettant à l'ED d'exploiter le domaine restant maintenu par le BCI sont présentées dans la sous-section 5.3.4.

5.3.1 Gestion rigoureuse de la fonction objectif

Notons f_{near} l'implémentation de f sur une machine en mode d'arrondi « au plus près ». Pour tout $\boldsymbol{x}$ du domaine, $f_{near}(\boldsymbol{x}) \in F(\boldsymbol{x})$. Lorsque $\boldsymbol{x}$ est réalisable, seule la borne $\overline{F(\boldsymbol{x})}$ est un majorant fiable du minimum global f^*. Deux approches sont envisageables :

- la fonction objectif est systématiquement évaluée par AI. Cette approche revient à minimiser la fonction $\boldsymbol{x} \mapsto \overline{F(\boldsymbol{x})}$;
- la fonction objectif n'est évaluée en AI que lorsque la meilleure évaluation est améliorée.

La première stratégie est la plus rigoureuse, car elle manipule la même fonction objectif que le BCI, à savoir la borne droite de l'évaluation par AI. Elle implique néanmoins un surcoût par rapport à un algorithme standard basé sur l'AF. La deuxième stratégie fait l'hypothèse que f_{near} et $\overline{F}$ ont la même monotonie, et donc que $\min_{\boldsymbol{x}} f_{near}(\boldsymbol{x}) = \min_{\boldsymbol{x}} \overline{F(\boldsymbol{x})}$. Nous avons réalisé des tests numériques montrant que pour certains problèmes de test, à $\boldsymbol{x}_1$

et $\boldsymbol{x}_2$ donnés, on a $f_{near}(\boldsymbol{x}_1) > f_{near}(\boldsymbol{x}_2)$ et $\overline{F(\boldsymbol{x}_1)} < \overline{F(\boldsymbol{x}_2)}$ (figure 5.2). Néanmoins, l'écart entre les évaluations « au plus près » et par AI est en général très inférieur (de l'ordre de 10^{-15}) à la précision demandée sur l'optimum global.

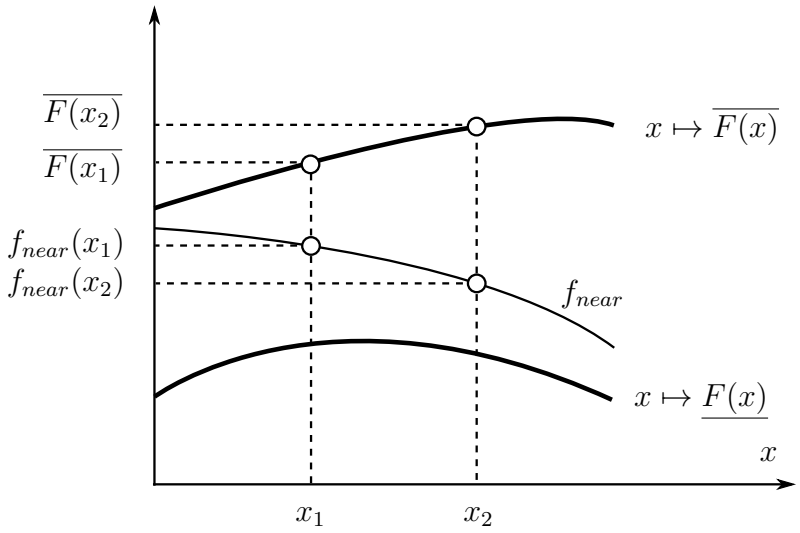


FIGURE 5.2 – Différence de monotonie entre f_{near} et $x \mapsto \overline{F(x)}$

Cette observation a motivé l'implémentation de la deuxième stratégie dans Charibde. Les individus de l'ED sont systématiquement évalués en utilisant l'AF. Lorsque la meilleure évaluation « au plus près » est améliorée, l'individu est évalué par AI. La borne droite est alors comparée au meilleur majorant *fiable* connu : si ce dernier est amélioré, le nouveau majorant rigoureux est transmis au BCI (algorithme 20). Ce choix a permis de réduire substantiellement le coût des évaluations et d'accélérer l'algorithme à ED, tout en garantissant que toutes les valeurs envoyées au BCI sont fiables.

Algorithme 20 Mise à jour du meilleur individu à la fin de chaque génération

fonction FINGÉNÉRATION($\boldsymbol{x}$: meilleur individu de la génération, **in-out** $\tilde{f}_{near}$: meilleure évaluation en AF connue, **in-out** $\tilde{f}$: meilleure évaluation *rigoureuse* connue)
  si $\boldsymbol{x}$ est réalisable **alors**
    $f_{\boldsymbol{x}} \leftarrow f_{near}(\boldsymbol{x})$ ▷ évaluation « au plus près »
    si $f_{\boldsymbol{x}} < \tilde{f}_{near}$ **alors**
      $\tilde{f}_{near} \leftarrow f_{\boldsymbol{x}}$ ▷ mise à jour de la meilleure évaluation connue
      si $\overline{F(\boldsymbol{x})} < \tilde{f}$ **alors**
        $\tilde{f} \leftarrow \overline{F(\boldsymbol{x})}$ ▷ majorant rigoureux
        MPI_EnvoyerBorneBCI($\tilde{f}$)
      fin si
    fin si
  fin si
fin fonction

5.3.2 Gestion rigoureuse des contraintes

L'extension des AE à l'optimisation sous contraintes a fait l'objet de nombreux travaux. Les approches les plus largement considérées incluent les méthodes de pénalités et la gestion directe des contraintes (voir section 2.1.1). Nous avons adapté cette dernière approche faisant intervenir la violation des contraintes (leur nombre et leur magnitude) dans l'opérateur de comparaison entre deux individus $\boldsymbol{x}$ et $\boldsymbol{y}$. Nous notons pour un individu $\boldsymbol{x}$:

- $f_{\boldsymbol{x}}$ la valeur objectif de $\boldsymbol{x}$;
- $n_{\boldsymbol{x}}$ le nombre de contraintes violées ($\in \{1, \ldots, m\}$) par $\boldsymbol{x}$;
- $s_{\boldsymbol{x}} := \sum_{i=1}^{m} \max(0, g_i(\boldsymbol{x}))$ la somme des magnitudes des contraintes violées par $\boldsymbol{x}$.

L'évaluation d'un individu $\boldsymbol{x}$ s'écrit alors sous la forme :

- *Feasible*($f_{\boldsymbol{x}}$) lorsque $\boldsymbol{x}$ est un individu réalisable ;
- *Unfeasible*($n_{\boldsymbol{x}}, s_{\boldsymbol{x}}$) lorsque $\boldsymbol{x}$ n'est pas réalisable. Si au moins une des contraintes est violée, la fonction objectif n'est pas évaluée.

Les règles de comparaison suivantes définissent une relation d'ordre entre les individus $\boldsymbol{x}$ et $\boldsymbol{y}$; elles permettent de déterminer lequel de $\boldsymbol{x}$ ou de $\boldsymbol{y}$ est conservé dans la population :

1. un individu réalisable est meilleur qu'un individu non réalisable ;
2. le meilleur de deux individus réalisables est celui dont la valeur objectif est la plus faible ;
3. le meilleur de deux individus non réalisables est celui dont le nombre de contraintes violées est le plus faible, puis celui dont la magnitude des contraintes violées est la plus faible.

De nombreux solveurs d'optimisation non-linéaire tolèrent une légère violation des contraintes d'inégalité (par exemple $g \leq 10^{-6}$ au lieu de $g \leq 0$). L'évaluation d'une solution « pseudo-réalisable » $\boldsymbol{x}$ (satisfaisant ces contraintes relaxées) ne permet pas de déterminer un majorant fiable du minimum global, les erreurs numériques menant parfois à des résultats aberrants : l'évaluation de $\boldsymbol{x}$ est parfois inférieure au minimum global exact, et (ou) est très éloignée de solutions réalisables dans l'espace de recherche.

Nous avons pris le parti d'étiqueter un individu comme réalisable uniquement lorsqu'il est numériquement établi qu'il satisfait les contraintes du problème. Dans Charibde, un individu $\boldsymbol{x}$ soumis à des contraintes d'inégalité $g_i \leq 0$ ($i \in \{1, \ldots, m\}$) est considéré comme réalisable lorsque :

$$\forall i \in \{1, \ldots, m\}, \quad \overline{G_i(\boldsymbol{x})} \leq 0 \tag{5.6}$$

5.3.3 Evaluation par comparaison

Notre implémentation de l'ED invoque une fonction EvalParComparaison qui évalue chaque individu nouvellement généré $\boldsymbol{y}$ en le comparant à son individu parent $\boldsymbol{x}$ (algorithme 21). L'idée sous-jacente est d'exploiter l'évaluation connue de l'individu parent dans un test de réfutation pour limiter le nombre d'évaluations des contraintes.

Algorithme 21 Evaluation d'un individu relative à son parent

fonction EvalParComparaison($\boldsymbol{x}$: individu parent, $\boldsymbol{y}$: individu, $\mathcal{C}$: système de contraintes)
  si $\boldsymbol{x}$ n'est pas réalisable **alors**
    $(realisable_{\boldsymbol{y}}, n_{\boldsymbol{y}}, s_{\boldsymbol{y}}) \leftarrow$ EvalContraintes($\boldsymbol{y}$, $\mathcal{C}$) ▷ algorithme 22
    si $realisable_{\boldsymbol{y}}$ **alors**
      renvoyer $(\boldsymbol{y}, Feasible(f_{\boldsymbol{y}}))$
    sinon si $(n_{\boldsymbol{y}}, s_{\boldsymbol{y}})$ meilleur que $(n_{\boldsymbol{x}}, s_{\boldsymbol{x}})$ **alors**
      renvoyer $(\boldsymbol{y}, Unfeasible(n_{\boldsymbol{y}}, s_{\boldsymbol{y}}))$
    sinon
      renvoyer $(\boldsymbol{x}, Unfeasible(n_{\boldsymbol{x}}, s_{\boldsymbol{x}}))$
    fin si
  sinon
    si $f_{\boldsymbol{y}} < f_{\boldsymbol{x}}$ **alors**
      $realisable_{\boldsymbol{y}} \leftarrow$ EstRealisable($\boldsymbol{y}$, $\mathcal{C}$) ▷ algorithme 23
      si $realisable_{\boldsymbol{y}}$ **alors**
        renvoyer $(\boldsymbol{y}, Feasible(f_{\boldsymbol{y}}))$
      sinon
        renvoyer $(\boldsymbol{x}, Feasible(f_{\boldsymbol{x}}))$
      fin si
    sinon ▷ $\boldsymbol{y}$ ne peut pas améliorer la valeur objectif
      renvoyer $(\boldsymbol{x}, Feasible(f_{\boldsymbol{x}}))$
    fin si
  fin si
fin fonction

Lorsque $\boldsymbol{x}$ n'est pas réalisable, la fonction EvalContraintes (algorithme 22) évalue séquentiellement les contraintes du système $\mathcal{C}$ par AI et comptabilise les contraintes violées par $\boldsymbol{y}$ et leurs magnitudes. L'évaluation est interrompue dès que $\boldsymbol{y}$ viole davantage de contraintes (nombre ou magnitude) que son parent $\boldsymbol{x}$ (ce raffinement n'est pas détaillé dans le pseudo-code).

Algorithme 22 Evaluation rigoureuse des contraintes

fonction EvalContraintes($\boldsymbol{y}$: individu, $\mathcal{C}$: système de contraintes)
  $(realisable, n, s) \leftarrow (vrai, 0, 0.)$
  pour $c \in \mathcal{C}$ **faire**
    si $0 < \overline{C(\boldsymbol{y})}$ **alors** ▷ contrainte sur intervalles non satisfaite
      $(realisable, n, s) \leftarrow (faux, n+1, s + \overline{C(\boldsymbol{y})})$
    fin si
  fin pour
  renvoyer $(realisable, n, s)$
fin fonction

Si en revanche $\boldsymbol{x}$ est réalisable, un premier filtrage consiste à vérifier si la valeur objectif de $\boldsymbol{y}$ est inférieure à celle de $\boldsymbol{x}$. Ce test permet d'éviter l'évaluation des contraintes si la valeur objectif ne peut être améliorée. S'il est effectivement possible d'améliorer la valeur objectif de $\boldsymbol{x}$, un appel à la fonction EstRealisable (algorithme 23) détermine si l'individu courant est réalisable ; l'évaluation séquentielle des contraintes est interrompue à la première contrainte violée.

L'évaluation d'une contrainte g par AI étant estimée à 2 à 4 fois plus coûteuse qu'en AF [Neumaier 90], il est nécessaire de privilégier les évaluations flottantes lorsque cela est possible. Bien que non rigoureuse, nous exploitons dans Charibde le potentiel de réfutation de l'AF. Puisque l'évaluation de g_{near} (en arrondi au plus près) est contenue dans l'évaluation par AI $G(\boldsymbol{y})$, on a la relation :

$$g_{near}(\boldsymbol{y}) \leq \overline{G(\boldsymbol{y})} \tag{5.7}$$

Le test de réfutation suivant peut être implémenté dans l'ED pour substituer l'AF à l'AI dans la vérification de réalisabilité :

$$0 < g_{near}(\boldsymbol{y}) \Rightarrow 0 < \overline{G(\boldsymbol{y})} \tag{5.8}$$

Si le test de réfutation est satisfait ($0 < g_{near}(\boldsymbol{y})$), $\boldsymbol{y}$ n'est pas une solution réalisable pour le critère numérique de satisfaction retenu (équation 5.6). La réalisabilité exacte de $\boldsymbol{y}$ n'est en revanche pas connue. Si le test de réfutation échoue ($g_{near}(\boldsymbol{y}) \leq 0$), une évaluation de g par AI est nécessaire car aucune information sur le signe de $\overline{G(\boldsymbol{y})}$ n'est disponible : la figure 5.3 illustre différentes possibilités d'évaluations suivant le signe de $g_{near}(\boldsymbol{y})$ et le signe de $\overline{G(\boldsymbol{y})}$ (l'évaluation par AI est représentée en mauve).

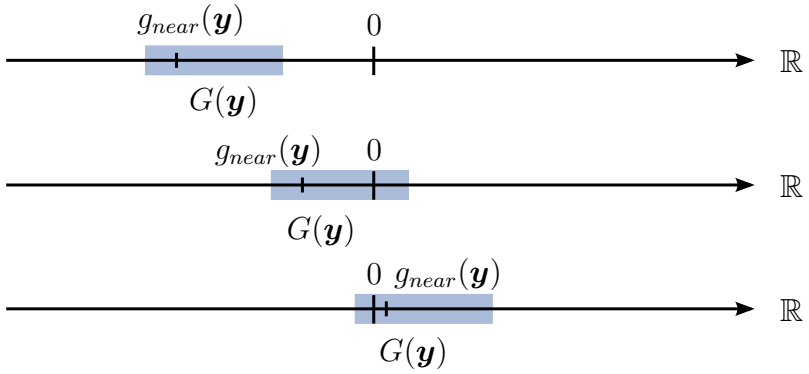


FIGURE 5.3 – Signe des contraintes

La fonction EstRealisable (algorithme 23) vérifie si $\boldsymbol{y}$ est réalisable en utilisant le test de réfutation 5.8 : les contraintes sont dans un premier temps évaluées en mode d'arrondi au plus près, dans l'espoir de réfuter $\boldsymbol{y}$ à faible coût. Si toutes les contraintes sont satisfaites en AF, la fonction EstRealisableAI (algorithme 24) – une version rigoureuse de EstRealisable – est invoquée.

Les figures 5.4 et 5.5 illustrent, pour les deux versions de l'ED, l'évolution de la meilleure valeur objectif au cours du temps sur 15 problèmes de test de la base COCONUT. La

Algorithme 23 Test de satisfaction des contraintes en arithmétique flottante

fonction EstRealisable($\boldsymbol{x}$: individu, $\mathcal{C}$: système de contraintes)
  pour $c \in \mathcal{C}$ **faire**
    si $0 < c(\boldsymbol{x})$ **alors** ▷ contrainte non satisfaite
      renvoyer *faux*
    fin si
  fin pour
  renvoyer EstRealisableAI($\boldsymbol{x}$) ▷ algorithme 24
fin fonction

Algorithme 24 Test de satisfaction des contraintes en arithmétique d'intervalles

fonction EstRealisableAI($\boldsymbol{x}$: individu, $\mathcal{C}$: système de contraintes)
  pour $c \in \mathcal{C}$ **faire**
    si $0 < \overline{C(\boldsymbol{x})}$ **alors** ▷ contrainte sur intervalles non satisfaite
      renvoyer *faux*
    fin si
  fin pour
  renvoyer *vrai*
fin fonction

première version implémente l'évaluation des contraintes uniquement par AI, tandis que la seconde version utilise le test de réfutation 5.8 et l'évaluation combinée par AF et AI. Les hyperparamètres de l'ED sont fixés expérimentalement à $(NP, W, CR) = (40, 0.7, 0.9)$, excepté pour ex2_1_7 ($NP = 20$).

Les résultats sur les différents problèmes suggèrent un début de convergence plus rapide tantôt de la première version (ex2_1_10, ex7_2_1, expfita), tantôt de la seconde (ex2_1_7, ex7_2_9, keane, s365mod). Ceci peut s'expliquer par le fait que lorsqu'un individu améliore l'évaluation de son parent, la deuxième version requiert l'évaluation de $2m$ contraintes (m en AF, m en AI) : le nombre de contraintes m et le nombre d'améliorations de la meilleure solution dans l'ED affectent donc le temps de convergence. Le tableau 5.2 montre des gains relatifs en temps CPU compris entre 5% et 60% en faveur de la deuxième version. La somme cumulée des temps de convergence indique un gain relatif total de 21%.

5.3.4 Exploitation du domaine sur intervalles

La file de priorité $\mathcal{L}$ maintenue par l'algorithme de BCI contient les boîtes réalisables et indécidables partitionnant l'espace de recherche, c'est-à-dire les boîtes qui n'ont pas été détectées sous-optimales ou non réalisables. Cette information peut être exploitée par l'algorithme à ED afin de réduire a priori les évaluations des individus non réalisables ou sous-optimaux a posteriori.

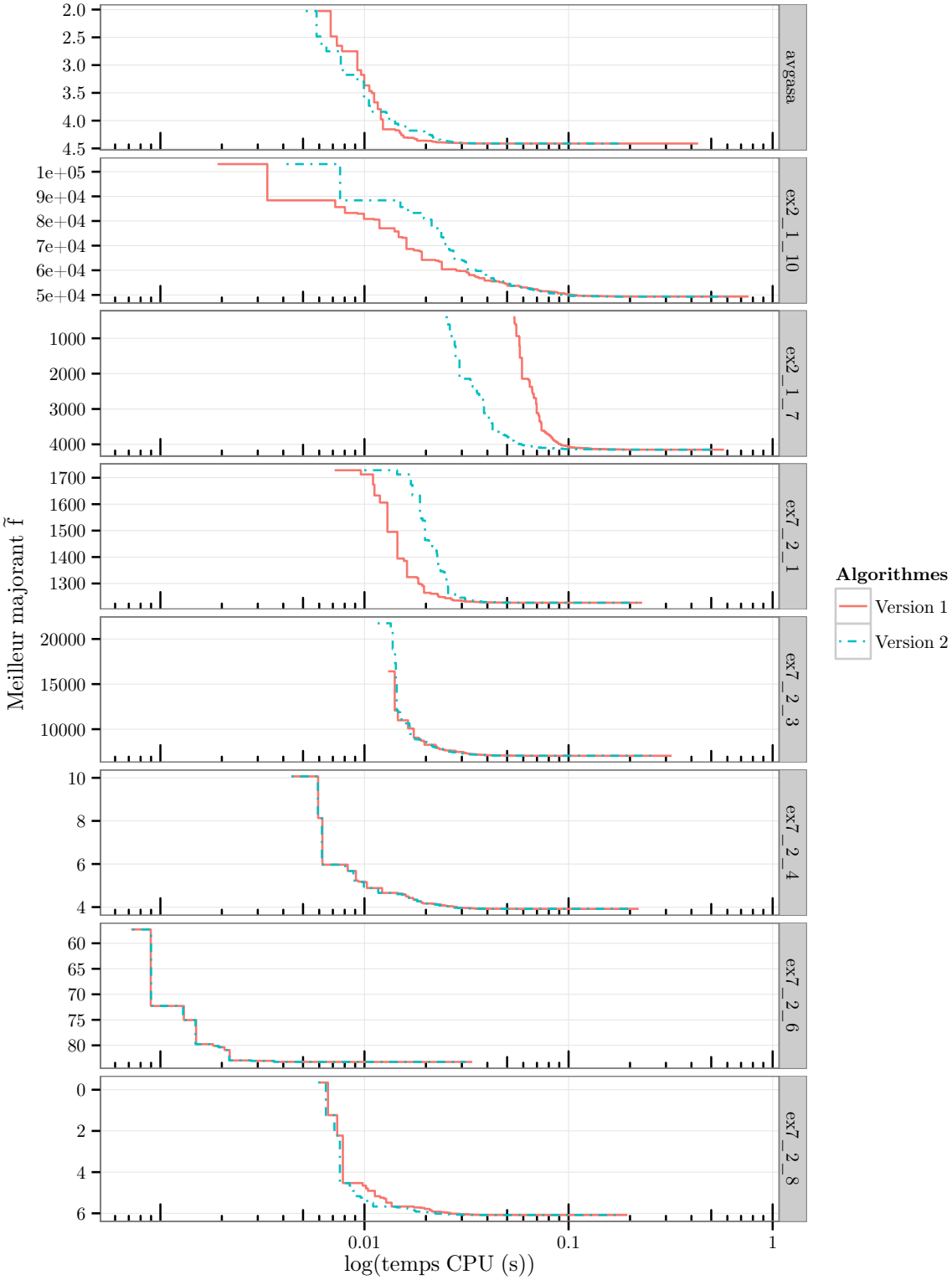


FIGURE 5.4 – Comparaison entre les deux versions de l'évolution différentielle

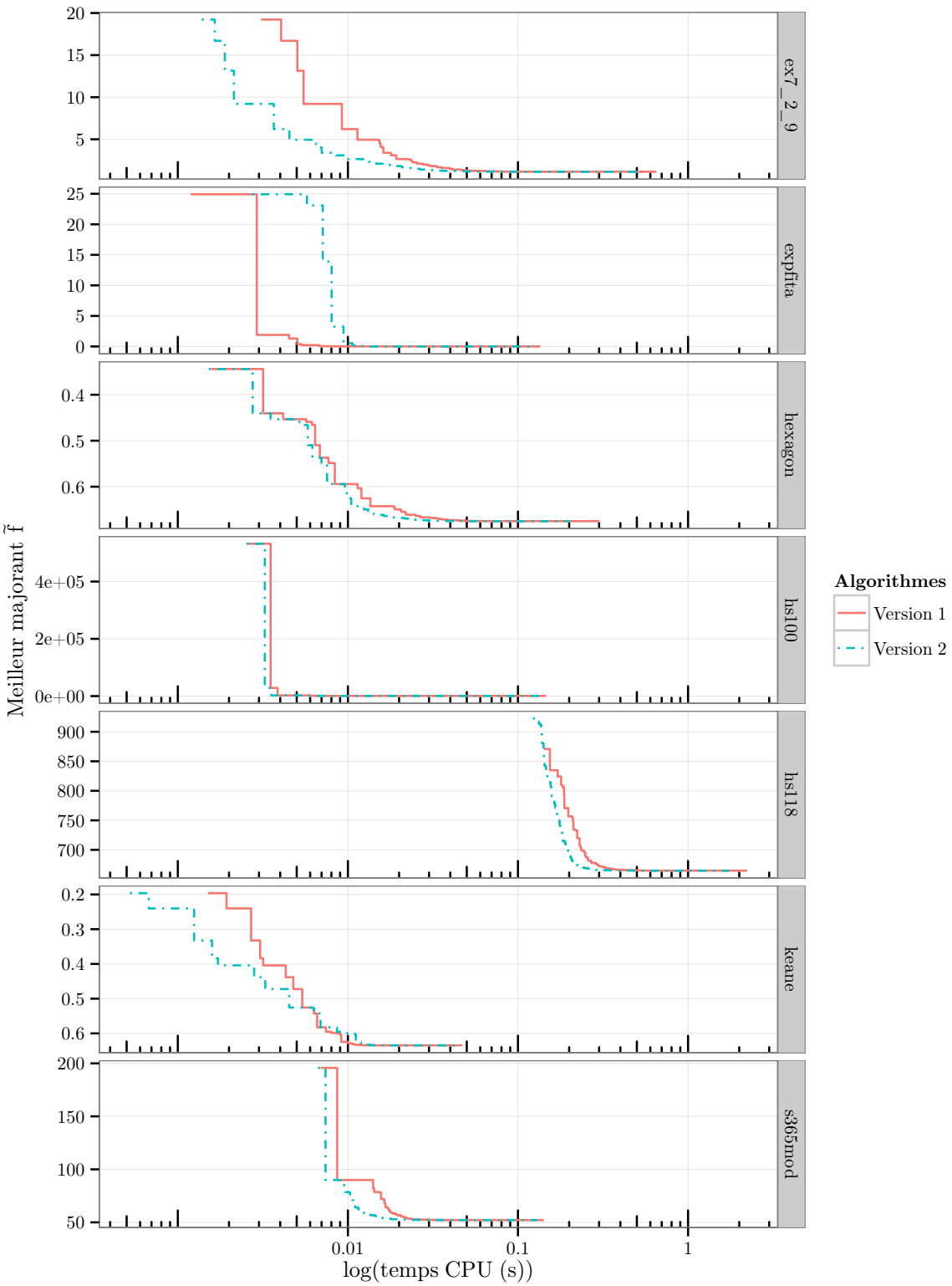


FIGURE 5.5 – Comparaison entre les deux versions de l'évolution différentielle (suite)

TABLE 5.2 – Temps de convergence des deux versions de l'évolution différentielle

Problème	Temps CPU (s) Version 1	Version 2	Gain (%)
avgasa	0.43	0.18	59
ex2_1_7	0.58	0.51	11.3
ex2_1_10	0.76	0.63	17.4
ex7_2_1	0.23	0.22	5.6
ex7_2_3	0.32	0.25	20.7
ex7_2_4	0.22	0.20	10.1
ex7_2_6	0.034	0.031	7.2
ex7_2_8	0.19	0.17	9.9
ex7_2_9	0.65	0.50	22.7
expfita	0.14	0.13	5.3
hexagon	0.30	0.20	33.2
hs100	0.15	0.14	5.3
hs118	2.23	1.74	21.7
keane	0.047	0.044	5.9
s365mod	0.14	0.13	8.8
Somme	6.421	5.075	21

Réduction du domaine initial

Il est possible d'éliminer des valeurs non réalisables du domaine initial de l'ED (correspondant à la boîte initiale du BCI) par des techniques de contraction. La population initiale de l'ED est alors initialisée dans le domaine contracté par le BCI.

L'équation 5.9 définit une mesure de volume d'une boîte $\boldsymbol{X} = (X_1, \ldots, X_n)$:

$$Vol(\boldsymbol{X}) := \prod_{i=1}^{n} w(X_i) \tag{5.9}$$

La figure 5.6 montre le gain relatif en volume obtenu après application séquentielle des contracteurs HC4, 3BCID(HC4) et X-Newton sur le domaine initial. Les problèmes de test considérés sont issus de la base COCONUT. Seul le domaine initial du problème ex_2_1_7 n'a pu être contracté. Les gains relatifs en volume varient de 1.7% (s365mod) à 57.8% (expfita).

Notre approche est similaire à l'approche de [Focacci 03], qui propose d'utiliser des techniques de programmation par contraintes comme préprocessing pour réduire l'espace de recherche initial d'une recherche locale.

Réduction périodique du domaine et restart

Une stratégie permettant d'exploiter le domaine restant maintenu par le BCI est de mettre à jour régulièrement le domaine de l'ED (sous la forme d'une boîte unique, convexe

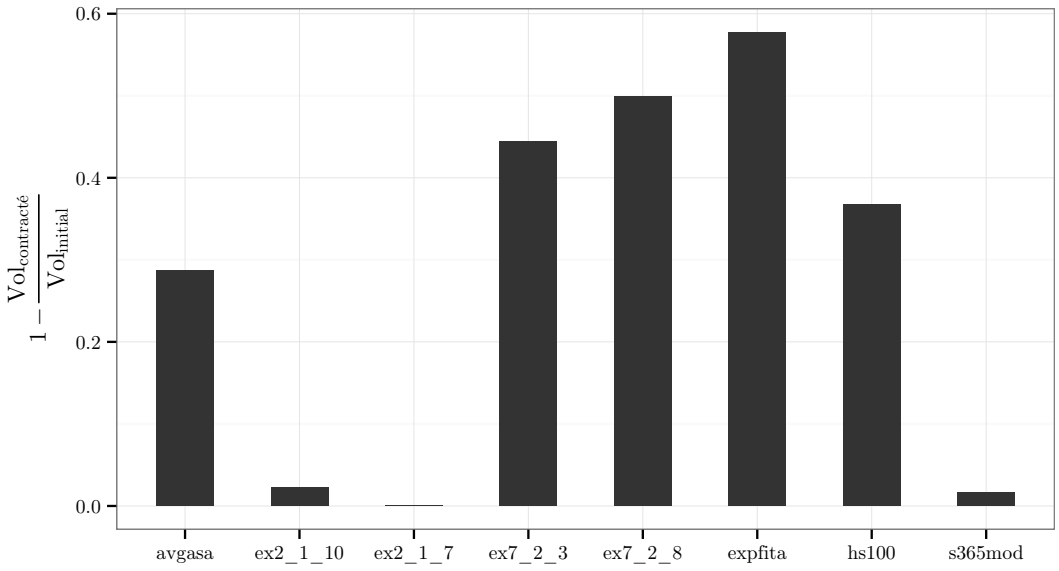


FIGURE 5.6 – Réduction relative du domaine initial sur les problèmes de test

et connexe) afin d'éliminer progressivement les minima locaux et les sous-espaces non réalisables. Notre approche consiste à transmettre périodiquement la file de priorité $\mathcal{L}$ du BCI à l'ED, puis à calculer l'enveloppe convexe $\square(\mathcal{L})$ des boîtes de $\mathcal{L}$ (c'est-à-dire la plus petite boîte les contenant). La population de l'ED est alors réinitialisée (regénérée aléatoirement) à l'intérieur du nouveau domaine $\square(\mathcal{L})$.

Notre stratégie de recherche MaxDist (sous-section 5.2.3) présente deux avantages majeurs :

1. le maintien d'une faible taille de la file de priorité $\mathcal{L}$ limite le coût de l'opération d'enveloppe convexe (de complexité linéaire) ;
2. par construction, l'heuristique traite les boîtes en périphérie du domaine restant, favorisant ainsi la réduction de l'enveloppe convexe.

La figure 5.7 présente l'évolution du volume du domaine (échelle logarithmique) en fonction du nombre d'évaluations de la fonction objectif sur 9 problèmes de test de la base COCONUT. Pour ces tests préliminaires, le processus de réduction du domaine est déclenché arbitrairement toutes les 20000 générations de l'ED. La figure souligne l'efficacité de l'approche, qui contracte rapidement le domaine de plusieurs ordres de grandeur et diminue les risques de rester piégé dans des minima locaux.

L'exemple 15 montre la réduction du domaine de l'ED au cours des générations, sans élimination du minimum global.

Exemple 15 *Soit un problème d'optimisation sous contraintes défini sur la boîte* $(X, Y) =$

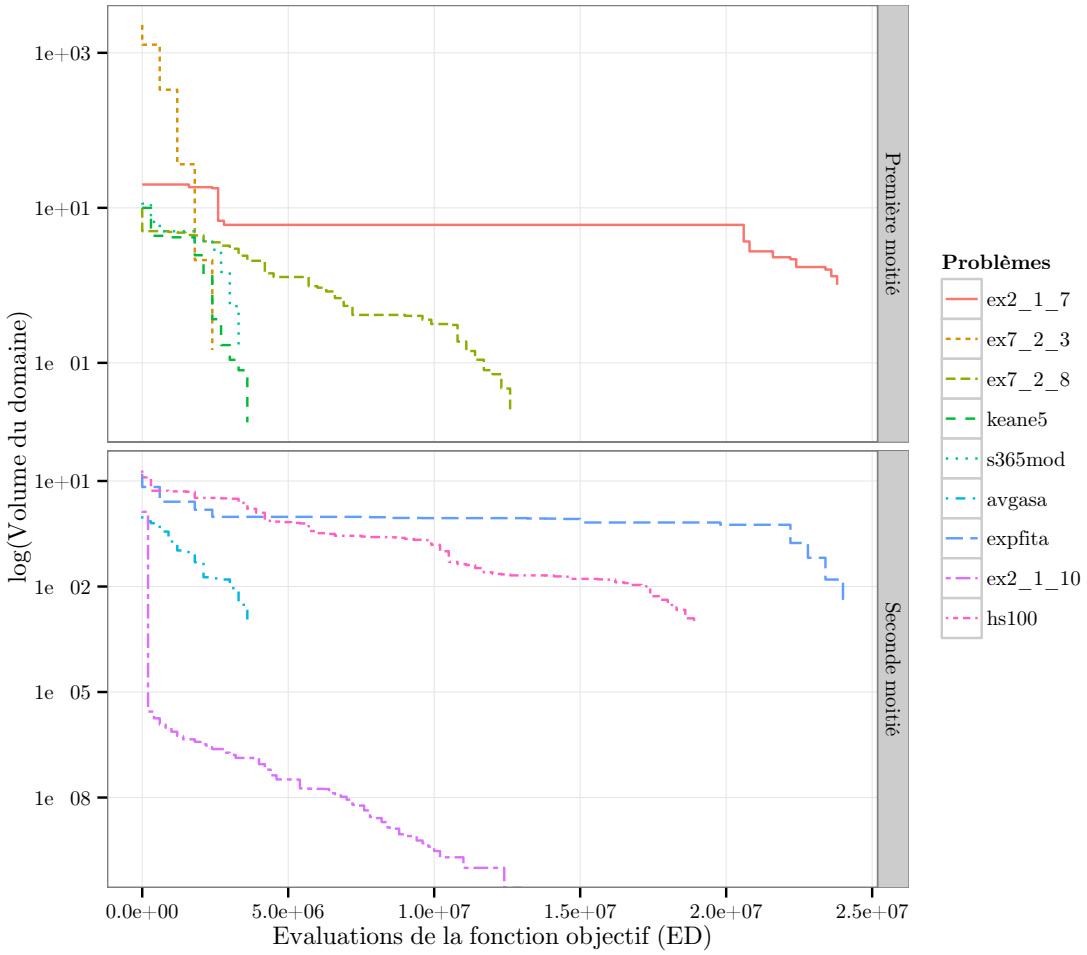


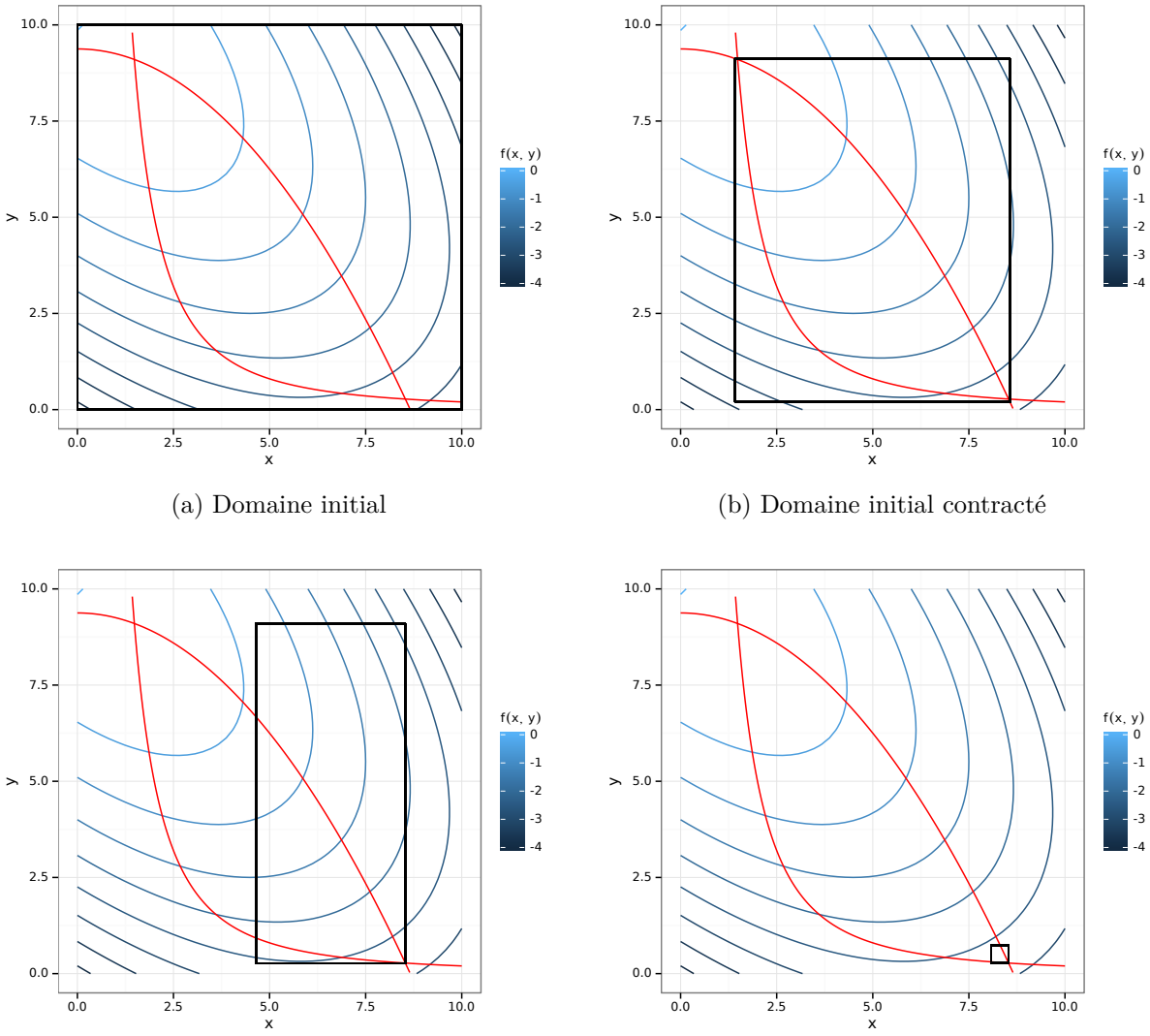
FIGURE 5.7 – Evolution du volume du domaine (échelle logarithmique) avec le nombre d'évaluations de la fonction objectif dans l'évolution différentielle

$([0, 10], [0, 10])$ *par :*

$$\begin{aligned} \min_{(x,y)\in(X,Y)} \quad & -\frac{(x+y-10)^2}{30} - \frac{(x-y+10)^2}{120} \\ s.c. \quad & \frac{20}{x^2} - y \leq 0 \\ & x^2 + 8y - 75 \leq 0 \end{aligned} \tag{5.10}$$

Le problème est représenté sur la figure 5.8a : les frontières des deux contraintes d'inégalité sont représentées en rouge et les courbes de niveau de la fonction objectif sont représentées en bleu. L'espace réalisable est l'ensemble bananesque délimité par les deux courbes rouges, et le minimum global se situe en son coin inférieur droit.

La figure 5.8b illustre le domaine initial contracté (représenté par le rectangle noir)



(a) Domaine initial

(b) Domaine initial contracté

(c) Génération 10

(d) Génération 20

FIGURE 5.8 – Evolution du domaine de l'évolution différentielle avec les générations

par rapport aux contraintes : $(X, Y) = ([1.4142, 8.5674], [0.2, 9.125])$. *Les contraintes sont simplement traitées l'une après l'autre par un opérateur 2B.*

Les figures 5.8c et 5.8d représentent l'enveloppe convexe des boîtes restantes maintenues par l'algorithme de BCI, respectivement après 10 et 20 générations de l'ED. Le minimum global trouvé par Charibde avec une précision $\varepsilon = 10^{-8}$ *est* $f(x^*, y^*) = f(8.532424, 0.274717) = -2.825296148$ *; les deux contraintes y sont actives. L'expression analytique de la solution correspondante est* $(x^*, y^*) = (\sqrt{\frac{\sqrt{4985}+75}{2}}, \frac{40}{\sqrt{4985}+75}) \simeq (8.532424404, 0.274716723)$.

Evaluations paresseuses

L'envoi de la file de priorité $\mathcal{L}$ à l'ED offre également la possibilité d'évaluer les contraintes sur un individu $\boldsymbol{x}$ de manière paresseuse : seules les contraintes indéterminées (dont l'évaluation en AI contient 0) sur la boîte contenant $\boldsymbol{x}$ sont évaluées. Si $\boldsymbol{x}$ n'appartient à aucune boîte, il se situe en dehors du domaine restant (il est donc soit sous-optimal, soit non réalisable) et n'est alors pas évalué. Le type des évaluations (composé de *Feasible* ou *Unfeasible*, voir sous-section 5.3.2) est enrichi d'une troisième option *Out*, caractérisant un individu hors domaine qui n'a pas été évalué.

L'algorithme 25 décrit la fonction d'évaluation paresseuse. S'il existe une boîte $\hat{\boldsymbol{X}}$ de $\mathcal{L}$ contenant l'individu $\boldsymbol{x}$, seules les contraintes encore indéterminées sur $\hat{\boldsymbol{X}}$ sont évaluées par une procédure EvalContraintesListe (non décrite ici) similaire à EvalContraintes (algorithme 22). Si $\hat{\boldsymbol{X}}$ est réalisable, $\boldsymbol{x}$ est marqué comme solution réalisable, sans aucune évaluation des contraintes. Si $\boldsymbol{x}$ appartient à l'enveloppe convexe $\square(\mathcal{L})$ mais pas à $\mathcal{L}$, l'individu n'est pas évalué et étiqueté *Out*.

Algorithme 25 Evaluation d'un individu sur la boîte le contenant

```
fonction EvalParesseuse(x : individu, L : file de priorité de boîtes)
    (X̂, d_min) ← PlusProcheBoite(x, L)                    ▷ algorithme 26
    si d_min = 0 alors
        si X̂ sujette à une liste de contraintes indéterminées alors
            (realisable, n, s) ← EvalContraintesListe(X̂, liste)
            si realisable alors
                renvoyer Feasible(f(x))
            sinon
                renvoyer Unfeasible(n, s)
            fin si
        sinon                                              ▷ solution réalisable
            renvoyer Feasible(f(x))
        fin si
    sinon
        renvoyer Out                                       ▷ aucune évaluation
    fin si
fin fonction
```

Algorithme 26 Calcul de la boîte la plus proche d'un individu

fonction PlusProcheBoite($\boldsymbol{x}$: individu, $\mathcal{L}$: file de priorité de boîtes)
  $\hat{\boldsymbol{X}} \leftarrow \varnothing$ ▷ plus proche boîte
  $d_{min} \leftarrow +\infty$ ▷ distance plus proche boîte/individu
  répéter
    $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\boldsymbol{X}\}$ ▷ extraction d'une boîte
    si $\boldsymbol{x} \in \boldsymbol{X}$ **alors**
      $d_{min} = 0$
      $\hat{\boldsymbol{X}} \leftarrow \boldsymbol{X}$
    sinon
      $d \leftarrow$ Distance$(\boldsymbol{x}, \boldsymbol{X})$ ▷ algorithme 18
      si $d < d_{min}$ **alors**
        $d_{min} \leftarrow d$
        $\hat{\boldsymbol{X}} \leftarrow \boldsymbol{X}$
      fin si
    fin si
  jusqu'à $\mathcal{L} = \varnothing$ ou $d_{min} = 0$
  renvoyer $(\hat{\boldsymbol{X}}, d_{min})$
fin fonction

5.4 Comparaison de solveurs sur la base COCONUT

Actuellement, les solveurs fiables GlobSol, IBBA et Ibex sont parmi les plus performants en optimisation sous contraintes. Partageant un squelette commun de BBI, ils diffèrent toutefois par la nature des techniques d'accélération utilisées. GlobSol [Kearfott 96b, Kearfott 05] calcule des relaxations linéaires basées sur une technique de reformulation/linéarisation. IBBA [Ninin 10b] invoque un algorithme d'évaluation-propagation et calcule une relaxation du système de contraintes en utilisant la variante AF2 de l'AA. Ibex [Chabert 09a] est un solveur dédié à la satisfaction de contraintes et à l'optimisation ; il intègre la plupart des contracteurs les plus récents et les plus performants (HC4, 3B, Mohc, CID, X-Newton).

5.4.1 Base de comparaison

Ces solveurs sont comparés à Charibde sur une base de problèmes sous contraintes issus de la base COCONUT, sélectionnés par [Araya 12] pour leur difficulté : ex2_1_7, ex2_1_9, ex6_2_6, ex6_2_8, ex6_2_9, ex6_2_11, ex6_2_12, ex7_2_3, ex7_3_5, ex14_1_7 et ex14_2_7. L'instance la plus petite compte trois variables et une contrainte, la plus grande compte 20 variables et 10 contraintes. En raison d'instabilités numériques (« assert failure »), la bibliothèque de programmation linéaire ocaml-glpk interrompt l'exécution de Charibde sur les problèmes ex6_1_1, ex6_1_3 et ex_6_2_10, pour lesquels aucun résultat n'est présenté.

5.4.2 Comparaison des résultats

La comparaison des quatre solveurs GlobSol, IBBA (résultats donnés dans [Ninin 10b]), Ibex (résultats donnés dans [Araya 12]) et Charibde sur le benchmark de 11 problèmes d'optimisation est détaillée dans le tableau 5.3. Pour chaque problème, la première ligne indique le temps de calcul (en secondes), et la deuxième ligne donne le nombre de bissections. La précision sur la fonction objectif est identique pour tous les solveurs ($\varepsilon = 10^{-8}$), de même que la précision de la relaxation des contraintes d'égalité ($\varepsilon_{=} = 10^{-8}$). TO (timeout) indique que la convergence n'a pas été atteinte au bout d'une heure.

TABLE 5.3 – Comparaison du temps de convergence (en s) et du nombre de bissections des solveurs GlobSol, IBBA, Ibex et Charibde sur des problèmes difficiles d'optimisation sous contraintes

Problème	n	m	GlobSol	IBBA	Ibex	Charibde
ex2_1_7	20	10		16.7	**7.74**	26
				1574	1344	67249
ex2_1_9	10	1		154	**9.07**	36
				60007	5760	328056
ex6_2_6	3	1	306	1575	136	**1.96**/6.92
				922664	61969	15787
ex6_2_8	3	1	204	458	59.3	**2.77**/10.66
				265276	25168	23047
ex6_2_9	4	2	463	523	25.2	**2.76**/4.54
				203775	27892	34591
ex6_2_11	3	1	273	140	7.51	**1.97**/2.76
				83487	8498	26952
ex6_2_12	4	2	196	112	22.2	**8.7**/10.75
				58231	7954	127198
ex7_2_3	8	6		TO	544	**1.59**
					611438	743
ex7_3_5	13	15		TO	28.91	**8.8**
					5519	36072
ex14_1_7	10	17		TO	406	**4**
					156834	8065
ex14_2_7	6	9		TO	66.39	**0.3**
					12555	587
Somme			> 1442	TO	1312.32	**94.85**/112.32

Deux temps de convergence de Charibde sont donnés pour les problèmes ex6_2_6, ex6_2_8, ex6_2_9, ex6_2_11 et ex6_2_12 : la deuxième valeur est le temps de convergence sur le problème initial, la première valeur est le temps de convergence sur le problème reformulé (voir paragraphes suivants).

Les résultats de GlobSol (programme non libre) ne sont pas disponibles pour tous les problèmes de test ; ne sont mentionnés que les résultats donnés dans [Ninin 10b]. Les résultats d'Ibex sont issus de [Araya 12] : seul le temps de résolution de la meilleure stratégie (appel au simplexe, à X-NewIter ou à X-Newton) pour le problème considéré apparaît dans le tableau. Charibde est exécuté sur un processeur Intel Xeon(R) CPU E31270 @ 3.40GHz x 8 avec 7.8 Go de mémoire. Les hyperparamètres de Charibde pour chacun des problèmes de test sont détaillés dans le tableau 5.4 ; NP est la taille de la population et η le ratio de point fixe. Trois heuristiques de partitionnement sont implémentées : RR (Round Robin) découpe les variables l'une après l'autre, Largest bissecte la dimension la plus large et Smear partitionne la dimension sur laquelle la fonction objectif varie le plus pour la forme de Taylor. Le facteur d'amplitude $W = 0.7$, le taux de croisement $CR = 0.9$ et l'heuristique MaxDist sont communs à tous les problèmes.

TABLE 5.4 – Hyperparamètres de Charibde sur les problèmes de test contraints

Problème	NP	Bissection	η	Simplexe	X-Newton
ex2_1_7	20	RR	0.9	✓	✓
ex2_1_9	100	RR	0.8	✓	
ex6_2_6	30	Smear	0	✓	
ex6_2_8	30	Smear	0	✓	
ex6_2_9	70	Smear	0		
ex6_2_11	35	Smear	0		
ex6_2_12	35	RR	0	✓	
ex7_2_3	40	Largest	0	✓	✓
ex7_3_5	30	RR	0	✓	
ex14_1_7	40	RR	0	✓	
ex14_2_7	40	RR	0	✓	

Charibde surpasse Ibex sur 9 des 11 problèmes considérés, et IBBA sur tous les problèmes. Le temps de calcul cumulé sur les 11 problèmes montre que les performances de Charibde (94.85s) améliorent d'un ordre de grandeur celles d'Ibex (1312.32s). Les nombres de bissections indiqués suggèrent qu'Ibex adopte une stratégie de résolution au pouvoir filtrant élevé : peu de points de choix sont globalement observés, notamment sur les deux premiers problèmes. A l'inverse, Charibde a davantage recours au partitionnement des boîtes, malgré des temps de calcul souvent inférieurs (problèmes ex6_2_9, ex6_2_11, ex6_2_12, ex7_3_5). Le profil de performance de Charibde et Ibex illustré sur la figure 5.9 affiche l'évolution du pourcentage de problèmes résolus avec le temps (échelle logarithmique).

Reformulation des problèmes ex6_2_6 et ex6_2_8

La fonction objectif a subi une factorisation très simple, consistant à regrouper les termes en facteurs des variables x_1, x_2, x_3 pour obtenir une expression similaire à :

$$f(\boldsymbol{x}) = x_1 f_1(x_1, x_2, x_3) + x_2 f_2(x_1, x_2, x_3) + x_3 f_3(x_1, x_2, x_3) \tag{5.11}$$

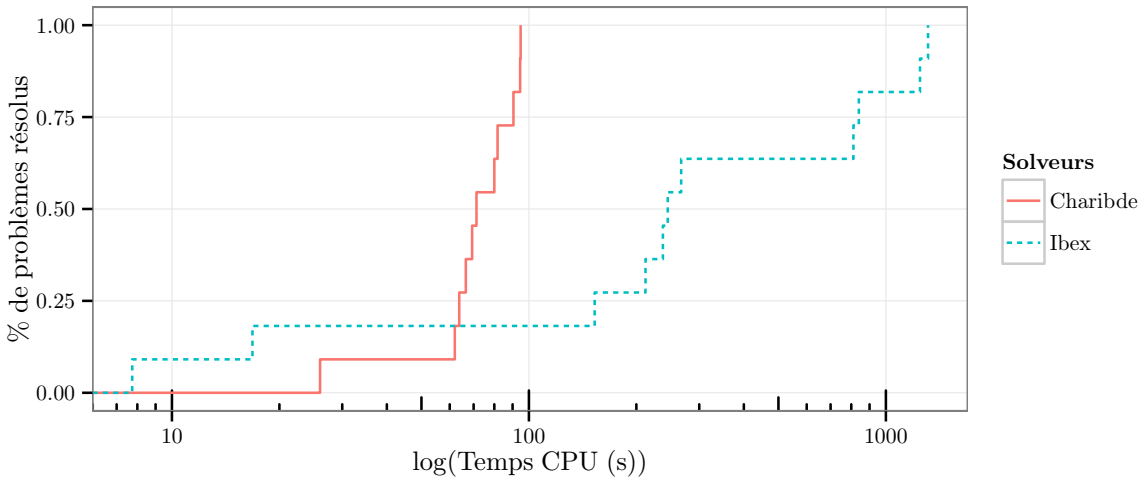


FIGURE 5.9 – Profil de performance : pourcentage de problèmes résolus par Charibde et Ibex en fonction du temps (échelle logarithmique)

où f_1, f_2 et f_3 sont composées de termes logarithmiques et de constantes. Cette manipulation symbolique permet de réduire le problème de dépendance associé à l'expression de f et d'améliorer la qualité de l'encadrement par AI.

Reformulation des problèmes ex6_2_9, ex6_2_11 et ex6_2_12

Une manipulation similaire aux problèmes ex6_2_6 et ex6_2_8 a été appliquée à la fonction objectif. Cette dernière contient également de nombreux termes logarithmiques sujets au problème de dépendance. Une simplification basique (division par un terme strictement positif) puis l'application d'une propriété du logarithme permettent de s'affranchir du problème de dépendance au sein de chaque terme :

$$\log(\frac{x_i}{ax_i + bx_j + cx_k}) = \log(\frac{1}{a + \frac{bx_j+cx_k}{x_i}}) = -\log(a + \frac{bx_j + cx_k}{x_i}) \tag{5.12}$$

Note sur le problème ex7_2_3

Le problème ex7_2_3 apparaît comme un cas à part. Sa résolution semble difficile pour les meilleurs solveurs (544s pour Ibex) : [Araya 12] explique en effet que ce problème ne converge en temps raisonnable qu'avec un calcul de gradients par la variante récursive de Hansen. En revanche, la convergence est atteinte par Charibde en 1.6s, car la solution optimale $\tilde{f} = 7049.248020528667439$ très rapidement obtenue par l'ED (4663 générations, soit 0.31s) intensifie les coupes du BCI de manière surprenante.

En procédant par dichotomie, il peut être établi que le BCI de Charibde converge en moins de 2s lorsqu'un majorant initial $\tilde{f} \in [7049.248020\mathbf{528667439}, 7049.248020\mathbf{5344641}]$ lui est fournie. Ceci suggère donc que l'algorithme de BCI converge rapidement sur ce

problème lorsque le minimum global est connu à $5.8 \cdot 10^{-9}$ près, une valeur inférieure à la précision demandée !

5.5 Nouveaux minima globaux de problèmes difficiles

Nous présentons de nouveaux résultats optimaux et des preuves d'optimalité de résultats connus pour les fonctions Michalewicz [Michalewicz 96a], Sine Envelope Sine Wave, Eggholder [Whitley 96], Keane [Keane 94] et Rana [Whitley 96]. Nous fournissons les minima globaux de différentes instances et les solutions correspondantes.

La sous-section 5.5.1 présente les expressions, domaines et meilleures solutions connues des cinq fonctions de test. Les résultats optimaux certifiés par Charibde sont fournis dans la sous-section 5.5.2. Une comparaison entre Charibde, un algorithme à ED et un algorithme de BCI sur une instance de chaque fonction est présentée dans la sous-section 5.5.3. Dans la sous-section 5.5.4, Charibde est comparé à des solveurs mathématiques, des algorithmes à population et le solveur de programmation non-linéaire Couenne [Belotti 09].

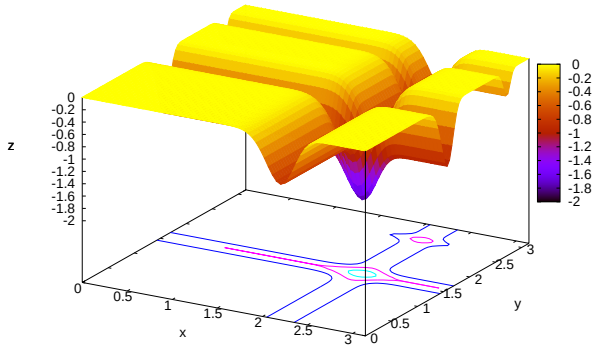
5.5.1 Fonctions de test

Les expressions et les domaines de définition des cinq fonctions de test sont donnés dans le tableau 5.5. La figure 5.10 illustre l'aspect multimodal des fonctions de test pour $n = 2$.

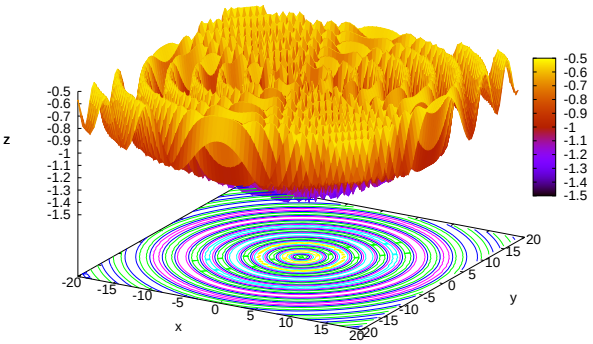
TABLE 5.5 – Expressions et domaines des fonctions de test multimodales

Fonction	Expression de f_n	Domaine
Michalewicz	$-\sum_{i=1}^{n} \sin(x_i) \left[\sin(\frac{ix_i^2}{\pi})\right]^{20}$	$[0, \pi]^n$
Sine Envelope	$-\sum_{i=1}^{n-1} \left(0.5 + \frac{\sin^2(\sqrt{x_{i+1}^2+x_i^2}-0.5)}{(0.001(x_{i+1}^2+x_i^2)+1)^2}\right)$	$[-100, 100]^n$
Eggholder	$-\sum_{i=1}^{n-1} \left[(x_{i+1}+47)\sin\left(\sqrt{\lvert x_{i+1}+47+\frac{x_i}{2}\rvert}\right) + x_i \sin\left(\sqrt{\lvert x_i-(x_{i+1}+47)\rvert}\right)\right]$	$[-512, 512]^n$
Keane	$-\frac{\lvert\sum_{i=1}^{n}\cos^4(x_i)-2\prod_{i=1}^{n}\cos^2(x_i)\rvert}{\sqrt{\sum_{i=1}^{n} ix_i^2}}$ s.c. $0.75 \leq \prod_{i=1}^{n} x_i$ et $\sum_{i=1}^{n} x_i \leq 7.5n$	$[0, 10]^n$
Rana	$\sum_{i=1}^{n-1} \left(x_i \cos\sqrt{\lvert x_{i+1}+x_i+1\rvert}\sin\sqrt{\lvert x_{i+1}-x_i+1\rvert} + (1+x_{i+1})\sin\sqrt{\lvert x_{i+1}+x_i+1\rvert}\cos\sqrt{\lvert x_{i+1}-x_i+1\rvert}\right)$	$[-512, 512]^n$

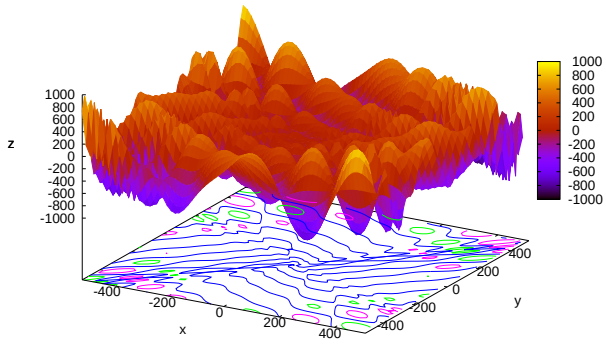
1. La fonction Michalewicz est une fonction séparable et hautement multimodale (de l'ordre de $n!$ optima locaux). Les meilleurs solutions jusqu'à 50 variables, obtenues



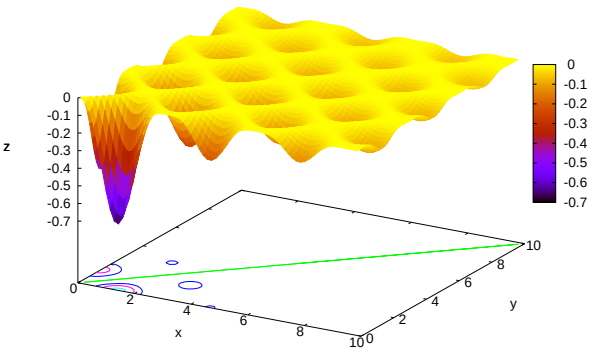
(a) Michalewicz



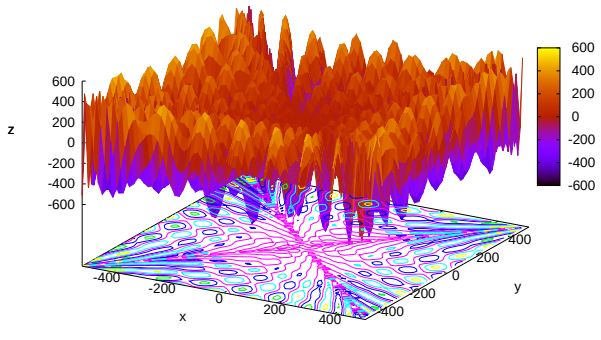
(b) Sine Envelope Sine Wave



(c) Eggholder



(d) Keane



(e) Rana

FIGURE 5.10 – Fonctions de test multimodales ($n = 2$)

par un algorithme à essaim particulaire répulsif, sont données dans [Mishra 06] : $f^*_{10} = -9.6602, f^*_{20} = -19.6370, f^*_{30} = -29.6309, f^*_{50} = -49.6248$.

Peu de résultats ont été obtenus par des méthodes déterministes ; la surface accidentée de la fonction rend en effet la recherche du minimum global difficile pour les méthodes d'optimisation. [Alliot 12a] prétend avoir prouvé l'optimalité de la solution pour $n = 12$ avec une précision $\varepsilon = 10^{-4}$ en 6000s : $f^*_{12} = -11.64957$. Dans leur implémentation, les boîtes $\boldsymbol{X}$ satisfaisant $w(\boldsymbol{X}) \leq 10^{-3}$ sont éliminées de la file de priorité sans être davantage explorées. Puisque la recherche n'est pas exhaustive, l'optimalité ne saurait être certifiée. En comparaison, Charibde atteint la convergence sur le même problème en 0.03s.

2. La meilleure solution connue pour la fonction Sine Envelope Sine Wave est $f^*_2 = f_2(-1.1773, -1.6985) = -1.4915$ [Pohl 10].
3. La meilleure solution connue pour la fonction Eggholder est $f^*_2 = -959.641$ [Oplatková 08].
4. Les meilleures solutions connues pour la fonction Keane sont $f^*_2 = -0.36497975$, $f^*_3 = -0.51578550$, $f^*_4 = -0.62228103$ et $f^*_5 = -0.63444869$[Kang 02].
5. La meilleure solution connue pour la fonction Rana est $f^*_2 = f_2(-488.63, 512) = -511.7329$ [Tao 07].

5.5.2 Résultats expérimentaux

Charibde converge en temps raisonnable jusqu'à :
- 70 variables pour la fonction Michalewicz ;
- 5 variables pour la fonction Sine Envelope Sine Wave ;
- 10 variables pour la fonction Eggholder ;
- 5 variables pour la fonction Keane ;
- 7 variables pour la fonction Rana.

Les trois fonctions Eggholder, Keane et Rana contiennent des valeurs absolues, non différentiables en zéro. En s'inspirant de la valeur du sous-différentiel de la fonction valeur absolue [Kearfott 96a], Charibde calcule un encadrement des dérivées partielles par DA :

$$|\cdot|'(X) = \begin{cases} [-1,-1] & \text{si } \overline{X} < 0 \\ [1,1] & \text{si } \underline{X} > 0 \\ [-1,1] & \text{sinon} \end{cases} \tag{5.13}$$

Le tableau 5.6 détaille les temps de calcul (en secondes) moyen et maximum, et le nombre moyen d'évaluations (NE) de f, F et de ses dérivées partielles, après 100 exécutions de Charibde. Par commodité, seules les statistiques jusqu'à $n = 6$ sont présentées pour la fonction Rana. Les hyperparamètres de Charibde utilisés sur les différentes instances sont donnés dans le tableau 5.7.

TABLE 5.6 – Temps CPU moyen et nombre moyen d'évaluations après 100 exécutions de Charibde

	n	Temps moy.	Temps max.	NE f	NE F (ED)	NE F (BCI)	NE ∇F
Michalewicz	10	0.018	0.022	2601	54.5	1015.6	633
	20	0.092	0.13	12987	182.1	8552.9	6623.6
	30	0.32	0.39	41016	397	33403.8	28260.8
	40	1.37	1.49	141978	655.9	146354.4	129950.1
	50	5.09	6.39	427344	852.7	515656.9	468812.3
	60	26.08	34.61	1815786.4	1106.6	2806303.5	2608734
	70	94.14	113.3	5363505.3	1288.8	10088068.6	9482448.7
Sine	2	0.33	0.57	697975	9.7	102105.4	45449.2
	3	2.10	2.15	2657588.1	16.6	473732.4	264384.9
	4	24.29	25,43	21673266.5	31.3	4144215.5	2543524
	5	194.25	195.7	142043888.5	51.4	29603730.4	19362164.1
Eggholder	2	0.0035	0.005	3325	9.1	484.3	163.2
	3	0.05	0.067	73830	69.1	9703.7	5648.1
	4	0.18	0.26	201585	105.1	28070.2	16266.1
	5	1.72	1.87	1523307.6	146	270582.5	181441.3
	6	4.45	5.17	2920999	582.7	628228.7	413157.5
	7	8.37	8.49	5115327.3	146.7	1107179.1	716550.8
	8	28.52	28.7	13250573.8	158.2	3478698.3	2208158
	9	185.47	187.42	76566411.1	550	20526451.9	15084418.4
	10	606.44	621.11	229408972.7	504.3	64625870.1	46654258.9
Keane	2	0.012	0.023	9824.9	99.8	653.9	154.2
	3	0.047	0.061	43334.3	70.8	3140.8	698.6
	4	0.41	0.74	386745.8	157.3	31236.5	6863.2
	5	2.72	3.11	1804713	158.8	188803.6	42587.4
Rana	2	0.011	0.014	12855	39.8	1513.9	439.3
	3	0.13	0.13	199770	41	21645	8068.1
	4	1.44	1.56	1613409.2	64.7	187906.9	78977.9
	5	18.85	19.13	15819144.5	89	2055106.5	957616.6
	6	247.29	248.98	161239558.6	78.5	23158792.9	11753033

TABLE 5.7 – Hyperparamètres de Charibde sur les cinq fonctions de test multimodales

Hyperparamètre	Michalewicz	Sine Envelope	Eggholder	Keane	Rana
ε (précision)	10^{-8}	10^{-6}	10^{-8}	10^{-8}	10^{-8}
NP (taille de la population)	10 à 70	50	50	30	50
W (facteur d'amplitude)	0.7	0.7	0.7	0.7	0.7
CR (taux de croisement)	0	0.9	0.4	0.9	0.5
Bissection	RR	RR	Largest	Largest	RR
Priorité	MaxDist	MaxDist	MaxDist	MaxDist	MaxDist
η (ratio de point fixe)	0	0.8	0.8	0	0.9

Note sur la fonction Michalewicz

La séparabilité du problème permet de déterminer les minima globaux sur des instances plus grandes que celles généralement traitées par des méthodes d'intervalles. Le choix du taux de croisement $CR = 0$ dans l'ED joue pour beaucoup dans la convergence rapide de Charibde sur la fonction Michalewicz : un individu nouvellement généré diffère seulement d'une composante par rapport à son parent. Dès lors, il est aisé d'optimiser indépendamment les termes de f qui dépendent d'une seule variable.

Note sur la fonction Rana

Charibde converge en temps raisonnable jusqu'à 4 variables. Néanmoins, une réécriture de l'expression de f permet de réduire drastiquement le temps de calcul et de porter la taille de la plus grande instance à 7 variables. En appliquant l'identité trigonométrique $\forall (u, v) \in \mathbb{R}^2, \cos u \sin v = \frac{1}{2}(\sin(u+v) - \sin(u-v))$ à f_n, on obtient :

$$\begin{aligned} f_n(\boldsymbol{x}) = \quad & \frac{1}{2} \sum_{i=1}^{n-1} \Big((x_{i+1} + 1 + x_i) \sin(\sqrt{|x_{i+1} + x_i + 1|} + \sqrt{|x_{i+1} - x_i + 1|}) - \\ & (x_{i+1} + 1 - x_i) \sin(\sqrt{|x_{i+1} - x_i + 1|} - \sqrt{|x_{i+1} + x_i + 1|}) \Big) \end{aligned} \tag{5.14}$$

Les temps de convergence de Charibde pour la première expression syntaxique (tableau 5.5) et la seconde (équation 5.14) sont comparés dans le tableau 5.8. TO indique que la convergence n'a pas été atteinte au bout d'une heure (timeout).

TABLE 5.8 – Fonction Rana : comparaison du temps de convergence de Charibde sur deux expressions syntaxiques équivalentes ($NP = 70$, $W = 0.7$, $CR = 0.5$)

n	Temps CPU (s)	
	Première syntaxe	Seconde syntaxe
2	0.25	0.009
3	6.5	0.12
4	254	1.45
5	TO	18.5
6	TO	244
7	TO	3300

Minima globaux et solutions correspondantes

Les minima de la fonction Michalewicz certifiés par Charibde et les solutions correspondantes sont fournis dans les tableaux 5.9 et 5.10. Puisque la fonction est séparable, seule la solution pour $n = 70$ est fournie ; la solution d'une instance de taille $k < 70$ se déduit immédiatement en prenant les k premières composantes de la solution donnée. Les minima globaux et solutions des autres fonctions sont présentés dans le tableau 5.11.

TABLE 5.9 – Minima globaux de la fonction Michalewicz certifiés par Charibde ($\varepsilon = 10^{-8}$)

n	f_n^*
10	-9.66015171564
20	-19.63701359935
30	-29.63088385032
40	-39.62674886468
50	-49.62483231828
60	-59.62314622857
70	-69.62222020764

TABLE 5.10 – Solutions de la fonction Michalewicz pour $n = 1$ à 70

(2.202905,	1.5707963,	1.2849915,	1.9230584,	1.7204697,
1.5707963,	1.4544139,	1.7560865,	1.6557174,	1.5707963,
1.4977288,	1.6966163,	1.6300760,	1.5707963,	1.5175461,
1.6660645,	1.6163286,	1.5707963,	1.5289070,	1.6474563,
1.6077572,	1.5707963,	1.5362725,	1.6349315,	1.6019018,
1.5707963,	1.5414351,	1.6259253,	1.5976479,	1.5707963,
1.5452545,	1.6191375,	1.5944175,	1.5707963,	1.5481947,
1.6138382,	1.5918810,	1.5707963,	1.5505278,	1.6095861,
1.5898364,	1.5707963,	1.5524243,	1.6060986,	1.5881533,
1.5707963,	1.5539962,	1.6031866,	1.5867435,	1.5707963,
1.5553204,	1.6007184,	1.5855456,	1.5707963,	1.5564510,
1.5985997,	1.5845151,	1.5707963,	1.5574277,	1.5967613,
1.5836191,	1.5707963,	1.5582799,	1.5951509,	1.5828331,
1.5707963,	1.5590300,	1.5937286,	1.5821378,	1.5707963)

La figure 5.11, illustrant l'évolution du temps moyen de calcul de Charibde (échelle logarithmique) avec la taille de chacun des problèmes, confirme que la complexité algorithmique d'un algorithme de BBI est exponentielle en le nombre de variables.

Les résultats obtenus suggèrent que la valeur du minimum global f_n^* de quatre des cinq problèmes satisfait une équation affine fonction du nombre de variables n. Les régressions linéaires peuvent être trouvées dans le tableau 5.12, où R^2 est le coefficient de corrélation linéaire.

5.5.3 Bénéfices de l'hybridation

Le tableau 5.13 compare Charibde à ses deux composantes (ED et BCI) sur une instance particulière de chacune des fonctions de test. Les heuristiques MinLb (recherche dans les sous-espaces les plus prometteurs) ou Rand (recherche aléatoire dans l'espace de recherche) se sont avérées les plus efficaces pour l'algorithme de BCI. Charibde utilise quant à lui l'heuristique MaxDist décrite dans la sous-section 5.2.3. Précisons que lorsque l'ED au

TABLE 5.11 – Minima globaux des fonctions Sine Envelope Sine Wave, Eggholder, Keane et Rana certifiés par Charibde

	n	f_n^*	x_n^*
Sine	2	-1.4914953	(-0.086537, 2.064868)
	3	-2.9829906	(1.845281, -0.930648, 1.845281)
	4	-4.4744859	(2.066680, 0.001365, 2.066680, 0.001422)
	5	-5.9659811	(-1.906893, -0.796823, 1.906893, 0.796823, -1.906893)
Eggholder	2	-959.6406627	(512, 404.231805)
	3	-1888.3213909	(481.462894, 436.929541, 451.769713)
	4	-2808.1847922	(482.427433, 432.953312, 446.959624, 460.488762)
	5	-3719.7248363	(485.589834, 436.123707, 451.083199, 466.431218, 421.958519)
	6	-4625.1447737	(480.343729, 430.864212, 444.246857, 456.599885, 470.538525, 426.043891)
	7	-5548.9775483	(483.116792, 438.587598, 453.927920, 470.278609, 425.874994, 441.797326, 455.987180)
	8	-6467.0193267	(481.138627, 431.661180, 445.281208, 458.080834, 472.765498, 428.316909, 443.566304, 457.526007)
	9	-7376.2797668	(482.785353, 438.255330, 453.495379, 469.651208, 425.235102, 440.658933, 454.142063, 468.699867, 424.215061)
	10	-8291.2400675	(480.852413, 431.374221, 444.908694, 457.547223, 471.962527, 427.497291, 442.091345, 455.119420, 469.429312, 424.940608)
Keane	2	-0.3649797	(1.600860, 0.468498)
	3	-0.5157855	(3.042963, 1.482875, 0.166211)
	4	-0.6222810	(3.065318, 1.531047, 0.405617, 0.393987)
	5	-0.6344487	(3.075819, 2.991995, 1.475794, 0.236691, 0.233309)
Rana	2	-511.7328819	(-488.632577, 512)
	3	-1023.4166105	(-512, -512, -511.995602)
	4	-1535.1243381	(-512, -512, -512, -511.995602)
	5	-2046.8320657	(-512, -512, -512, -512, -511.995602)
	6	-2558.5397934	(-512, -512, -512, -512, -512, -511.995602)
	7	-3070.2475210	(-512, -512, -512, -512, -512, -512, -511.995602)

TABLE 5.12 – Valeur supposée du minimum global en fonction de la taille du problème

Fonction	f_n^*	R^2
Michalewicz	$-0.9994729257n + 0.3467746311$	0.9999998949
Sine Envelope	$-1.49150n + 1.49150$	1
Eggholder	$-915.61991n + 862.10466$	0.9999950
Rana	$-511.70430n + 511.68714$	1

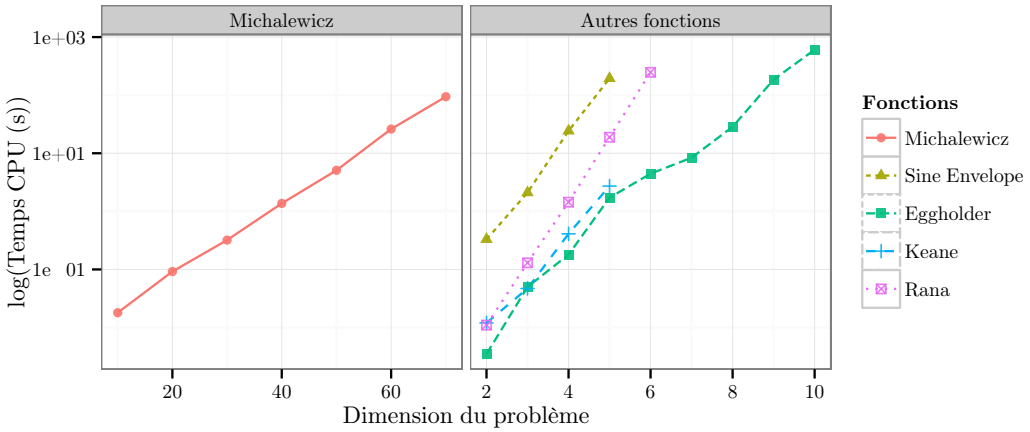


FIGURE 5.11 – Evolution du temps de calcul de Charibde avec la taille du problème

sein de Charibde converge vers le minimum global, le processus poursuit son exécution jusqu'à l'obtention de la preuve d'optimalité du BCI. Le nombre d'évaluations de la fonction objectif en AF (NE f) présenté est donc plus élevé que le nombre effectif d'évaluations requis pour atteindre le minimum global.

Les algorithmes individuels ont des comportements variés sur les différents problèmes de test. L'ED converge prématurément sur la fonction Michalewicz (écart de 10^{-3}) et la fonction Rana. Elle atteint le minimum global pour les fonctions Sine Envelope Sine Wave, Eggholder et Keane. Le BCI converge en temps raisonnable pour quatre des cinq problèmes. L'échec de la convergence pour la fonction Michalewicz est signalé par le sigle TO (timeout) ; le meilleur résultat obtenu dans le temps imparti est obtenu avec l'heuristique Rand.

Charibde converge en 4.9s sur la fonction Michalewicz. Le minimum fourni par l'ED, bien que sous-optimal, permet de couper une grande partie de l'espace de recherche. En échange, le BCI relance la convergence prématurée de l'ED lorsqu'il améliore la meilleure solution connue. Charibde surpasse finalement largement le BCI seul. Les résultats sur les autres instances montrent que Charibde améliore systématiquement le temps de convergence par rapport au BCI seul : le gain relatif en temps de Charibde par rapport au BCI est donné à la ligne Gain /BCI (%). Il varie sur ces instances entre 27.8 et 75.2% (hors Michalewicz). Notre heuristique de recherche MaxDist maintient une taille maximale $|\mathcal{L}|_{max}$ relativement faible de la file de priorité $\mathcal{L}$ dans Charibde (entre 9 et 142 boîtes), comparé au BCI seul (entre 4605 et 870564 boîtes). Ceci permet de transmettre la file de boîtes restantes à l'ED à moindre coût.

La figure 5.12 illustre l'évolution du meilleur majorant $\tilde{f}$ en fonction du temps de calcul des trois algorithmes. Elle confirme qu'un algorithme standard de BCI, même muni de l'heuristique MinLb considérée comme performante, n'est en général pas capable d'obtenir rapidement un bon majorant du minimum global. Si le gain de Charibde par rapport au BCI seul est encourageant et s'accentue généralement avec le nombre de variables sur une instance donnée, les bénéfices de l'hybridation sont moins flagrants que sur les problèmes

contraints. Les problèmes traités dans cette section sont hautement multimodaux (présence de termes trigonométriques) et souffrent fortement du problème de dépendance en raison des nombreuses occurrences de chaque variable. La connaissance d'un bon minimum local (ou global) fourni par l'ED n'est en général pas suffisante pour accélérer de manière significative les coupes de l'espace de recherche sans avoir recours au partitionnement. En revanche, les problèmes contraints précédemment traités (section 5.4) sont plus simples à résoudre dans la mesure où la surestimation calculée sur la fonction objectif et les contraintes est plus faible, et où la difficulté majeure réside dans la satisfaction du système de contraintes. Dans ce cas, disposer d'une bonne solution réalisable grâce à l'AE semble porter ses fruits.

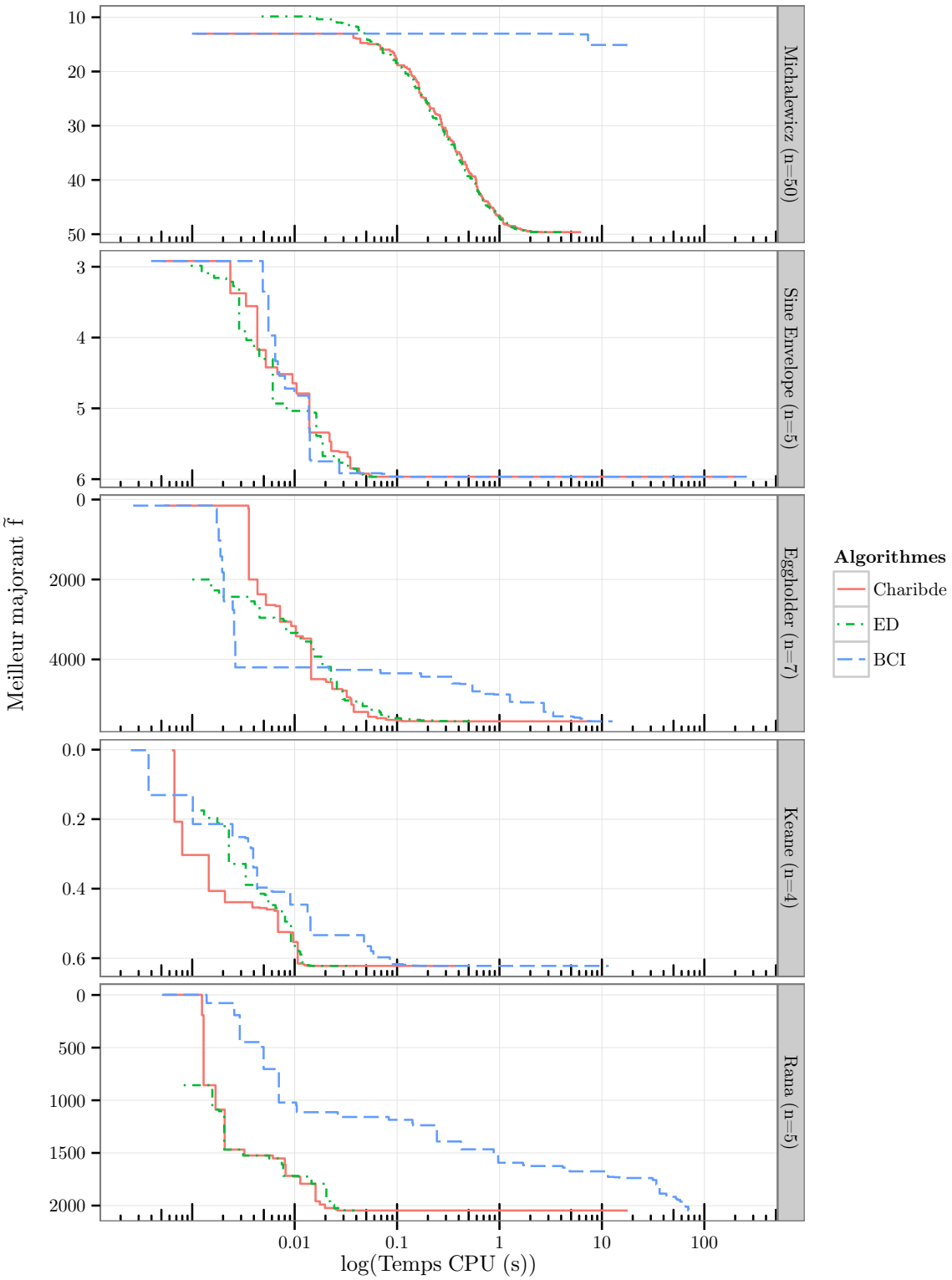


FIGURE 5.12 – Evolution du meilleur majorant de Charibde, de l'évolution différentielle et du branch and contract par intervalles en fonction du temps de calcul

TABLE 5.13 – Comparaison de Charibde, de l'évolution différentielle et du branch and contract par intervalles sur les fonctions de test

		Michalewicz ($n = 50$)	Sine Envelope ($n = 5$)	Eggholder ($n = 7$)	Rana ($n = 5$)	Keane ($n = 4$)
ED	$\tilde{f}$	-49.6237668	-5.9659811	-5548.9775483	-2021.9520121	-0.6222810
	Temps (s)	4.22	0.060	0.51	0.076	0.028
	NE f	335580	49900	350350	51650	13500
	NE F	878	74	142	93	74
	NP	60	50	50	50	30
	W	0.7	0.7	0.7	0.7	0.7
	CR	0	0.9	0.4	0.5	0.9
BCI	$\tilde{f}$	-16.09385	**-5.9659811**	**-5548.9775483**	**-2046.8320657**	**-0.6222810**
	Temps (s)	TO	269.9	12.7	75.76	0.56
	Bissections	-	1934639	105659	756812	6059
	NE F	-	29766788	1475165	6240504	42755
	NE ∇F	-	19462956	983730	3269639	9180
	$\lvert\mathcal{L}\rvert_{max}$	-	870564	50330	481099	4605
	Bissection	RR	RR	Largest	RR	Largest
	Priorité	Rand	MinLb	MinLb	MinLb	MinLb
	η	0.9	0.8	0.9	0.9	0
Charibde	$\tilde{f}$	**-49.6248323**	**-5.9659811**	**-5548.9775483**	**-2046.8320657**	**-0.6222810**
	Temps (s)	6.3	194.9	7.85	18.8	0.36
	Bissections	11823	1922833	55177	240945	3047
	NE f	520800	108944542	4756387	16478673	342780
	NE F (ED)	809	52	141	64	104
	NE F (BCI)	673529	29603444	980929	2049552	26523
	NE ∇F	614451	19361735	689304	954871	5860
	$\lvert\mathcal{L}\rvert_{max}$	142	60	43	47	9
	NP	60	50	50	50	30
	W	0.7	0.7	0.7	0.7	0.7
	CR	0	0.9	0.4	0.5	0.9
	Bissection	RR	RR	Largest	RR	Largest
	Priorité	MaxDist	MaxDist	MaxDist	MaxDist	MaxDist
	η	0	0.8	0.9	0.9	0
	Gain /BCI (%)	-	27.8	38.2	75.2	35.7

5.5.4 Comparaison de solveurs

Une comparaison entre Charibde et sept solveurs du serveur NEOS [2] sur une instance particulière de chacune des fonctions de test est présentée dans le tableau 5.14. Lorsque disponible, le nombre d'évaluations de la fonction objectif est affiché sous la valeur du minimum de chaque solveur ; à défaut, le temps de calcul (en secondes) est mentionné. Notons que le solveur BARON [Sahinidis 96] ne supporte pas les fonctions trigonométriques.

Pour mettre en évidence le caractère multimodal des fonctions considérées, trois solveurs d'optimisation locale (Ipopt, LOQO, MINOS) sont comparés sur les fonctions de test. Ils requièrent généralement peu d'itérations pour atteindre un minimum local depuis une solution initiale. La qualité de ce minimum dépend de cette solution initiale et de la taille des bassins d'attraction de la fonction. Ces solveurs s'avèrent peu efficaces sur les cinq fonctions considérées et convergent vers des minima locaux de faible qualité. Parmi les algorithmes à population, PGAPack converge vers des minima systématiquement meilleurs que ceux obtenus par PSwarm, à un coût toutefois supérieur. Les hyperparamètres par défaut de NEOS ont été conservés ; les résultats des deux algorithmes seraient probablement meilleurs en réglant finement leurs hyperparamètres. Les tirets dans la colonne Keane indiquent que PGAPack et PSwarm ne gèrent pas les contraintes d'inégalité.

Couenne est un solveur d'optimisation global qui procède à une exploration exhaustive de l'espace de recherche ; il s'agit d'un algorithme de branch and bound spatial basé sur des techniques de reformulation. Bien qu'il soit considéré comme un des meilleurs solveurs actuellement, Couenne n'est pas fiable : les sous-estimations et surestimations obtenues en relaxant la fonction objectif et les contraintes en AF ne sont pas conservatives et sont sujettes à des approximations numériques ; la correction du résultat ne saurait donc être garantie. Ce problème est mis en évidence dans le tableau 5.14 : les minima globaux obtenus par Couenne pour les fonctions Michalewicz, Sine Envelope Sine Wave, Eggholder et Keane ne sont pas corrects en comparaison des minima *certifiés* par Charibde (à ε près). Les décimales incorrectes sont soulignées.

Ces résultats montrent que Charibde est fortement compétitif par rapport à Couenne au niveau du temps de calcul (excepté sur la fonction Sine Envelope Sine Wave), tout en garantissant l'optimalité de la solution à ε près. Charibde converge plus rapidement que Couenne sur la fonction Michalewicz (rapport 54), la fonction Eggholder (rapport 5.6), la fonction Rana (rapport 1.1) et la fonction Keane (rapport 5.6). Couenne converge plus rapidement que Charibde sur la fonction Sine Envelope Sine Wave (rapport 487), mais fournit un majorant trop bas, incorrect à la troisième décimale (-5.9660007 contre -5.9659811). Couenne procède donc à des coupes non fiables de l'espace de recherche et converge rapidement vers une solution erronée.

L'évolution de la meilleure solution de Couenne et Charibde au cours du temps (échelle logarithmique) est illustrée dans la figure 5.13. Les temps intermédiaires des autres solveurs ne sont pas connus. A l'image de la collaboration entre ED et BCI au sein de Charibde, Couenne obtient rapidement un bon majorant du minimum global en invoquant le solveur non-linéaire Ipopt.

2. `http://www.neos-server.org/neos/solvers/`

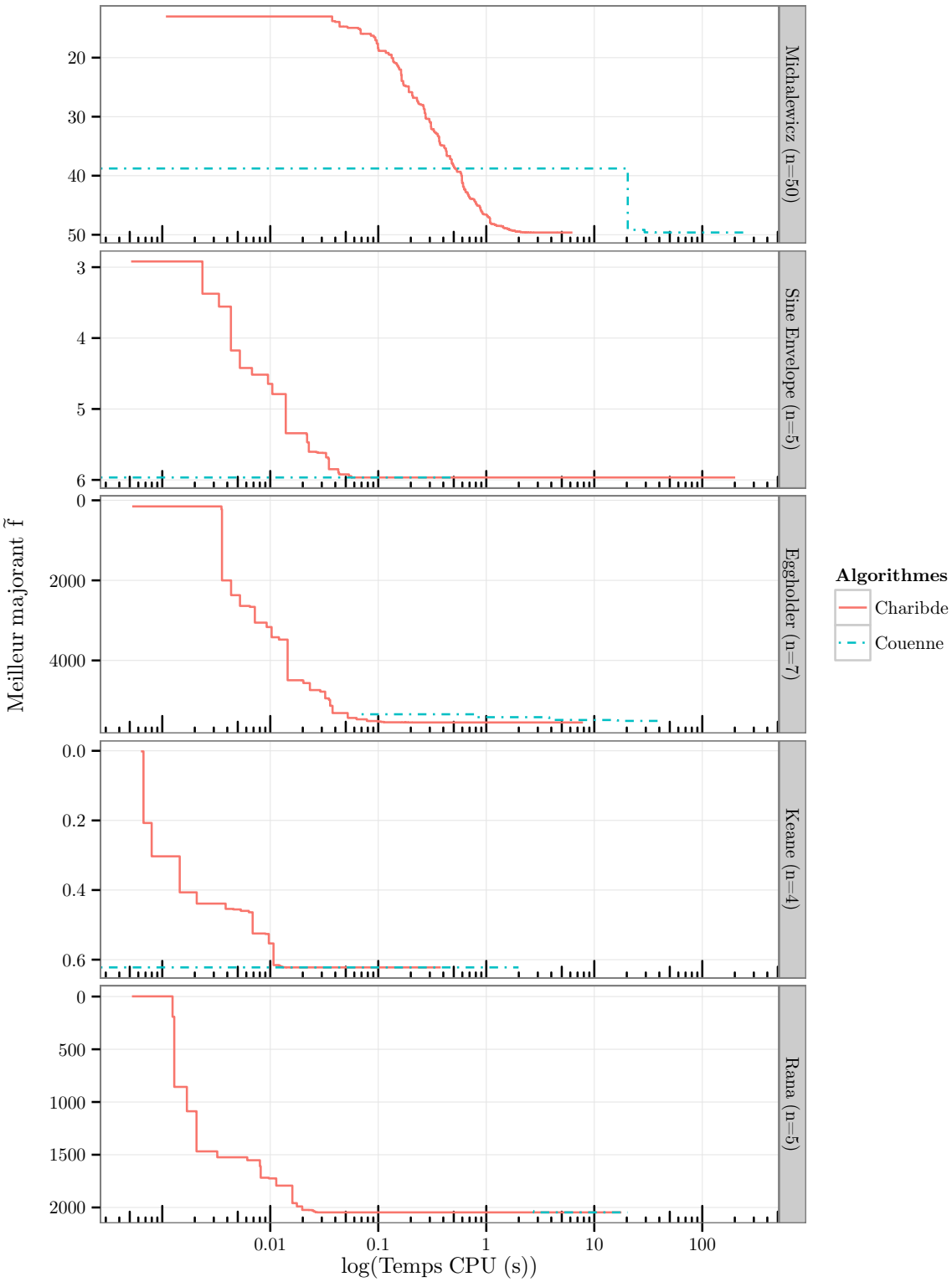


FIGURE 5.13 – Evolution du meilleur majorant de Charibde et Couenne en fonction du temps de calcul

TABLE 5.14 – Comparaison de Charibde et sept solveurs du serveur NEOS sur les fonctions de test

	Michalewicz ($n = 50$)	Sine Envelope ($n = 5$)	Eggholder ($n = 7$)	Rana ($n = 5$)	Keane ($n = 4$)
Ipopt	-19.773742	-5.8351843	-5199.5967304	-75.512076	-0.2010427
	(167 éval)	(24 éval)	(7 éval)	(16 éval)	(8 éval)
LOQO	-0.0048572	-5.8351843	-44.45892854	-69.5206	-0.0983083
	(88 éval)	(17 éval)	(5406 éval)	(138 éval)	(50 éval)
MINOS	0	-5.87878	-5199.59673	-233.592	-0.2347459
	(3 éval)	(38 éval)	(3 éval)	(1 éval)	(3 éval)
PGAPack	-37.60465	-5.569544	-4369.204	-2091.068	-
	(9582 éval)	(9615 éval)	(9593 éval)	(9622 éval)	-
PSwarm	-24.38158	-5.835182	-3429.485	-1595.056	-
	(2035 éval)	(2049 éval)	(2054 éval)	(2046 éval)	-
Couenne	-49.619042	-5.9660007	-5510.513933430007	**-2046.8320657**	-0.6222999
	(265s)	(0.4s)	(44s)	(20.3s)	(2s)
Charibde	**-49.6248323**	**-5.9659811**	**-5548.9775483**	**-2046.8320657**	**-0.6222810**
	(4.9s)	(194.9s)	(7.85s)	(18.8s)	(0.36s)

5.6 Conclusion

Notre algorithme coopératif Charibde combine l'efficacité des AE et la fiabilité de l'AI afin d'accélérer la convergence des méthodes conventionnelles sur les problèmes d'optimisation difficiles et certifier l'optimalité de la solution. Les deux processus échangent majorant du minimum global, meilleur individu et domaine restant afin d'intensifier les coupes de l'espace de recherche et d'éviter la convergence vers des minima locaux.

Des tests numériques montrent que Charibde améliore d'un ordre de grandeur les performances du solveur Ibex sur certains problèmes contraints de la base COCONUT sélectionnés pour leur difficulté. Nous avons également fourni de nouveaux minima globaux pour cinq problèmes multimodaux.

Le cadre d'application de Charibde s'avère très général, à l'instar de l'algorithme de [Alliot 12a]. Néanmoins, de nombreuses techniques d'accélération permettent d'exploiter la structure du problème (décomposabilité, dérivabilité). En particulier, la combinaison de propagation de contraintes et de différentiation automatique (voir section 4.5) exploite la réduction des nœuds intermédiaires pour obtenir un encadrement des dérivées partielles plus précis.

Chapitre 6

Contributions à la résolution de conflits aériens

Sommaire

6.1 Conflits aériens 112
6.1.1 Séparation 112
6.1.2 Approches pour la résolution de conflits 113
6.2 Approche centralisée par algorithmes évolutionnaires 114
6.2.1 Modèle adopté 114
6.2.2 Un problème d'optimisation sous contraintes 115
6.2.3 Simulateur de trafic 116
6.2.4 Problèmes de test 117
6.2.5 Résultats expérimentaux 118
6.2.6 Conclusion et limites du modèle 128
6.3 Manœuvres latérales optimales 128
6.3.1 Expression de la position 129
6.3.2 Contracteur implémenté 130
6.3.3 Problèmes de test 134
6.3.4 Résultats expérimentaux 137
6.4 Perspectives 141

La résolution de conflits aériens, modélisée sous la forme d'un problème d'optimisation sous contraintes, est un problème fortement combinatoire et très contraint. Il consiste à générer des trajectoires les plus proches possibles des trajectoires initiales, tout en respectant la séparation entre avions à tout instant.

Les différentes approches de résolution de conflits sont introduites dans la section 6.1. Dans la section 6.2, nous proposons une approche en changement de cap paramétrée par des variables continues et résolue par des algorithmes à population. Nous adoptons dans la section 6.3 un modèle sous contraintes universellement quantifiées que nous résolvons à l'optimalité avec des méthodes d'intervalles.

6.1 Conflits aériens

Le trafic aérien est organisé en réseaux de routes aériennes, constituées de segments convergeant vers des points de virages ou waypoints (figure 6.1). Parfois, des équipements radio-électriques sont placés au sol en ces points pour permettre aux avions de se situer à l'aide d'équipements de radionavigation à bord.

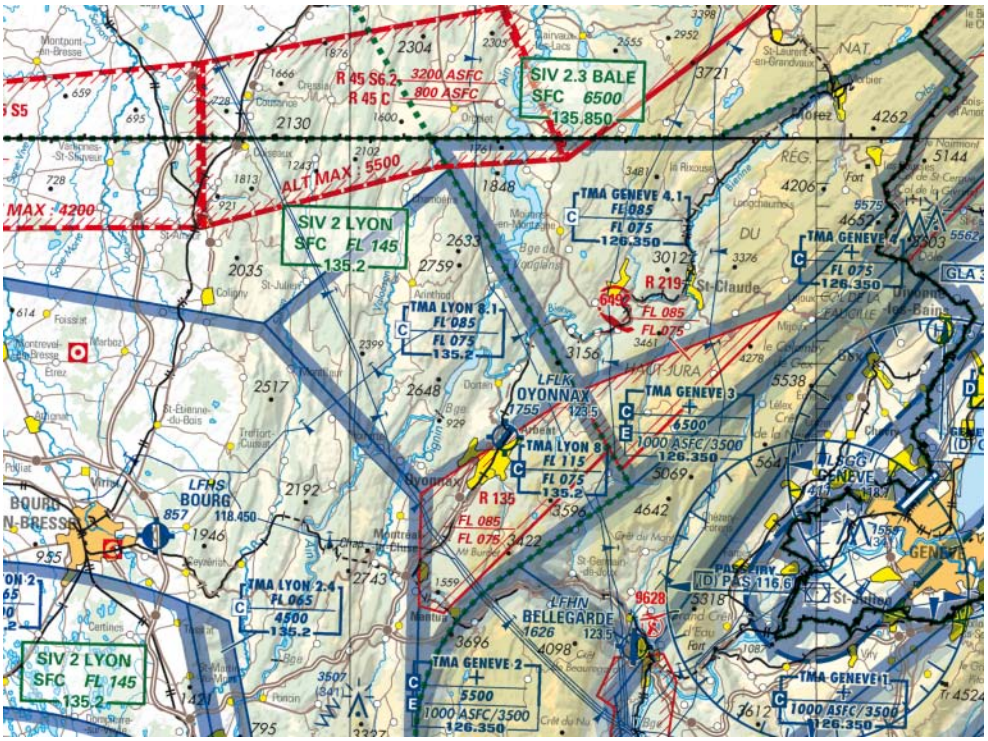


FIGURE 6.1 – Carte aéronautique des routes aériennes

6.1.1 Séparation

Le contrôle en phase de croisière (ou en route) consiste à gérer la progression des avions en dehors des zones proches des aéroports. L'objectif principal du service de contrôle au sol est d'empêcher les collisions entre avions en maintenant à tout instant une séparation des trajectoires d'au moins 5 milles nautiques (9.3km) sur le plan horizontal et 1000 pieds (305m) verticalement (figure 6.2).

Un *conflit* est une perte de séparation entre les prévisions de trajectoires. La résolution de conflits entre trajectoires d'avions consiste à manœuvrer les appareils pour garantir leur séparation à tout instant. Différents types de manœuvres sont généralement considérés : les allocations de niveaux de vol, la résolution en vitesse et les évitements latéraux. L'*allocation de niveaux de vol* concerne les changements d'altitude, à raison d'un niveau de vol disponible tous les 100 pieds (30m). La *résolution en vitesse* est une modification de la vitesse de

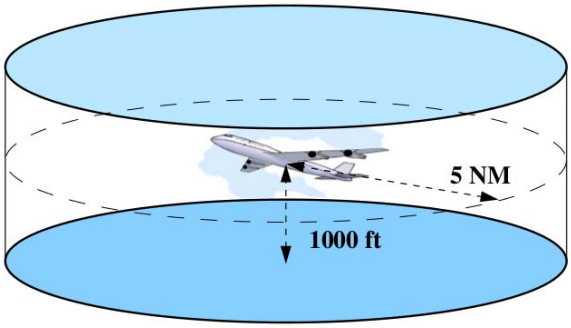


FIGURE 6.2 – Espace de séparation pour la phase de croisière

l'avion en tenant compte de contraintes de bornes sur l'accélération. L'*évitement latéral* est un changement de cap dans le plan horizontal.

6.1.2 Approches pour la résolution de conflits

Deux types d'approches existent pour la résolution de conflits : les approches *autonomes* (ou distribuées) dans lesquelles chaque avion est équipé d'un système de contrôle embarqué et n'a qu'une visibilité limitée des appareils qui l'entourent, et les approches *centralisées* dans lesquelles les contrôleurs aériens au sol ont un aperçu global du trafic.

Les approches autonomes ont été conçues dans les années 1990 dans le cadre de la méthode de contrôle aérien Free Flight. Elles laissent la liberté au pilote de s'affranchir en partie des restrictions de vol, et de sélectionner des trajectoires en temps réel. [Zeghal 98] exploite des champs de forces : la cohérence des mouvements de plusieurs objets mobiles repose sur la définition de force glissante entre deux objets. Il n'existe néanmoins aucune garantie d'optimalité au-delà de deux avions. FACES [Durand 00] est un solveur de conflits embarqué dédié à l'espace aérien de Free Flight, dans lequel la coordination entre appareils est garantie par une stratégie d'allocation de jeton. Les conflits sont résolus par un algorithme A^* après que les vols ont été classés par niveau de priorité. Cette approche a fait l'objet de tests intensifs impliquant un petit nombre d'avions dans le simulateur CATS [Alliot 97] ; seule l'optimalité locale est assurée. Un algorithme de détection et d'analyse des conflits potentiels en phase de croisière [Isaacson 97] génère l'ensemble des conflits futurs probables, et assiste le contrôleur dans la tâche de résolution de son secteur. Les conflits sont détectés par comparaison des trajectoires à des instants consécutifs proches.

Les seules approches centralisées permettant de résoudre efficacement des conflits impliquant plus d'une vingtaine d'avions en recherchant un optimum global, sont la programmation linéaire mixte [Pallottino 02, Cafieri 10] et les AE [Durand 96, Delahaye 10, Vanaret 12]. Dans [Pallottino 02], les hypothèses sur la vitesse des avions sont fortes et peu réalistes dans le cadre d'une utilisation opérationnelle : vitesses constantes pour des résolutions par évitement latéral, ou changements de vitesses instantanés pour des résolutions en vitesse. Dans [Durand 96], un AG fait évoluer une population d'individus encodant des

manœuvres de déviations latérales pour les avions initialement en conflit. La modélisation proposée s'appuie sur des variables discrétisées (temps de début et de fin de manœuvres, angles de déviation). Les hypothèses relativement réalistes, permettant notamment la prise en compte d'incertitudes sur les vitesses, donnent des résultats très convaincants. Dans [Delahaye 10], les déviations latérales sont approximées par des B-splines. Les changements de cap continus rendent l'approche inexploitable par les systèmes de gestion de vol ou flight management system (FMS). Dans [Vanaret 12], des manœuvres rectilignes par morceaux, paramétrées par des variables continues et optimisées par des AE, fournissent une séquence de waypoints directement exploitable par les FMS.

6.2 Approche centralisée par algorithmes évolutionnaires

Nous avons introduit dans [Vanaret 12] un benchmark de problèmes de résolution de conflits, sur lequel nous avons comparé les performances de trois algorithmes à population. La résolution de conflits est modélisée par un problème d'optimisation sous contraintes : le but de la résolution est de générer des trajectoires les plus proches possibles des trajectoires initiales, tout en respectant la séparation entre avions à tout instant. Nous adoptons une approche « boîte noire », dans la mesure où l'évaluation des contraintes de séparation nécessite de simuler les trajectoires grâce à un petit simulateur de trafic.

Notre modèle, basé sur des manœuvres latérales, est identique à celui de [Pallottino 02]. Il pourrait aisément être étendu à des manœuvres verticales et en vitesse. Contrairement à [Durand 96], le début de manœuvre, l'angle et la longueur de déviation sont des variables continues. Nous comparons sur le benchmark proposé trois algorithmes à population manipulant des variables continues : un AG avec un encodage réel, une OEP et une ED. Les résultats obtenus mettent en lumière la supériorité de certains algorithmes sur le problème de résolution de conflits.

6.2.1 Modèle adopté

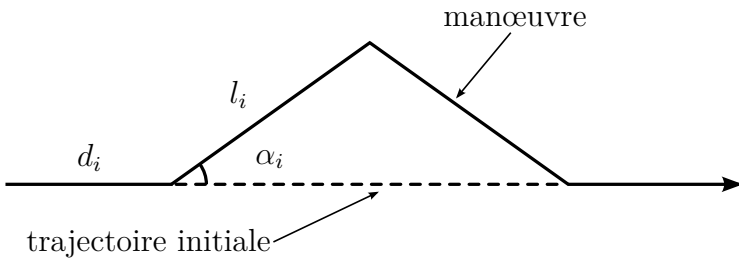


FIGURE 6.3 – Point tournant

La manœuvre par évitement latéral d'une trajectoire i est modélisée par un point tournant (figure 6.3). Elle est caractérisée par un triplet de réels (d_i, α_i, l_i) : l'avion s'écarte

de sa trajectoire initiale après une distance d_i, avec un angle de déviation α_i. Après avoir parcouru le segment de déviation de longueur l_i, l'avion revient à sa trajectoire initiale avec un angle $-2\alpha_i$. Une configuration à N avions est donc représentable par un vecteur de $n = 3N$ variables continues.

6.2.2 Un problème d'optimisation sous contraintes

Contraintes de séparation

Notons $\vec{p_i}(t)$ et $\vec{p_j}(t)$ les positions de deux avions i et j à un instant t. La contrainte de séparation entre les deux avions est donnée par :

$$\forall t \in [0, t_f], \quad S^2 \leq \|\vec{p_i}(t) - \vec{p_j}(t)\|_2^2 \tag{6.1}$$

où $[0, t_f]$ est l'intervalle de temps considéré et $\| \cdot \|_2$ est la norme euclidienne. La figure 6.4 illustre les zones de séparation de cinq avions, représentées dans le plan horizontal par des cercles de diamètre S, centrés sur les appareils, qui ne doivent s'intersecter à aucun moment.

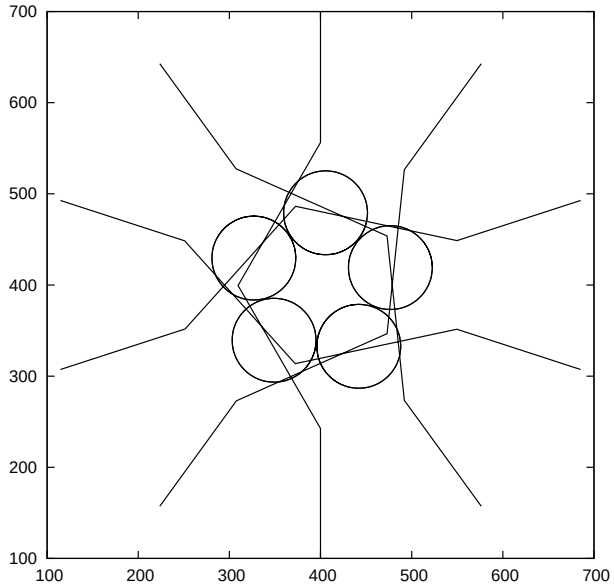


FIGURE 6.4 – Zones de séparation

Fonction objectif

Afin de réduire la déviation induite par la manœuvre, l'angle et la longueur du segment de déviation doivent être les plus faibles possibles. En revanche, le début de la manœuvre doit être repoussé au plus tard, de sorte que les manœuvres prématurées futures soient réduites ou évitées (en raison de la diminution de l'incertitude sur la position des avions). En

notant $\boldsymbol{x} = (d_1, \alpha_1, l_1, \ldots, d_N, \alpha_N, l_N)$ les variables de décision, la fonction objectif s'exprime comme une somme de carrés :

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \left((1 - \frac{d_i}{\overline{d_i}})^2 + (\frac{\alpha_i}{\overline{\alpha_i}})^2 + (\frac{l_i}{\overline{l_i}})^2 \right) \tag{6.2}$$

Le problème d'optimisation sous contraintes modélisant le problème de résolution de conflits aériens est alors le suivant :

$$\begin{array}{ll} \min\limits_{\boldsymbol{x}} & \sum\limits_{i=1}^{N} \left((1 - \frac{d_i}{\overline{d_i}})^2 + (\frac{\alpha_i}{\overline{\alpha_i}})^2 + (\frac{l_i}{\overline{l_i}})^2 \right) \\ \text{s.c.} & d_i \in [\underline{d_i}, \overline{d_i}] \\ & \alpha_i \in [\underline{\alpha_i}, \overline{\alpha_i}] \\ & l_i \in [\underline{l_i}, \overline{l_i}] \\ & \forall i < j, \quad \forall t \in [0, t_f], \quad S^2 \leq \|\vec{p_i}(t) - \vec{p_j}(t)\|^2 \end{array} \tag{6.3}$$

6.2.3 Simulateur de trafic

Pour pallier le manque de résultats dans la littérature concernant la résolution de conflits aériens, nous avons proposé une bibliothèque OCaml encapsulant un petit simulateur de trafic aérien. Une fonction *conflits* calcule la violation des contraintes de séparation pour un vecteur de déviations latérales $\boldsymbol{x}$ donné. Un conflit est ici défini par un intervalle compact de temps pendant lequel la contrainte de séparation entre les avions i et j (équation 6.1) est violée. Puisque plusieurs conflits peuvent avoir lieu entre deux avions, le terme $C_{i,j}$ de la matrice C obtenue en invoquant *conflits*($\boldsymbol{x}$) représente une liste des durées des conflits entre les avions i et j. La matrice C rend donc directement accessibles le nombre et la durée des conflits. Lorsque les segments de trajectoires sont linéaires, les conflits peuvent être détectés en résolvant une équation du second degré (à vitesses constantes) ou une équation du quatrième degré (à accélérations constantes).

Le simulateur est doté de nouvelles fonctionnalités en comparaison de précédents simulateurs, tels que CATS/OPAS [Alliot 97] : les actions possibles (accélérer, suivre une route, etc.) sont modélisées comme les arêtes d'un graphe orienté acyclique (DAG), dont les nœuds déclenchent des événements (par exemple, capture de vitesse ou de waypoint). Les trajectoires sont prédites avec un pas de temps par défaut constant δt et les conflits sont détectés deux à deux en considérant les distances entre avions. Néanmoins, un pas de temps plus faible est considéré lorsqu'un événement est déclenché entre le temps courant t et le temps suivant $t + \delta t$ dans la boucle de simulation. Ceci permet de simuler des déviations latérales dont les variables (d_i, α_i, l_i) sont continues (plutôt que de discrétiser leurs domaines [Durand 96]) et d'utiliser des méthodes d'optimisation continue.

Les plans de vol des avions et leurs états initiaux sont chargés depuis un fichier XML, ainsi que les paramètres de simulation et les domaines des variables paramétrant les déviations de trajectoires. Les fichiers XML standardisés peuvent alors être mis à disposition de la communauté de trafic aérien.

6.2.4 Problèmes de test

Nous considérons deux configurations de test à N avions : une configuration initiale C_N en cercle (figure 6.5, les dimensions sont en milles nautiques) et une configuration B_N en cercle bruité (figure 6.6) :

a) Configuration initiale C_N en cercle : les avions sont arrangés régulièrement autour d'un cercle, et leurs trajectoires convergent simultanément vers le centre du cercle. Deux solutions optimales existent : tous les avions tournent soit à droite, soit à gauche. Ces configurations sont appelées « ronds-points ».

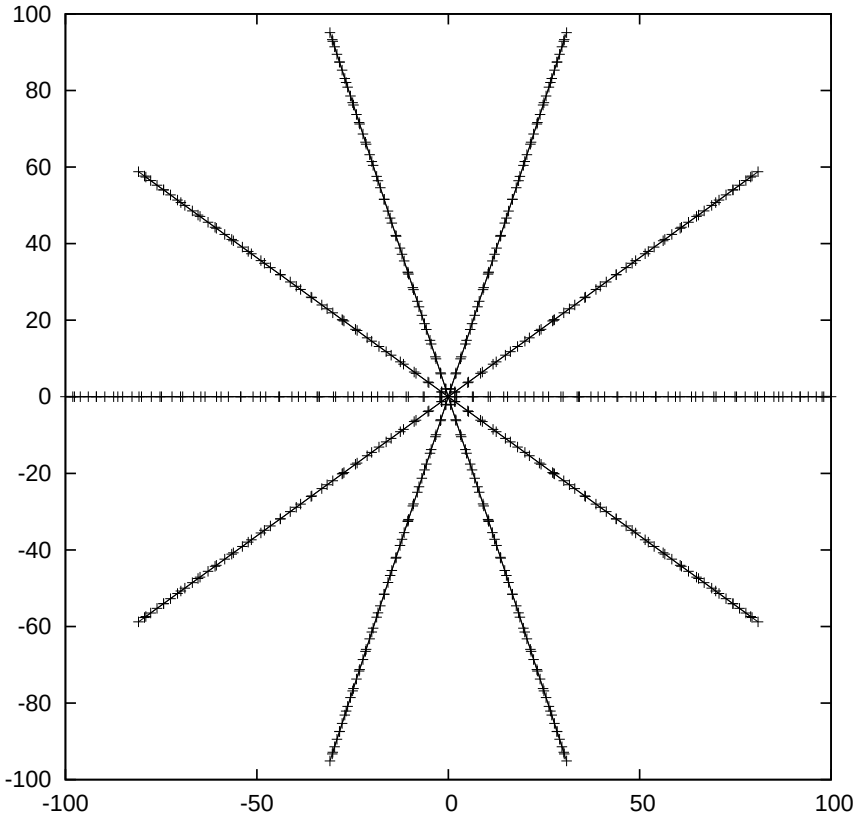


FIGURE 6.5 – Configuration initiale C_{10} en cercle

Il s'agit d'un problème très contraint. Pour s'en convaincre, on peut dénombrer le nombre de situations sans conflits parmi un million de configurations de manœuvres à N avions générées aléatoirement : pour $N = 5$, seules 497 situations satisfont les contraintes de séparation. Pour $N > 8$, aucune des situations générées n'est sans conflit ;

b) Configuration initiale B_N en cercle bruité : bien que très combinatoire, C_N reste un problème symétrique. Afin de complexifier les comportements des avions, nous introduisons une deuxième configuration initiale de test dans laquelle les avions sont à l'identique disposés sur un cercle, et leurs caps par rapport au diamètre du cercle sont

tirés aléatoirement dans $[-\frac{\pi}{6}, \frac{\pi}{6}]$. Le point final de la trajectoire appartient également au cercle.

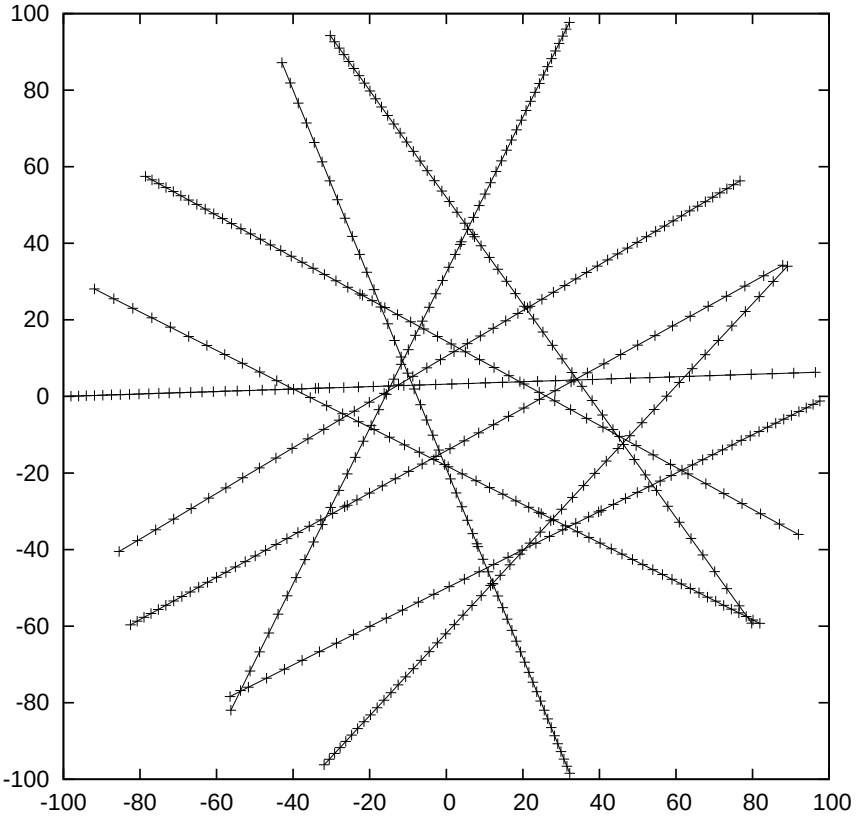


FIGURE 6.6 – Configuration initiale B_{10} en cercle bruité

6.2.5 Résultats expérimentaux

Nous comparons les résolutions obtenues par trois algorithmes à population (un AG, une OEP et une ED) sur les deux configurations C_N et B_N avec une séparation de $S = 5$ NM. Le nombre maximum d'évaluations de la fonction objectif est fixé à 120000.

Les figures 6.9 à 6.13 (en échelle logarithmique sur l'axe des abscisses) décrivent, pour diverses instances du problème, l'évolution du meilleur individu en fonction du nombre d'évaluations de la fonction objectif. Les courbes sont séparées en deux parties : les courbes du haut représentent la durée totale de conflits restants entre les N avions, et les courbes du bas affichent l'évaluation du meilleur individu une fois les conflits résolus. La figure 6.9 montre que les trois algorithmes à population utilisés produisent des solutions relativement proches en terme de valeur objectif pour le problème C_{25} (40.2 pour lED, 46.7 pour l'AG et 49.1 pour l'OEP), bien que le nombre d'évaluations nécessaires pour résoudre les conflits varie grandement (10800 pour l'ED, 43200 pour l'AG et 68250 pour l'OEP). La solution du problème obtenue par une ED, proche de la configuration optimale en rond-point dans

laquelle tous les avions tournent à droite, est présentée en figure 6.7.

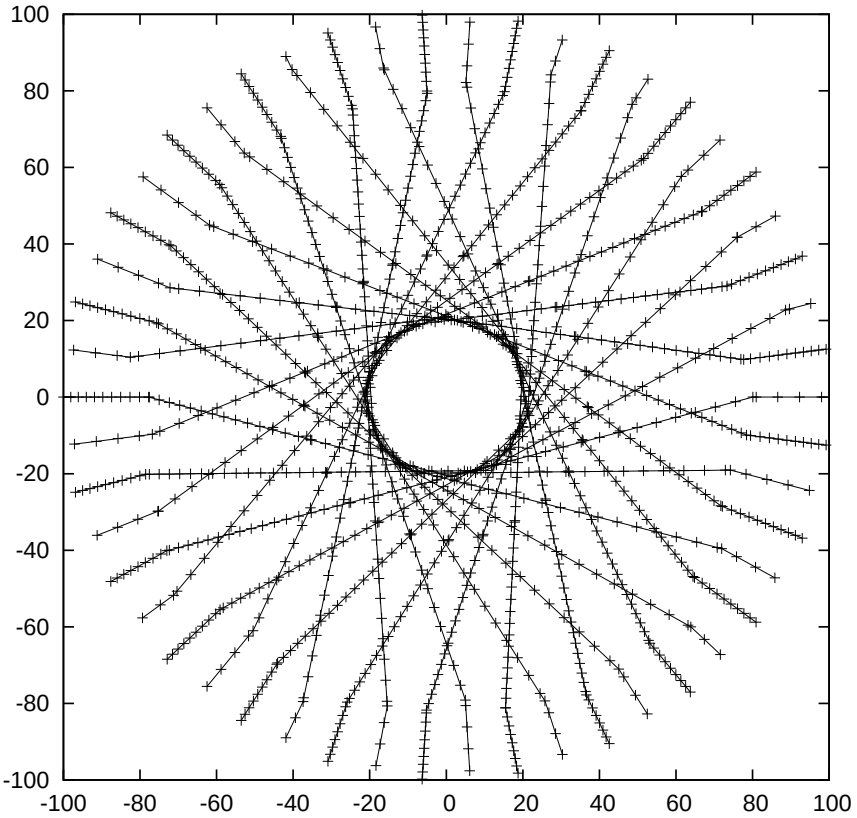


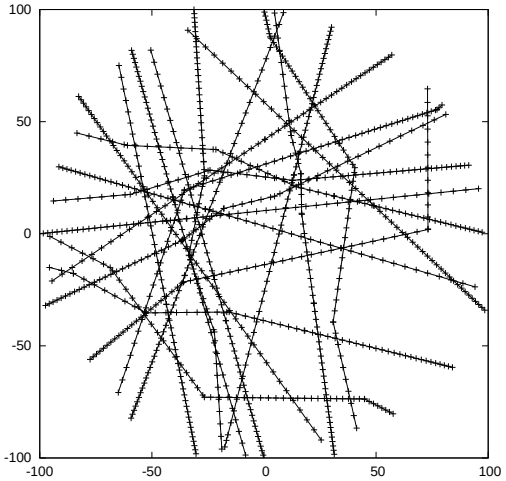
FIGURE 6.7 – Solution du problème C_{25} par un algorithme à évolution différentielle

Les figures 6.10 (problème B_{20}) et 6.11 (problème B_{30}) suggèrent que l'OEP est capable de résoudre les conflits en peu d'itérations sur les instances les plus petites, mais peine à explorer l'espace de recherche sur les instances plus importantes et reste piégée dans des minima locaux. En revanche, l'AG et l'ED semblent dotés de mécanismes d'exploration efficaces, capables de s'échapper des minima locaux. Ces deux problèmes donnent un léger avantage à l'ED. La figure 6.8 illustre la meilleure solution trouvée par chacun des algorithmes à population utilisés sur le problème B_{20}. On notera que, malgré le faible écart entre les évaluations de l'AG et l'ED, leurs solutions apparaissent visuellement différentes.

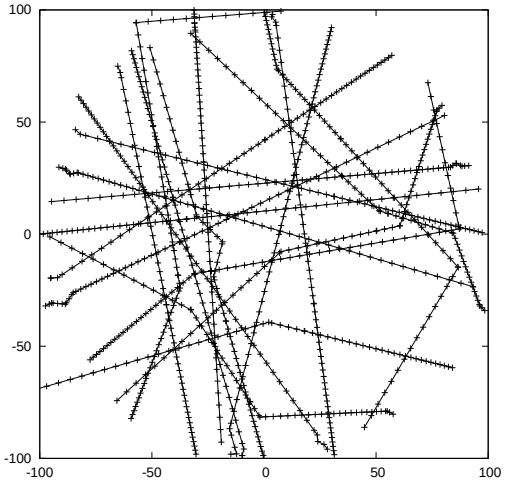
La tendance s'inverse sur les problèmes B_{40} (figure 6.12) et B_{50} (figure 6.13) ; bien qu'un peu plus lent pour résoudre les conflits, l'AG obtient une solution finale meilleure que celle atteinte par l'ED. Quant à l'OEP, les conflits sont de peu résolus pour le problème B_{40} et non résolus pour le problème B_{50}.

Le tableau 6.1 compile les évaluations des meilleurs individus sur 20 exécutions par les trois algorithmes à population. La colonne 3 indique le nombre d'exécutions non résolues, c'est-à-dire où il reste des conflits. Les colonnes 4 à 7 donnent les durées minimum, maximum et moyenne des conflits restants sur les exécutions non résolues et leur écart type. Les colonnes 8 à 11 donnent le coût minimum, maximum et moyen des exécutions sans conflits

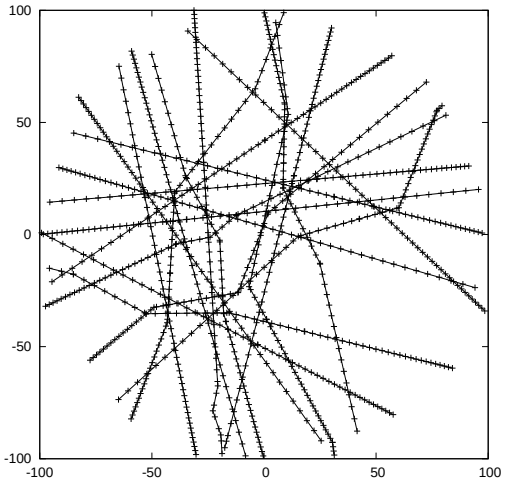
et leur écart type. Les deux premières instances C_{25} et B_{20} sont rapidement résolues par l'AG, l'OEP et l'ED : ceci suggère que les trois algorithmes gèrent bien la combinatoire des problèmes. Lorsque la taille du problème augmente, l'OEP perd rapidement en efficacité, en témoigne le nombre d'exécutions non résolues. Le nombre d'exécutions résolues par l'AG diminue peu sur les grandes instances, tandis que l'ED résout toutes les exécutions.



(a) Algorithme génétique

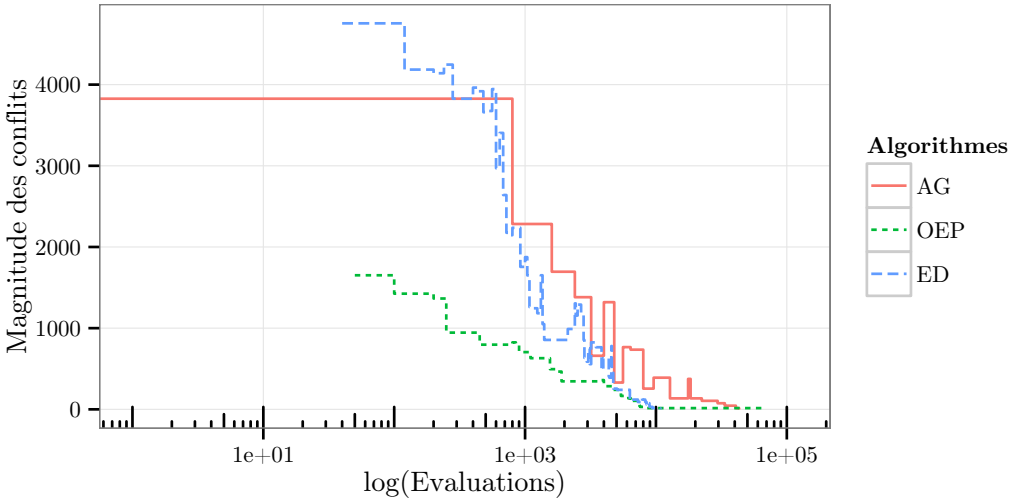


(b) Essaim particulaire

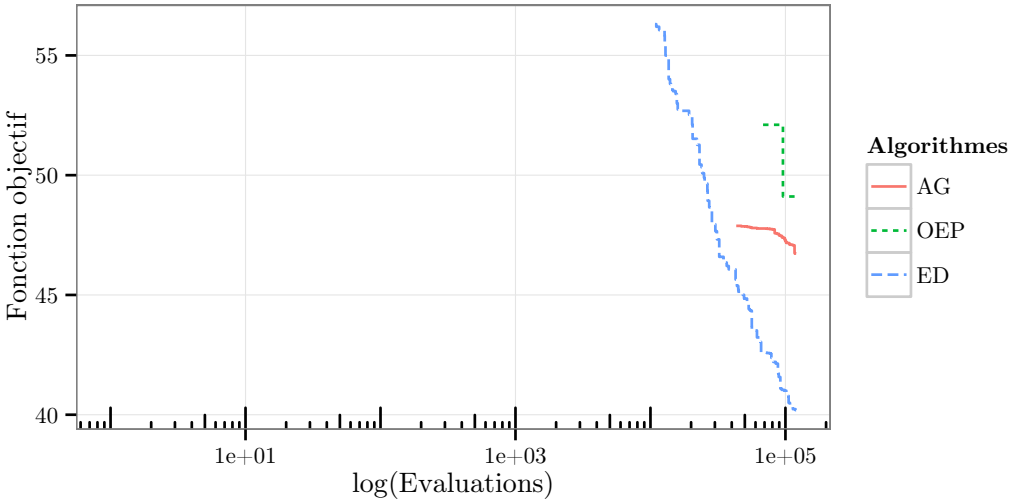


(c) Evolution différentielle

FIGURE 6.8 – Solutions du problème B_{20}

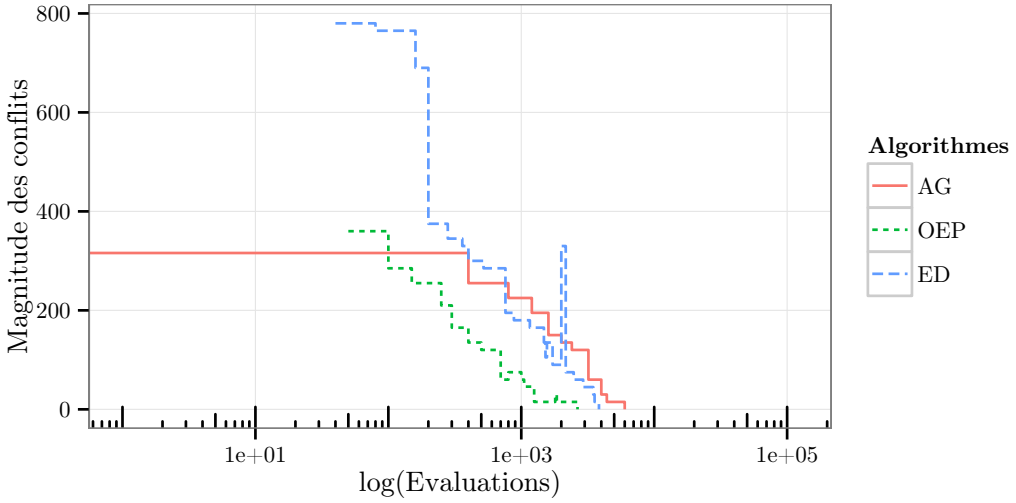


(a) Conflits restants

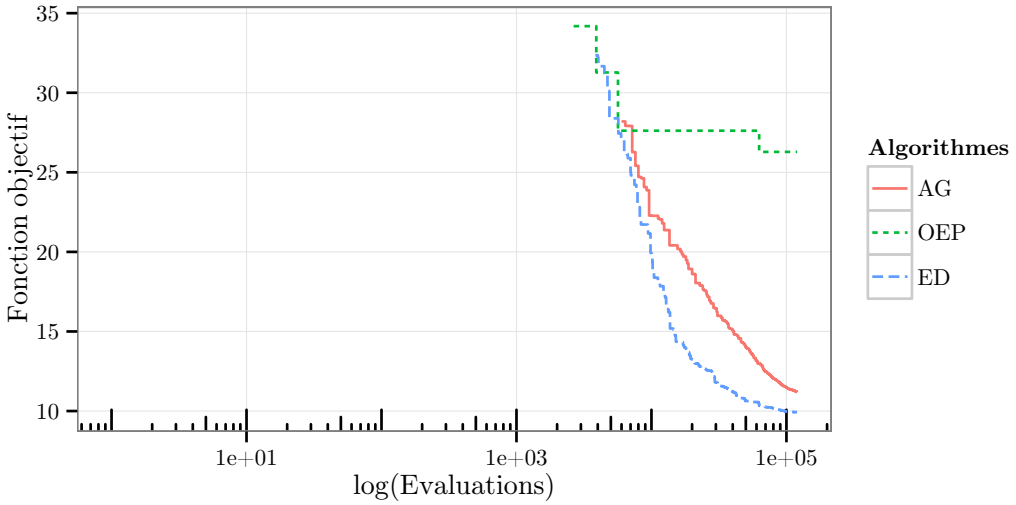


(b) Valeur objectif

FIGURE 6.9 – Problème C_{25}

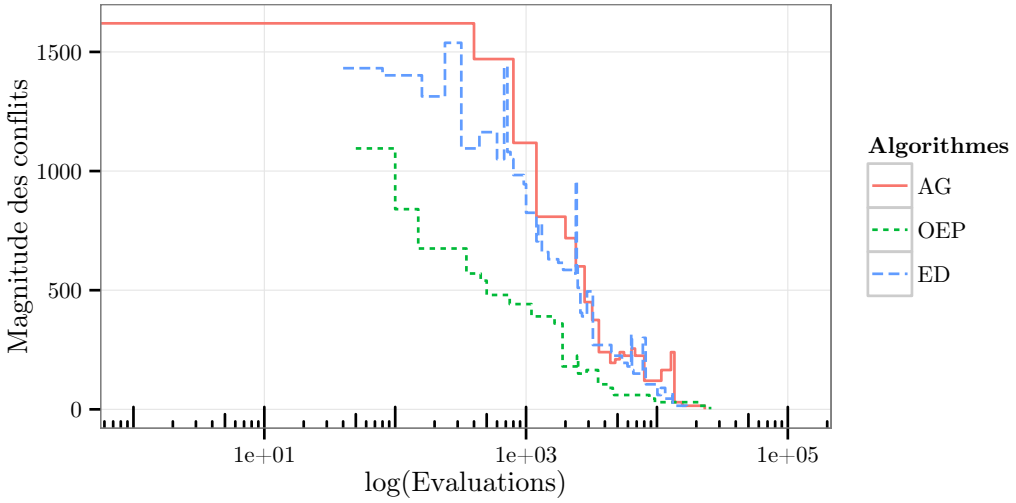


(a) Conflits restants

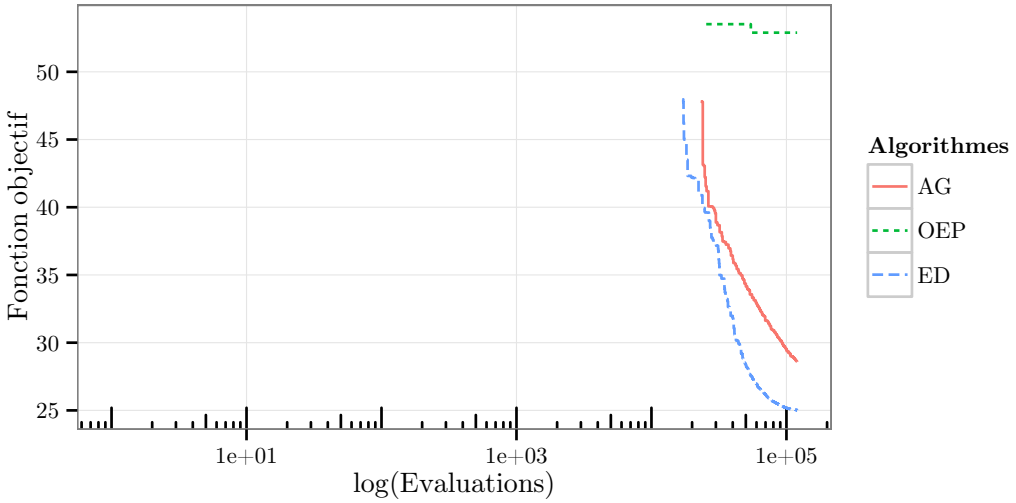


(b) Valeur objectif

FIGURE 6.10 – Problème B_{20}

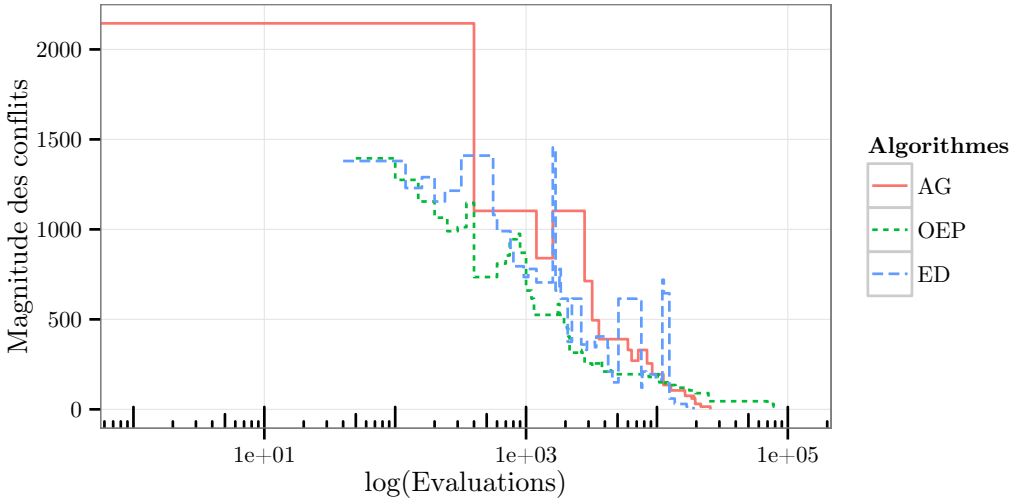


(a) Conflits restants

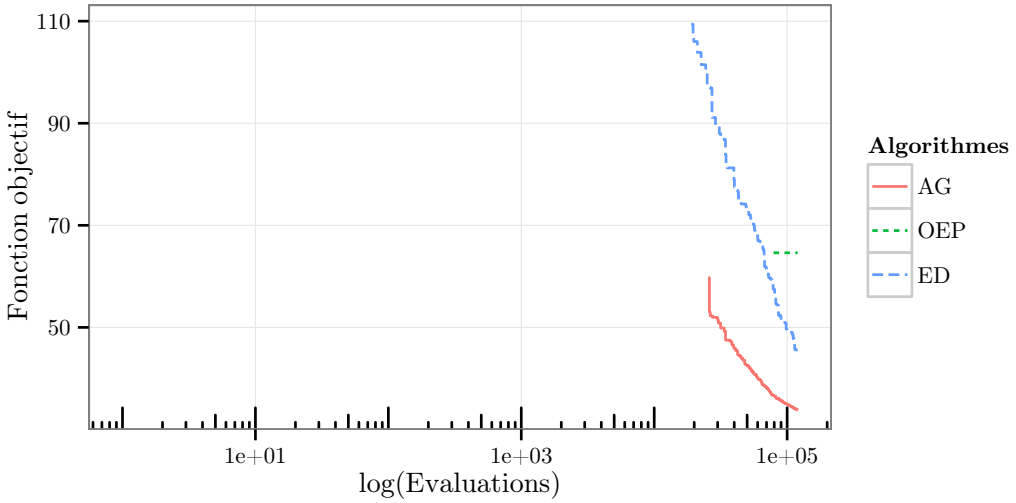


(b) Valeur objectif

FIGURE 6.11 – Problème B_{30}

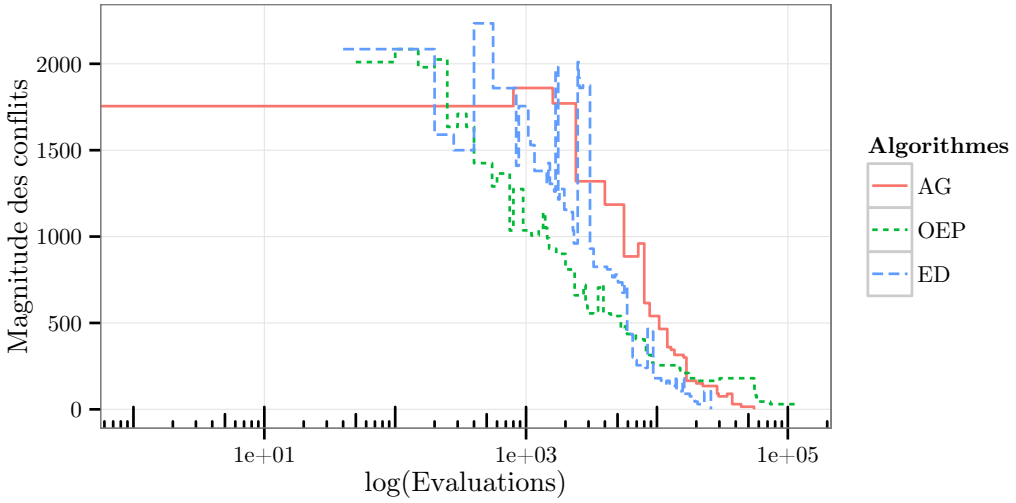


(a) Conflits restants

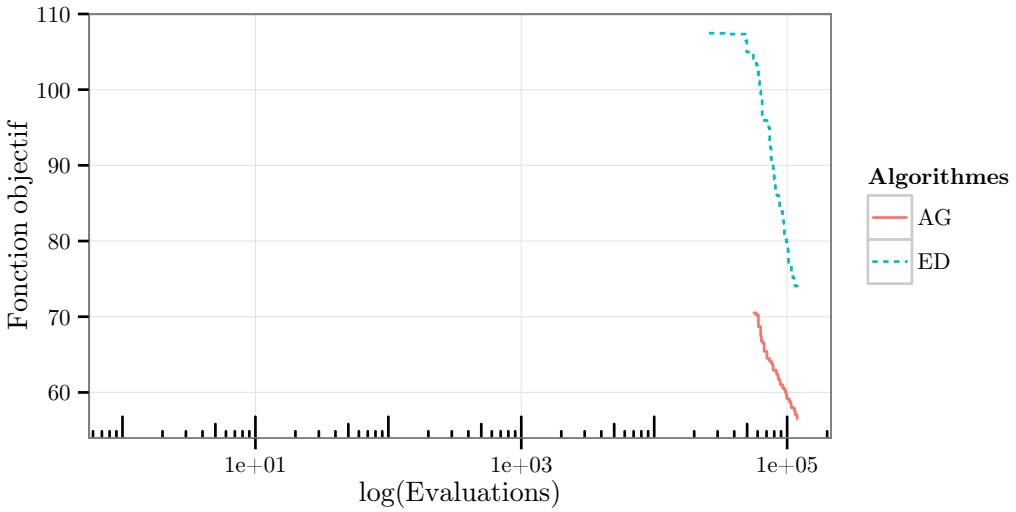


(b) Valeur objectif

FIGURE 6.12 – Problème B_{40}



(a) Conflits restants



(b) Valeur objectif

FIGURE 6.13 – Problème B_{50}

TABLE 6.1 – Durée moyenne des conflits et valeur objectif moyenne sur 20 exécutions, pour 120000 évaluations de la fonction objectif

		Non résolues	Durée des conflits restants (en s)				Valeur objectif des exécutions résolues			
	Méthode	(sur 20)	Min	Max	Moy	σ	Min	Max	Moy	σ
C_{25}	AG	0	-	-	-	-	46.7291	56.7154	49.5578	2.2153
	OEP	0	-	-	-	-	49.1103	55.1911	51.6155	1.3208
	ED	0	-	-	-	-	**40.1904**	54.7370	44.7435	3.9793
B_{20}	AG	0	-	-	-	-	11.1921	13.8793	12.4216	0.9086
	OEP	0	-	-	-	-	26.2831	30.2721	28.0791	1.2368
	ED	0	-	-	-	-	**9.9302**	11.8789	10.5836	0.5232
B_{30}	AG	5	15	30	21	8.2158	28.5927	36.6782	32.8227	2.1727
	OEP	6	15	15	15	0	52.8917	58.6254	55.6343	1.8290
	ED	**0**	-	-	-	-	**25.0167**	33.5501	29.6390	2.3471
B_{40}	AG	3	15	15	15	0	**33.7887**	46.0925	41.3232	3.1890
	OEP	12	15	105	37.5	28.2441	64.6237	76.2097	70.1448	3.6213
	ED	**0**	-	-	-	-	45.5938	67.4946	55.6706	6.2888
B_{50}	AG	3	30	75	45	25.9808	**56.4666**	72.5151	63.9583	3.4726
	OEP	20	30	180	129.75	41.9422	-	-	-	-
	ED	**0**	-	-	-	-	73.3092	115.5933	88.7457	13.0205

6.2.6 Conclusion et limites du modèle

La comparaison des trois algorithmes à population (AG, OEP, ED) sur le benchmark esquisse une vue d'ensemble des difficultés spécifiques au problème de résolution de conflits, modélisé par un problème d'optimisation sous contraintes de séparation. Il s'agit d'un problème combinatoire, très contraint dans de nombreuses (voire toutes les) dimensions de l'espace de recherche. En conséquence, les méthodes d'optimisation qui explorent l'espace de recherche en se déplaçant à l'aveugle sont plus susceptibles de violer plusieurs contraintes simultanément dans de nombreuses dimensions ; c'est le cas de l'OEP. Les méthodes qui opèrent des mouvements sélectifs dans chaque dimension, selon la violation de la contrainte et la valeur objectif de chaque vol, sont au contraire plus efficaces ; c'est le cas de l'ED et de l'AG doté d'un opérateur de croisement adapté, qui se sont avérés plus performants que l'OEP sur le jeu de tests considéré. Rappelons toutefois qu'il est parfois difficile d'ajuster les hyperparamètres des algorithmes stochastiques.

Ces résultats préliminaires indiquent que résoudre le problème de résolution de conflits dans un espace de recherche continu est une alternative viable aux méthodes existantes. La modélisation adoptée permet d'exprimer les manœuvres proposées comme une suite de points géographiques, directement utilisables par les FMS. Néanmoins, notre modèle n'est limité qu'aux manœuvres latérales et devrait être enrichi de manœuvres verticales et en vitesse. Les incertitudes sur les positions futures et sur les vitesses ne sont pas non plus considérées dans notre modèle simplifié.

6.3 Manœuvres latérales optimales

De nombreux domaines de l'ingénierie font intervenir un invariant dans leurs lois de commandes : « se déplacer à au moins un mètre du bord », « conserver un niveau de batterie supérieur à 50% », etc. Une alternative à la résolution approchée décrite dans la section précédente est de considérer la contrainte de séparation :

$$\forall t \in [0, t_f], \quad S^2 \leq \|\vec{p_i}(t) - \vec{p_j}(t)\|_2^2 \tag{6.4}$$

comme une inégalité universellement quantifiée (IUQ) (définition 27) et de résoudre le problème de résolution de conflits grâce à un panel de techniques d'intervalles spécifiques. Celles-ci sont décrites dans la suite de la section, et font intervenir un paramètre universellement quantifié (PUQ) p, correspondant ici au paramètre temporel t de l'équation 6.1.

Définition 27 (Inégalité universellement quantifiée) *Soit $c(x_1, \ldots, x_n, p)$ une fonction de n variables réelles $(x_1, \ldots, x_n) \in (X_1, \ldots, X_n)$ et d'un PUQ réel $p \in P$. On appelle IUQ la contrainte :*

$$\forall p \in P, \quad c(x_1, \ldots, x_n, p) \leq 0 \tag{6.5}$$

On note ρ_c la relation définie par l'IUQ c :

$$\rho_c := \{(x_1, \ldots, x_n) \in X_1 \times \ldots \times X_n \mid \forall p \in P, c(x_1, \ldots, x_n, p) \leq 0\} \tag{6.6}$$

Un test de réfutation permettant de prouver qu'une boîte $\boldsymbol{X}$ ne contient aucune solution de c, c'est-à-dire que $\boldsymbol{X} \cap \rho_c = \varnothing$, est donné par :

$$\boldsymbol{X} \cap \rho_c = \varnothing \quad \Leftrightarrow \quad \exists p \in P, \quad \exists (x_1, \ldots, x_n) \in \boldsymbol{X}, \quad c(x_1, \ldots, x_n, p) > 0 \tag{6.7}$$

Dans cette section, nous étendons le modèle précédemment proposé, basé sur des manœuvres rectilignes par morceaux, et proposons une résolution optimale des conflits certifiée par AI [Vanaret 14]. Les méthodes par intervalles écartent la possibilité de détecter les conflits entre deux trajectoires en calculant la distance minimale entre segments respectifs pendant un même intervalle de temps. L'approche utilisée ici est l'utilisation d'une unique expression analytique caractérisant la position d'un avion sur sa trajectoire à tout instant.

6.3.1 Expression de la position

Chaque manœuvre d'évitement est paramétrée par un triplet (d, α, l), et suit une trajectoire constituée d'un segment rectiligne écartant l'avion de sa trajectoire initiale, suivi d'un segment de retour (voir figure 6.3). On note (voir figure 6.14) :

- $\vec{u_1}$ le vecteur directeur unitaire sur la trajectoire initiale ;
- $\vec{u_2}$ le vecteur directeur unitaire sur le segment de déviation ;
- $\vec{u_3}$ le vecteur directeur unitaire sur le segment de retour ;
- $T_1 = \frac{d}{v}$ l'instant de début de manœuvre ;
- $T_2 = T_1 + \frac{l}{v}$ l'instant de retour ;
- $T_3 = T_2 + \frac{l}{v}$ l'instant auquel l'avion récupère sa trajectoire initiale.

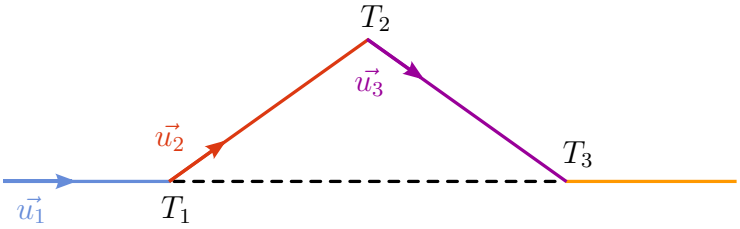


FIGURE 6.14 – Vecteurs directeurs de la manœuvre

Posons v la vitesse constante de l'avion. Le vecteur vitesse $\vec{v}(t)$ sur la trajectoire est égal à $\vec{v_1} = v\vec{u_1}$ sur l'intervalle de temps $[0, T_1]$, à $\vec{v_2} = v\vec{u_2}$ sur $[T_1, T_2]$, à $\vec{v_3} = v\vec{u_3}$ sur $[T_2, T_3]$ et $\vec{v_1}$ sur $[T_3, t_f]$. On peut donc exprimer le vecteur vitesse à tout instant t comme une combinaison convexe des trois vitesses $\vec{v_1}$, $\vec{v_2}$ et $\vec{v_3}$:

$$\begin{aligned}
\vec{v}(t) &= (1 - H(t - T_1) + H(t - T_3))\,\vec{v_1} + (H(t - T_1) - H(t - T_2))\,\vec{v_2} \\
&\quad + (H(t - T_2) - H(t - T_3))\vec{v_3} \\
&= \vec{v_1} + (\vec{v_2} - \vec{v_1})H(t - T_1) + (\vec{v_3} - \vec{v_2})H(t - T_2) + (\vec{v_1} - \vec{v_3})H(t - T_3) \\
&= v\,(\vec{u_1} + (\vec{u_2} - \vec{u_1})H(t - T_1) + (\vec{u_3} - \vec{u_2})H(t - T_2) + (\vec{u_1} - \vec{u_3})H(t - T_3))
\end{aligned} \tag{6.8}$$

où $H : \mathbb{R} \to [0,1]$ est la fonction de Heaviside :

$$\forall x \in \mathbb{R}, \quad H(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0 \end{cases} \tag{6.9}$$

En notant $\vec{p_0}$ la position initiale de l'avion, on obtient la position à l'instant t en intégrant la vitesse entre 0 et t :

$$\begin{aligned} \vec{p}(t) &= \int_0^t \vec{v}(u)du \\ &= \vec{p_0} + v\left(\vec{u_1}t + (\vec{u_2} - \vec{u_1})\max(0, t - T_1) + (\vec{u_3} - \vec{u_2})\max(0, t - T_2)\right. \\ &\quad \left. +(\vec{u_1} - \vec{u_3})\max(0, t - T_3)\right) \end{aligned} \tag{6.10}$$

En notant $\vec{d} = \begin{pmatrix} d_x \\ d_y \end{pmatrix}$ le vecteur directeur de la trajectoire initiale (soit $\vec{d} = \vec{u_1}$) et

$$\beta = \begin{cases} \arccos(d_x) & \text{si } d_y \geq 0 \\ -\arccos(d_x) & \text{sinon} \end{cases} \tag{6.11}$$

on obtient les vecteurs $\vec{u_3} - \vec{u_2}$, $\vec{u_2}$ et $\vec{u_3}$ par un raisonnement trigonométrique :

$$\vec{u_3} - \vec{u_2} = \begin{pmatrix} 2\sin(\alpha)d_y \\ -2\sin(\alpha)d_x \end{pmatrix} \qquad \vec{u_2} = \begin{pmatrix} \cos(\alpha + \beta) \\ \sin(\alpha + \beta) \end{pmatrix} \qquad \vec{u_3} = \begin{pmatrix} \cos(\alpha - \beta) \\ \sin(\beta - \alpha) \end{pmatrix} \tag{6.12}$$

6.3.2 Contracteur implémenté

Dans cette section, nous détaillons les différentes techniques d'intervalles permettant de contracter une boîte par rapport à une IUQ. Ces techniques incluent notamment la réduction du domaine du PUQ et la contraction de la boîte après instanciation du PUQ. L'algorithme 27 détaille l'implémentation du contracteur principal de Charibde spécifique aux IUQ. Les briques qui le composent sont détaillées dans les paragraphes suivants.

Algorithme 27 Contraction par rapport à une inégalité universellement quantifiée

fonction ContracterIUQ(**in-out** $\boldsymbol{X}$: boîte, g : IUQ, **in-out** P : domaine du PUQ p)
  si P dégénéré **alors**
    Contracter2B($\boldsymbol{X}, g, P$) ▷ algorithme 28
  sinon
    si g monotone par rapport à p sur P **alors**
      InstancierParamètre($\boldsymbol{X}, g, P$) ▷ contrainte équivalente
      Contracter2B($\boldsymbol{X}, g, P$) ▷ algorithme 28
    sinon
      InstancierParamètre($\boldsymbol{X}, g, P$) ▷ algorithme 29
      Contracter2B($\boldsymbol{X}, g, P$) ▷ algorithme 28
      ContracterNégation($\boldsymbol{X}, g, P$) ▷ algorithme 30
      RognerParamètre($\boldsymbol{X}, g, P$) ▷ algorithme 31
    fin si
  fin si
fin fonction

Contraction 2B

L'invocation d'un opérateur de cohérence partielle (par exemple 2B) sur la boîte $\boldsymbol{X}$ peut contracter les variables du problème, ainsi que le PUQ p. Une contraction sur P indique que certaines valeurs du domaine sont incohérentes avec l'IUQ, auquel cas la boîte $\boldsymbol{X}$ est éliminée (algorithme 28).

Algorithme 28 Contraction 2B par rapport à une inégalité universellement quantifiée

fonction Contracter2B(**in-out** $\boldsymbol{X}$: boîte, g : IUQ, P : domaine du PUQ)
  $P' \leftarrow P$
  $G_{\boldsymbol{X},P'} \leftarrow$ HC4Revise($G(\boldsymbol{X}, P') \leq 0$)
  si $\boldsymbol{X} = \varnothing$ ou $P' = \varnothing$ ou $w(P') < w(P)$ **alors**
    contrainte violée
  sinon si $\overline{G_{\boldsymbol{X},P'}} \leq 0$ **alors**
    contrainte satisfaite
  fin si
fin fonction

Instanciation du paramètre à une valeur particulière

L'IUQ g doit être vérifiée quelle que soit la valeur du PUQ dans son domaine. Contracter g en ayant au préalable instancié p à une valeur particulière $\hat{p}$ de son domaine offre un

pouvoir filtrant élevé et peut permettre d'éliminer la boîte si la contrainte est violée pour cette valeur particulière (voir équation 6.7).

Lorsque l'IUQ g est monotone par rapport au PUQ p sur $(\boldsymbol{X}, P)$, la valeur optimale de $\hat{p}$ est une des bornes de P ; il y a alors équivalence des contraintes [Goldsztejn 09]. Lorsque g est croissante (respectivement décroissante) par rapport à p, la contrainte $G(\boldsymbol{X}, P) \leq 0$ est équivalente à la contrainte $G(\boldsymbol{X}, \overline{P}) \leq 0$ (respectivement $G(\boldsymbol{X}, \underline{P}) \leq 0$).

Dans le cas contraire, on peut choisir arbitrairement le milieu de l'intervalle : $\hat{p} = m(P)$ (algorithme 29).

Algorithme 29 Instanciation du paramètre universellement quantifié

fonction InstancierParamètre(**in-out** $\boldsymbol{X}$: boîte, g : IUQ, P : domaine du PUQ)
  si g monotone par rapport à p **alors**
    $\hat{p} \leftarrow$ borne de P
  sinon
    $\hat{p} \leftarrow m(P)$
  fin si
  HC4Revise($G(\boldsymbol{X}, \hat{p}) \leq 0$)
  si $\boldsymbol{X} = \varnothing$ **alors**
    contrainte violée
  fin si
fin fonction

Contraction du paramètre

Lorsque la boîte courante $\boldsymbol{X}$ n'est pas réfutable par les tests précédents, contracter $\boldsymbol{X}$ par rapport à la *négation de la contrainte* permet de réduire le domaine du PUQ en éliminant des valeurs qui satisfont l'IUQ initiale [Benhamou 00, Goldsztejn 09].

Algorithme 30 Contraction par rapport à la négation d'une inégalité universellement quantifiée

fonction ContracterNégation($\boldsymbol{X}$: boîte, g : IUQ, **in-out** P : domaine du PUQ)
  $\boldsymbol{X'} \leftarrow \boldsymbol{X}$
  $P' \leftarrow P$
  HC4Revise($G(\boldsymbol{X'}, P') > 0$) ▷ contraction de $\boldsymbol{X'}$ et P'
  si $\boldsymbol{X'} = \varnothing$ **alors**
    g est satisfaite sur $(\boldsymbol{X}, \boldsymbol{P})$
  sinon
    $P \leftarrow P'$
  fin si
fin fonction

Rognage du paramètre

La technique de rognage ayant inspiré les opérateurs de cohérence kB et CID peut être exploitée pour réduire le domaine du PUQ [Collavizza 99b]. Evaluer la contrainte sur une bandelette P_i de P réduit la surestimation due à la dépendance et améliore l'encadrement de l'image de la contrainte (algorithme 31). Si l'évaluation de la contrainte est positive, alors l'IUQ est violée (voir équation 6.7). Si l'évaluation de la contrainte est négative, alors il n'est plus nécessaire de vérifier la contrainte sur cette bandelette et le domaine P de p est réduit à $P \setminus P_i$.

Algorithme 31 Rognage du paramètre universellement quantifié

fonction RognerParamètre(**in-out** $\boldsymbol{X}$: boîte, g : IUQ, **in-out** P : domaine du PUQ, s : nombre de bandelettes)
  $P' \leftarrow \varnothing$
  pour $i = \{1, \ldots, s\}$ **faire**
    $P_i \leftarrow$ Bandelette(P, i, s)
    si g monotone par rapport à p sur P_i **alors**
      remplacer P_i par une de ses bornes ▷ contrainte équivalente
    sinon
      InstancierParamètre$(\boldsymbol{X}, g, P_i)$ ▷ algorithme 29
    fin si
    Contracter2B$(\boldsymbol{X}, g, P_i)$ ▷ algorithme 28
    si $\boldsymbol{X} \neq \varnothing$ et $P_i \neq \varnothing$ **alors**
      $P' \leftarrow \square(P', P_i)$
    fin si
  fin pour
  $P \leftarrow P'$
fin fonction

La figure 6.15 illustre l'encadrement par AI d'une IUQ en fonction de la valeur du paramètre p. Le domaine P est découpé en s bandelettes. L'évaluation de la contrainte sur la première bandelette P_1 est négative, P est donc réduit à $P \setminus P_1$. La dernière bandelette P_s est un intervalle indécidable car l'évaluation de la contrainte contient 0. En revanche, l'évaluation sur la bandelette P_k est strictement positive, ce qui prouve que l'IUQ n'est pas satisfaite pour toutes les valeurs du paramètre. La contrainte est donc violée sur la boîte $(\boldsymbol{X}, P)$.

Bissection du paramètre

[Benhamou 00, Goldsztejn 09] proposent de bissecter le PUQ lorsque l'évaluation de la contrainte par AI est suffisamment améliorée sur une partition du domaine P. La contrainte :

$$\forall p \in P, \quad c(\boldsymbol{x}, p) \leq 0 \tag{6.13}$$

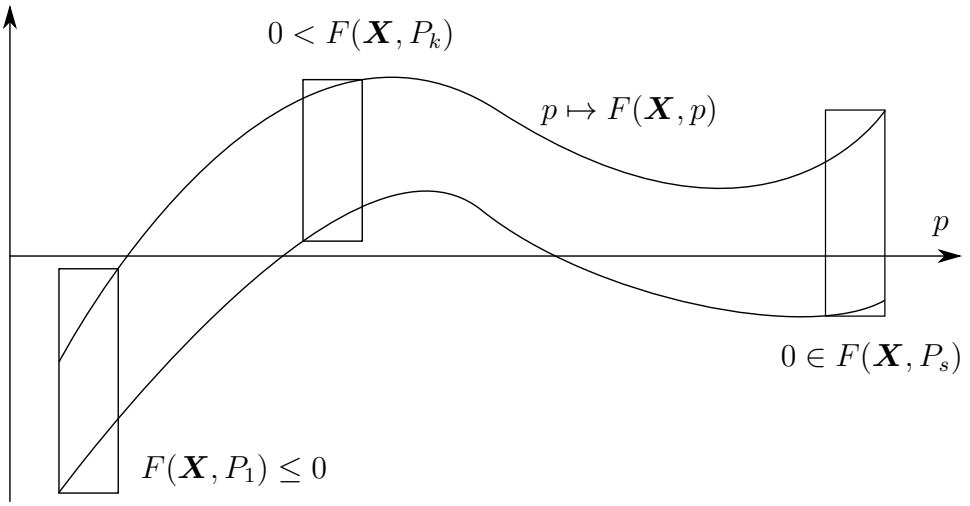


FIGURE 6.15 – Rognage du paramètre universellement quantifié

est alors séparée en deux contraintes distinctes :

$$\forall p_1 \in P_1, \quad c(\boldsymbol{x}, p_1) \leq 0 \qquad \wedge \qquad \forall p_2 \in P_2, \quad c(\boldsymbol{x}, p_2) \leq 0 \tag{6.14}$$

où $\{P_1, P_2\}$ est une partition du domaine P. Le critère de bissection est donné par :

$$w(\square(C(\boldsymbol{X}, P_1), C(\boldsymbol{X}, P_2))) < \kappa w(C(\boldsymbol{X}, P)) \tag{6.15}$$

où κ est un seuil choisi par l'utilisateur.

6.3.3 Problèmes de test

Les problèmes de test considérés sont trois instances à deux avions et trois instances à trois avions, initialement en conflit. Les configurations à deux avions incluent :

- le face à face (figure 6.16) : les deux avions sont placés l'un en face de l'autre, et volent l'un vers l'autre à la même vitesse (400 kts) ;
- le dépassement (figure 6.17) : les deux avions se suivent à distance sur la même trajectoire, l'avion de tête (volant à 400 kts) étant rattrapé par l'autre appareil, plus rapide (480 kts) ;
- le croisement (figure 6.18) : les trajectoires intiales des avions se croisent perpendiculairement en début de trajectoires. Les avions volent à la même vitesse (400 kts).

Afin de réduire les symétries des trois problèmes, une contrainte de signe est ajoutée sur l'angle de manœuvre d'un des deux avions.

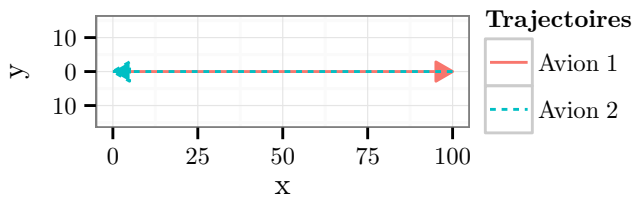


FIGURE 6.16 – Configuration initiale en face à face

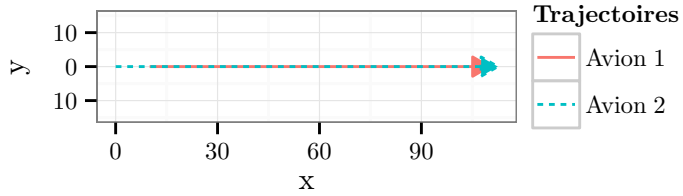


FIGURE 6.17 – Configuration initiale en dépassement

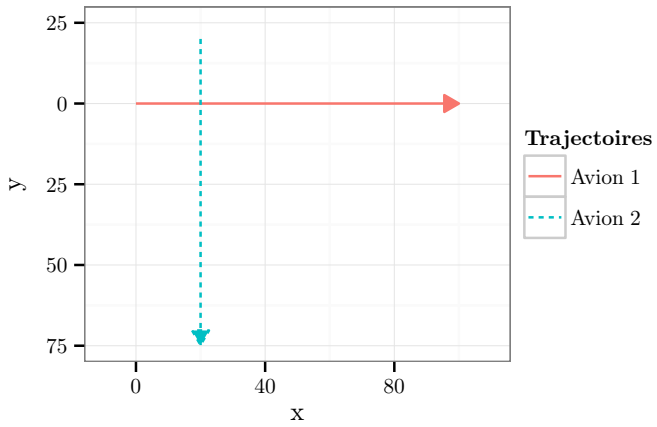


FIGURE 6.18 – Configuration initiale en croisement

Les configurations à trois avions incluent :

- la configuration symétrique (figure 6.19) : deux avions volent en face à face, un troisième les croise perpendiculairement. Les avions volent à la même vitesse (400 kts) ;
- la configuration bruitée (figure 6.20) : les avions sont placés comme dans la configuration symétrique, et leurs caps sont légèrement bruités. Les avions volent à la même vitesse (400 kts) ;
- le rond-point (figure 6.21) : les trois avions sont disposés régulièrement sur un cercle et convergent vers son centre à la même vitesse (400 kts).

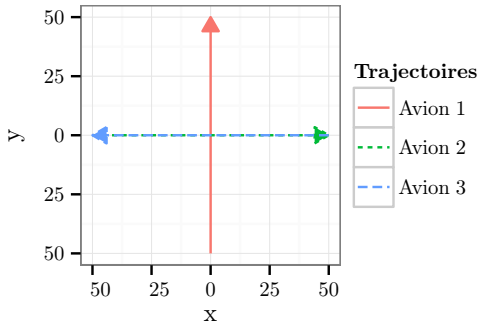


FIGURE 6.19 – Configuration initiale symétrique à trois avions

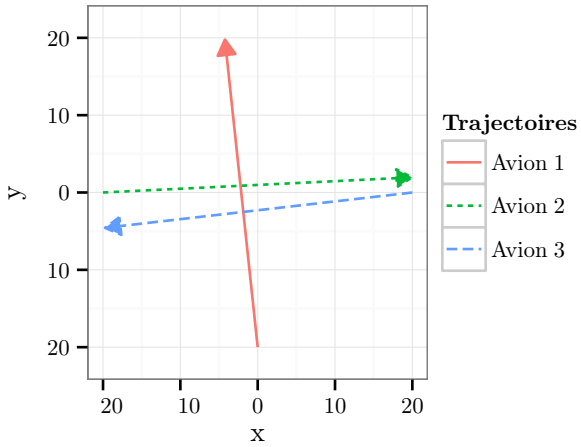


FIGURE 6.20 – Configuration initiale bruitée à trois avions

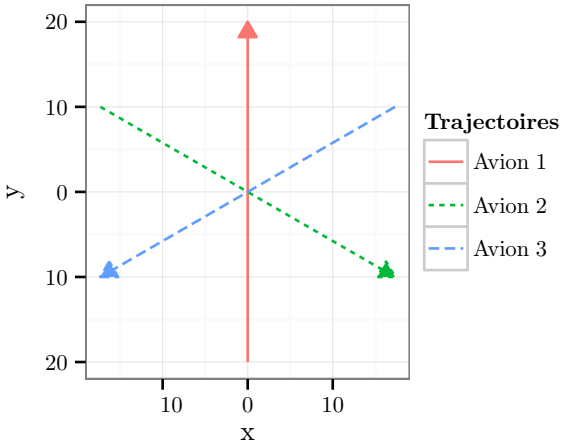


FIGURE 6.21 – Configuration initiale en rond-point à trois avions

6.3.4 Résultats expérimentaux

Nous présentons dans cette section les résultats optimaux certifiés par Charibde, pour une norme de séparation de $S = 5NM$. Le contracteur spécifique aux IUQ invoque les techniques présentées dans la sous-section 6.3.2, en optant soit pour une stratégie de rognage, soit pour une stratégie de bissection du domaine. Le contracteur 3BCID(20), hybridant les algorithmes de rognage 3B et CID, est utilisé pour l'intégralité des résultats expérimentaux.

Conflits à deux avions

Les trois instances à deux avions (6 variables) sont résolues avec les hyperparamètres du tableau 6.2. Les variables de décision du problème ont pour domaine $(d_i, \alpha_i, l_i) \in ([0, 100], [-\frac{\pi}{6}, \frac{\pi}{6}], [0, 100])$. Les résultats de Charibde sont donnés dans le tableau 6.3. Les solutions optimales obtenues sont présentées respectivement dans les figures 6.22 (face à face), 6.23 (croisement) et 6.24 (dépassement).

Table 6.2 – Conflits à deux avions : hyperparamètres de Charibde

Hyperparamètre	Valeur
ε (précision)	10^{-3}
NP (taille de la population)	30
W (facteur d'amplitude)	0.7
CR (taux de croisement)	0.9
Bissection	Largest
Priorité	MaxDist
η (ratio de point fixe)	0.8
Stratégie IUQ	Rognage

Table 6.3 – Conflits à deux avions : résultats de Charibde

	Face à face	Croisement	Dépassement
$\tilde{f}$	0.8420	0.9466	1.1762
Temps CPU (s)	1.5	0.6	2.5
Bissections	28	9	39
Taille maximale de $\mathcal{L}$	7	5	7
Nombre d'évaluations de F (BCI)	141976	64083	316833
Nombre d'évaluations de ∇F (BCI)	74	28	102
Nombre d'évaluations de f (ED)	9057	4517	6027
Nombre d'évaluations de F (ED)	71	30	49

Dans les manœuvres optimales obtenues par Charibde, un seul des deux avions est dévié. Ceci résulte de l'utilisation d'une somme dans la fonction objectif. Un coût basé sur le maximum des déviations individuelles fournirait en revanche une solution plus « équitable », dans laquelle tous les avions sont manœuvrés, avec des déviations plus faibles.

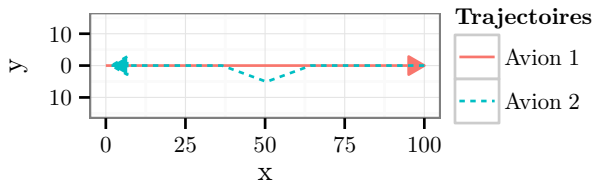


FIGURE 6.22 – Solution optimale à deux avions en face à face

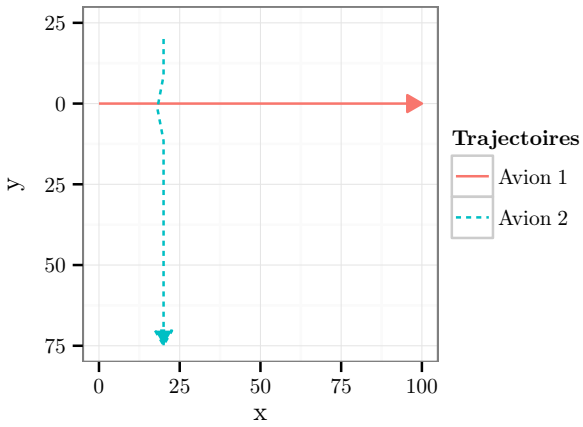


FIGURE 6.23 – Solution optimale à deux avions en croisement

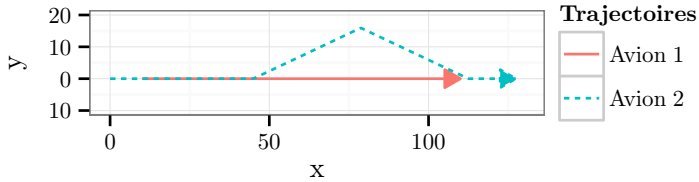


FIGURE 6.24 – Solution optimale à deux avions en dépassement

Conflits à trois avions

Les trois instances à trois avions (9 variables) sont résolues avec les hyperparamètres du tableau 6.4. Les variables de décision du problème ont pour domaine $(d_i, \alpha_i, l_i) \in ([0, 40], [-\frac{\pi}{6}, \frac{\pi}{6}], [0, 40])$. Les résultats de Charibde sont donnés dans le tableau 6.5.

Différentes stratégies de réduction du domaine du PUQ ont été utilisées : les résultats pour les configurations symétrique et bruitée ont été obtenus avec un opérateur basé sur la bissection du paramètre, et les résultats pour la configuration en rond-point ont été obtenus en utilisant un contracteur basé sur le rognage. Les solutions optimales obtenues sont présentées respectivement dans les figures 6.25 (configuration symétrique), 6.26 (configuration bruitée) et 6.27 (configuration en rond-point).

La configuration symétrique est résolue en 24 minutes. Les symétries du problème, non exploitées ici, ralentissent l'exploration exhaustive de l'espace de recherche. Lorsque les symétries sont brisées, la configuration bruitée est résolue en 4 minutes. La configuration en

rond-point apparaît comme un problème extrêmement difficile à résoudre, de nombreuses solutions équivalentes devant être éliminées : la solution optimale est obtenue après 11.5 heures. Ces résultats préliminaires, guère satisfaisants, reposent sur une modélisation dont la fonction objectif n'est pas suffisamment discriminante.

TABLE 6.4 – Conflits à trois avions : hyperparamètres de Charibde

Hyperparamètre	Symétrique	Bruitée	Rond-point
ε (précision)	10^{-1}	10^{-1}	10^{-1}
NP (taille de la population)	30	50	30
W (facteur d'amplitude)	0.7	0.7	0.7
CR (taux de croisement)	0.9	0.9	0.9
Bissection	Largest	Largest	Largest
Priorité	MaxDist	MaxDist	MaxDist
η (ratio de point fixe)	0.8	0.8	0.8
Stratégie IUQ	Bissection ($\kappa = 0.8$)	Bissection ($\kappa = 0.8$)	Rognage

TABLE 6.5 – Conflits à trois avions : résultats de Charibde

	Symétrique	Bruitée	Rond-point
$\tilde{f}$	1.8453	2.0420	3.2335
Temps CPU (s)	1432	242	41340
Bissections	31473	1330	129743
Taille maximale de $\mathcal{L}$	46	13	19
Nombre d'évaluations de F (BCI)	132163387	15347706	2425978874
Nombre d'évaluations de ∇F (BCI)	74080	2969	287221
Nombre d'évaluations de f (ED)	42205811	811090	36964505
Nombre d'évaluations de F (ED)	399	286	6057

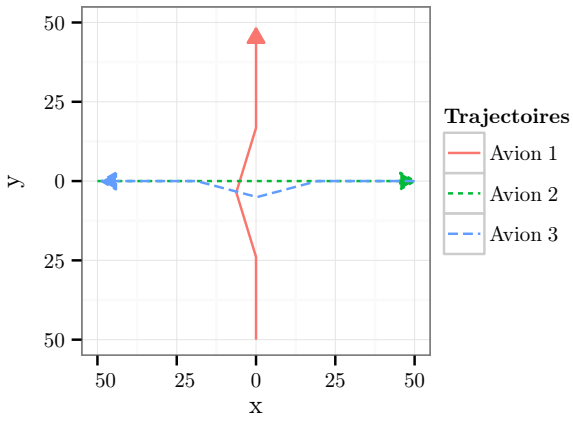


FIGURE 6.25 – Solution optimale à trois avions en configuration symétrique

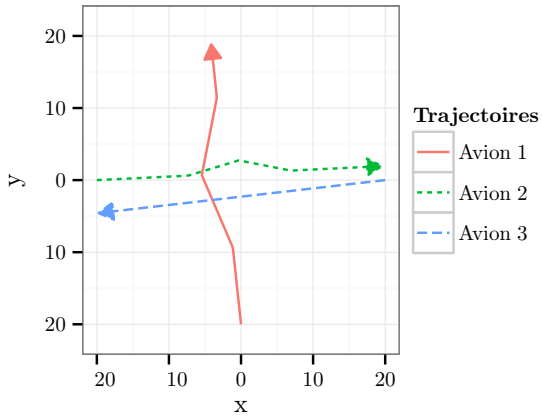


FIGURE 6.26 – Solution optimale à trois avions en configuration bruitée

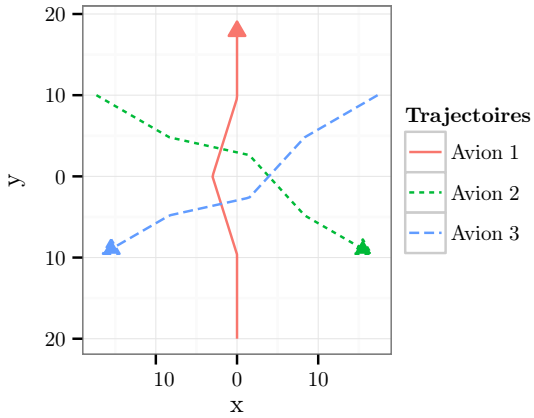


FIGURE 6.27 – Solution optimale à trois avions en configuration rond-point

6.4 Perspectives

La vitesse d'un appareil en vol est sujette à des incertitudes en raison du manque de fiabilité des mesures radar au sol. La connaissance de la position future de l'appareil se dégrade donc avec le temps, jusqu'à ce que la mesure radar suivante permette de rafraîchir l'incertitude sur la position. Par exemple, [Durand 96] résout les conflits en prenant en compte les incertitudes sur la vitesse : l'augmentation de la taille de l'espace de recherche est gérée par les AE.

Les techniques basées sur le calcul par intervalles sont intrinsèquement capables de résoudre des problèmes d'optimisation lorsque certains paramètres sont incertains. La vitesse incertaine d'un avion peut s'écrire sous la forme d'un intervalle :

$$v_{int} = [1 - p, 1 + p]v_{nom} \tag{6.16}$$

où v_{nom} est la vitesse nominale d'un avion en croisière (de l'ordre de 400 nœuds) et $p = 5\%$. Une perspective intéressante serait d'étendre notre modèle en intégrant la vitesse comme deuxième PUQ et de résoudre le problème sous contraintes d'IUQ avec les techniques de contraction décrites précédemment.

Chapitre 7

Preuve d'optimalité en dynamique moléculaire

Sommaire

7.1 Potentiel de Lennard-Jones 143
7.2 Optimisation de la configuration spatiale 144
7.3 Un problème ouvert 145
7.4 La première preuve d'optimalité pour cinq atomes 145
7.4.1 Réduction de la dépendance 145
7.4.2 Réduction des symétries 145
7.4.3 Preuve d'optimalité 146
7.4.4 Comparaison avec des solveurs de pointe 147

7.1 Potentiel de Lennard-Jones

Le potentiel de Lennard-Jones est un modèle simplifié proposé par [Jones 24] décrivant l'énergie d'interaction entre deux atomes sphériques. Il est considéré comme un modèle relativement précis pour les gaz rares au sein desquels les atomes se repoussent à faible distance et s'attirent à grande distance.

En notant d_{ij} la distance (en ångström) entre les centres des atomes i et j, le potentiel de Lennard-Jones est donné par :

$$V(d_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{d_{ij}} \right)^{12} - \left(\frac{\sigma}{d_{ij}} \right)^{6} \right] \tag{7.1}$$

où ϵ est appelé profondeur du puits de potentiel et σ est la distance pour laquelle le potentiel s'annule. La valeur remarquable $d_{min} := 2^{1/6}\sigma$ caractérise les atomes du gaz :

– lorsque $d_{ij} > d_{min}$, les forces attractives de Van der Waals, représentées par le terme $(\frac{\sigma}{d_{ij}})^6$, prennent le dessus sur les forces répulsives ;

– lorsque $d_{ij} < d_{min}$, les forces répulsives, représentées de manière approximative par le terme $(\frac{\sigma}{d_{ij}})^{12}$, l'emportent sur les forces attractives.

Généralement, le potentiel de Lennard-Jones s'exprime avec les valeurs réduites $\epsilon = 1eV$ et $\sigma = 1\text{Å}$ (figure 7.1) :

$$V(d_{ij}) = 4\left(\frac{1}{d_{ij}^{12}} - \frac{1}{d_{ij}^{6}}\right) \tag{7.2}$$

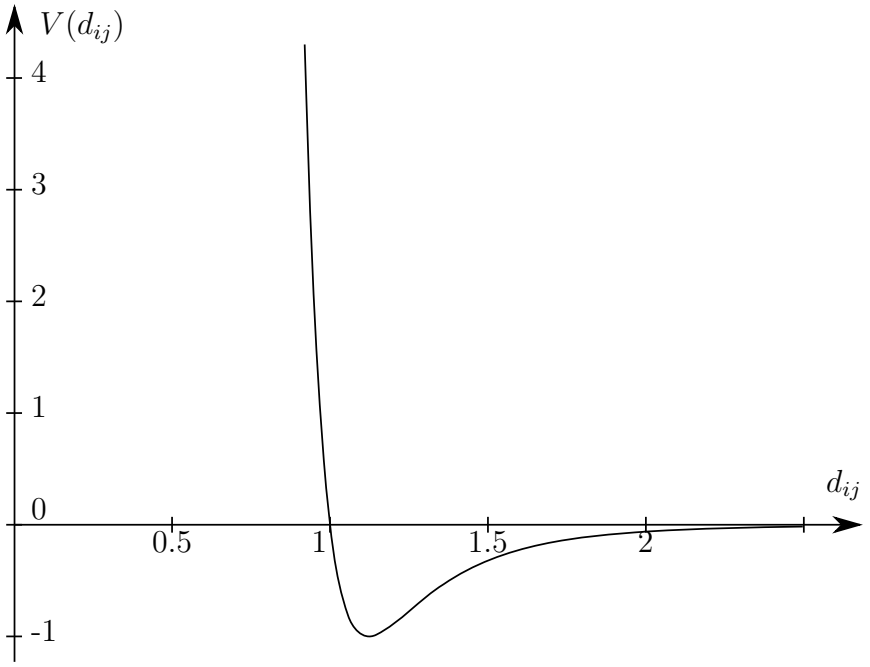


FIGURE 7.1 – Potentiel de Lennard-Jones

7.2 Optimisation de la configuration spatiale

Le problème d'optimisation sans contraintes associé au potentiel de Lennard-Jones consiste à minimiser les interactions entre paires d'atomes au sein d'une molécule (cluster) de gaz rare composée de N atomes. Chaque atome i du cluster est caractérisé par ses coordonnées cartésiennes (x_i, y_i, z_i) dans l'espace. On note $\boldsymbol{x} = (x_1, y_1, z_1, \ldots, x_N, y_N, z_N)$ l'ensemble des $3N$ variables du problème. Résoudre un cluster de Lennard-Jones à N atomes équivaut à trouver les coordonnées spatiales des atomes qui minimisent la fonction :

$$f_N(\boldsymbol{x}) = \sum_{i<j}^{N} V(d_{ij}) = 4\sum_{i<j}^{N}\left(\frac{1}{d_{ij}^{12}} - \frac{1}{d_{ij}^{6}}\right) \tag{7.3}$$

où la distance inter-atomique $d_{ij} > 0$ est donnée par :

$$d_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \tag{7.4}$$

7.3 Un problème ouvert

En dépit de son apparente simplicité, la fonction objectif f_N (équation 7.3) est non convexe et multimodale ; les tests numériques de [Locatelli 03] suggèrent que le nombre de minima locaux croît de manière exponentielle avec le nombre d'atomes N. Pour $N \leq 4$, les positions des atomes qui minimisent f_N correspondent aux sommets d'un tétraèdre régulier. En revanche, le problème de cluster de Lennard-Jones pour $N \geq 5$ demeure un problème ouvert qui n'a jamais été résolu par des méthodes globales fiables [Vavasis 94]. Nombre de configurations supposées optimales[1] ont été obtenues en utilisant un vaste panel de techniques de résolution approchées [Northby 87, Hoare 71, Leary 97, Wales 97]. La meilleure solution connue pour le cluster de Lennard-Jones à cinq atomes est une bipyramide à base triangulaire [Sloane 95], d'évaluation -9.103852415708.

7.4 La première preuve d'optimalité pour cinq atomes

Dans cette section, nous présentons la preuve d'optimalité obtenue par Charibde sur le problème ouvert à cinq atomes. Nous montrons que la meilleure solution connue, jamais numériquement certifiée auparavant, est optimale. Nous expliquons comment reformuler le problème pour réduire la surestimation de l'AI et comment réduire le nombre de variables et le domaine initial du problème. Enfin, nous présentons la première preuve numérique d'optimalité de la solution du problème ouvert à cinq atomes.

7.4.1 Réduction de la dépendance

L'équation 7.2 contient deux occurrences de la variable d_{ij}. Une méthode classique pour réduire à un le nombre d'occurrences est de compléter le carré :

$$V(d_{ij}) = 4\left(\frac{1}{d_{ij}^6} - \frac{1}{2}\right)^2 - 1 \tag{7.5}$$

La fonction d'inclusion naturelle du potentiel V est alors optimale par rapport aux occurrences de d_{ij}. Notons en revanche que l'expression 7.3 de la fonction objectif souffre du problème de dépendance puisque les coordonnées spatiales des atomes apparaissent plusieurs fois dans les termes de distances inter-atomiques d_{ij}.

7.4.2 Réduction des symétries

L'équation 7.3 ne faisant intervenir que des distances entre atomes, les solutions optimales du problème de cluster sont invariantes par translation et rotation. Il est donc nécessaire de fixer les coordonnées de certains atomes afin de réduire la taille de l'espace de recherche.

1. Une liste des meilleures solutions connues est disponible sur `http://doye.chem.ox.ac.uk/jon/structures/LJ.html`

Fixons le premier atome à l'origine du repère, le deuxième atome sur la demi-droite $x \geq 0$, le troisième sur le premier quadrant du plan $z = 0$ et le quatrième sur le premier octant :

$$\begin{cases} x_1 = y_1 = z_1 = 0 \\ x_2 \geq 0, y_2 = z_2 = 0 \\ x_3 \geq 0, y_3 \geq 0, z_3 = 0 \\ x_4 \geq 0, y_4 \geq 0, z_4 \geq 0 \end{cases} \tag{7.6}$$

La réduction de symétries permet de diminuer la taille du problème initial de 15 variables à 9 variables, et de réduire substantiellement les bornes du domaine restant.

7.4.3 Preuve d'optimalité

L'optimalité de la meilleure solution connue pour cinq atomes a été prouvée par Charibde avec une précision $\varepsilon = 10^{-9}$: $f_5^* = -9.103852415707552$. Les coordonnées spatiales correspondantes sont données dans le tableau 7.1, pour un domaine initial $(x_i, y_i, z_i) \in [-1.2, 1.2]$. La configuration spatiale optimale est représentée en figure 7.2. Les hyperparamètres de Charibde sont répertoriés dans le tableau 7.2.

TABLE 7.1 – Solution optimale de Lennard-Jones à cinq atomes

Atome	x	y	z
1	0	0	0
2	1.1240936	0	0
3	0.5620468	0.9734936	0
4	0.5620468	0.3244979	0.9129386
5	0.5620468	0.3244979	-0.9129385

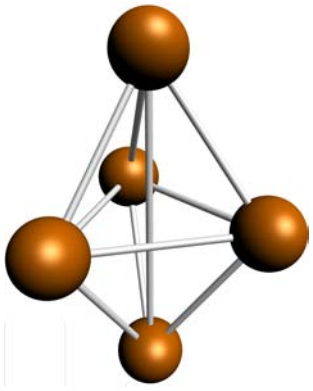


FIGURE 7.2 – Configuration optimale de Lennard-Jones à cinq atomes

Les temps de calcul et nombre d'évaluations de la fonction objectif et de son gradient sont donnés dans le tableau 7.3 en moyenne sur 100 exécutions de Charibde. L'ED atteint

TABLE 7.2 – Cluster de Lennard-Jones : hyperparamètres de Charibde

Hyperparamètre	Valeur
ε (précision)	10^{-9}
NP (taille de la population)	40
W (facteur d'amplitude)	0.7
CR (taux de croisement)	0.4
Bissection	Largest
Priorité	MaxDist
η (ratio de point fixe)	0

l'optimum global f_5^* après 764 itérations (en un temps de 0.11s), mais un total de 1436s est nécessaire à l'obtention de la preuve d'optimalité.

TABLE 7.3 – Cluster de Lennard-Jones : résultats moyens de Charibde sur 100 exécutions

Temps CPU moyen (s)	1436
Temps maximum (s)	1800
Taille maximale de $\mathcal{L}$	46
Nombre d'évaluations de F (BCI)	7088758
Nombre d'évaluations de ∇F (BCI)	78229737
Nombre d'évaluations de f (ED)	483642320
Nombre d'évaluations de F (ED)	132

7.4.4 Comparaison avec des solveurs de pointe

Une comparaison entre Charibde et les solveurs de programmation non-linéaire considérés comme les plus performants actuellement, BARON et Couenne, est fournie dans le tableau 7.4 sur le problème reformulé. Le temps de recherche indique le temps au bout duquel l'optimum est trouvé, et le temps total représente le temps nécessaire à la preuve d'optimalité. Ces résultats confirment que ces solveurs, malgré leurs très bons temps de convergence, ne garantissent pas l'optimalité de la solution en raison d'erreurs numériques (les décimales incorrectes sont soulignées). Ainsi, la solution trouvée par BARON est incorrecte à partir de la sixième décimale (pour une précision demandée de $\varepsilon = 10^{-9}$) et celle de Couenne l'est à partir de la cinquième décimale. En outre, si Couenne indique avoir trouvé un minimum global, BARON ne parvient pas à prouver l'optimalité de sa solution.

TABLE 7.4 – Cluster de Lennard-Jones : comparaison de BARON, Couenne et Charibde

	BARON	Couenne	Charibde
Minimum	-9.10385346444055	-9.103870325603582	-9.103852415707552
Temps recherche (s)	0.23	41.94	0.11
Temps total (s)	0.23	61.7	1436
Statut indiqué	localement optimal	optimal	certifié ($\varepsilon = 10^{-9}$)

Conclusion générale

Nous avons proposé dans ce document un nouveau cadre coopératif hybridant des algorithmes évolutionnaires et des méthodes d'intervalles alternant partitionnement de l'espace de recherche et filtrage des valeurs incohérentes. Notre étude s'est focalisée sur l'élaboration d'opérateurs visant à s'échapper des minima locaux dans l'algorithme évolutionnaire et à accélérer la convergence des méthodes d'intervalles.

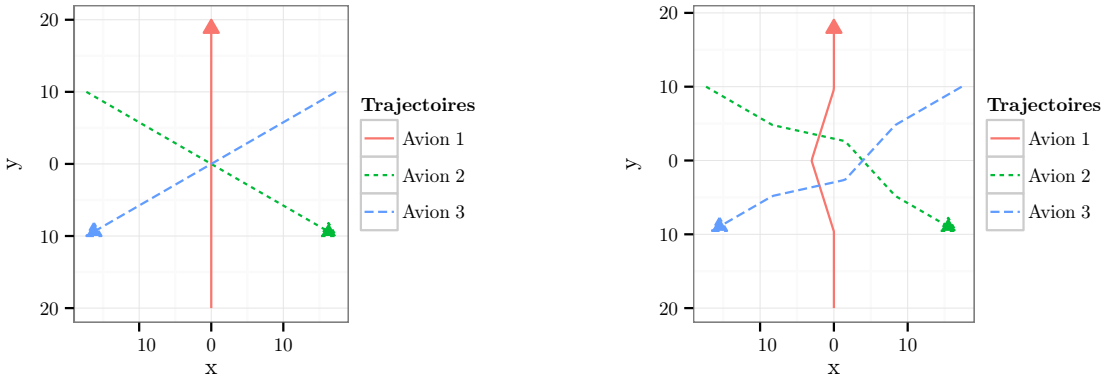
Contributions

Inspiré des travaux de [Alliot 12a], notre algorithme hybride Charibde combine un algorithme évolutionnaire et des méthodes d'intervalles, et certifie numériquement la solution avec une précision donnée. L'algorithme évolutionnaire explore rapidement l'espace de recherche pour obtenir une bonne solution réalisable, puis transmet son évaluation aux méthodes d'intervalles. Améliorer le meilleur majorant du minimum global intensifie les tests de réfutation visant à éliminer les sous-domaines sous-optimaux de l'espace de recherche. En retour, une solution qui améliore la meilleure évaluation courante, identifiée durant le partitionnement de l'espace de recherche, est intégrée à la population. La convergence prématurée vers des minima locaux est ainsi évitée, et la population guidée par le nouvel individu. L'espace de recherche restant à explorer, maintenu par la méthode d'intervalles sous la forme d'une liste de sous-domaines, est exploité par l'algorithme évolutionnaire via un opérateur de réduction de domaine ; la population est périodiquement réinitialisée à l'intérieur du nouveau domaine, purgé de sous-espaces sous-optimaux ou non réalisables. En combinant cet opérateur avec une stratégie particulière d'exploration de l'espace de recherche (basée sur la notion de plus grande distance entre solution courante et sous-domaine restant), nous avons montré que l'enveloppe convexe des sous-domaines restant à explorer converge rapidement vers le minimum global.

Charibde se montre compétitif par rapport aux solveurs de pointe : une comparaison avec les solveurs GlobSol, IBBA et Ibex sur des problèmes difficiles de la base COCONUT montre un gain en temps de calcul d'un ordre de grandeur. La stratégie de Charibde est davantage tournée vers le partitionnement de l'espace de recherche, tandis qu'Ibex mise sur des procédures de contraction au pouvoir filtrant élevé. De nouveaux résultats optimaux sont fournis pour cinq problèmes multimodaux (Michalewicz, Sine Wave Sine Envelope, Eggholder, Keane, Rana) ; peu de résultats – même approchés – sont disponibles dans la littérature. Nous suggérons que l'influence d'une bonne approximation du minimum global

fournie par l'algorithme évolutionnaire reste limitée sur les problèmes avec de multiples minima locaux.

Une comparaison de trois algorithmes à population sur un benchmark de problèmes de résolution de conflits aériens en phase de croisière a mis en évidence le comportement propre à chacune des méthodes. Il s'agit d'un problème très combinatoire et très contraint dans de nombreuses dimensions de l'espace de recherche. Résoudre le problème de résolution de conflits dans un espace de recherche continu apparaît comme une alternative viable aux méthodes existantes, basées principalement sur une discrétisation des variables. Notre modélisation en manœuvres latérales permet d'exprimer les déviations comme une suite de points géographiques directement utilisable par les FMS. Nous avons également proposé une nouvelle formulation pour la résolution de conflits dans laquelle les contraintes de séparation entre avions sont universellement quantifiées. Des techniques d'intervalles spécifiques aux contraintes universellement quantifiées permettent de résoudre de petites instances ; les tests préliminaires montrent que des problèmes à deux avions sont résolus en quelques secondes, tandis que les configurations à trois avions sont beaucoup plus difficiles à résoudre, en particulier lorsque la configuration initiale est symétrique. Nous avons établi que la solution optimale du rond-point est bien une configuration où tous les avions tournent du même côté (figure 7.3).



(a) Configuration initiale

(b) Solution optimale

FIGURE 7.3 – Rond-point à trois avions : configuration initiale et solution optimale certifiée par Charibde

La meilleure solution connue pour le cluster de Lennard-Jones à cinq atomes, un problème ouvert en dynamique moléculaire, est numériquement certifiée par Charibde. Nous montrons que les solveurs de programmation non-linéaire BARON et Couenne, bien que plus performants en temps de calcul, fournissent des solutions numériquement erronées.

Perspectives

Méthodes d'optimisation locale

Les méthodes d'optimisation locale n'ont pas été mentionnées dans ce document. Elles sont pourtant invoquées par la plupart des solveurs d'optimisation globale afin d'obtenir une bonne approximation du minimum global. Tous les problèmes d'optimisation abordés dans ce document possèdent une expression analytique connue, exploitable par des méthodes de descente (quasi-Newton, BFGS). Toutefois, la plupart des tests numériques a montré que l'algorithme à ED converge souvent vers le minimum global sur les problèmes considérés. Lorsque ce n'est pas la cas, les techniques avancées d'hybridation décrites dans le chapitre 5 – l'intégration à la population de nouvelles solutions et la réduction périodique du domaine – permettent généralement de s'échapper rapidement des minima locaux et de guider la population vers le minimum global.

Contraintes d'inégalité

La gestion des contraintes d'inégalité par les méthodes d'intervalles apparaît comme une piste de recherche prometteuse. Rappelons qu'en un minimiseur global :

1. les contraintes inactives n'ont aucune influence sur le minimum non contraint ;
2. les contraintes actives se comportent comme des contraintes d'égalité, au pouvoir filtrant plus élevé qu'une contrainte d'inégalité.

[Hansen 92] évoque la possibilité de construire le système KKT du problème sur la boîte courante, puis de le résoudre par un algorithme de Newton multivarié. Cette approche semble peu usitée par les solveurs modernes en raison de sa pénibilité (introduction de variables supplémentaires et résolution d'un système carré devant parfois être préconditionné). Elaborer une heuristique afin de distinguer les contraintes actives et inactives de manière disjonctive pour accélérer la contraction de la boîte courante constitue néanmoins une piste intéressante.

Méthodes d'encadrement

A l'instar de X-Newton [Araya 12], la variante récursive de [Hansen 68] est implémentée dans l'algorithme de convexification de Charibde. Bien qu'elle calcule un encadrement généralement plus précis que la forme de Taylor, les n dérivées partielles sont évaluées séparément sur des boîtes différentes.

Le calcul de pentes en mode adjoint (voir sous-section A.1.2) généralise la variante de Hansen, tout en évaluant les n pentes partielles simultanément. L'arithmétique affine (voir sous-section A.2) conserve les dépendances linéaires entre quantités et réduit généralement la dépendance de manière significative. Ces deux techniques, bien que complexes à implémenter, amélioreraient probablement la convexification dans Charibde et produiraient des minorants de meilleure qualité.

Parallélisation

Les sous-espaces disjoints traités par l'algorithme de BCI peuvent être explorés en parallèle et de manière indépendante. Il est donc possible de répartir l'exploration du domaine sur plusieurs processus esclaves, tout en maintenant à jour le meilleur majorant connu dans un processus maître. Il convient d'établir avec soin l'équilibrage des tâches entre processus, de sorte que tous aient une charge de travail équivalente.

Interfaçage avec AMPL

Les problèmes d'optimisation traités par Charibde sont décrits en langage OCaml ; ceci rend l'utilisation de notre solveur par les membres de la communauté d'optimisation globale peu aisée. L'interfaçage avec un langage de description mathématique standardisé, tel AMPL, semble indispensable à la distribution du solveur.

Annexe A

Calcul par intervalles

A.1 Arithmétique d'intervalles

Nous détaillons dans cette section l'expression analytique de la forme optimale de Baumann. Les formes de Taylor et de Baumann sont ensuite comparées à la forme centrée, qui fournit généralement une inclusion plus précise.

A.1.1 Forme de Baumann optimale

Soient X un intervalle, $f : \mathbb{R} \to \mathbb{R}$ une fonction continue dont on peut calculer un encadrement de l'image et de la dérivée sur X, et $c \in X$. La borne gauche de la forme de Taylor $F_T(X, c) := f(c) + F'(X)(X - c)$ est déterminée par le minimum entre les ordonnées des points A et B de la figure A.1.

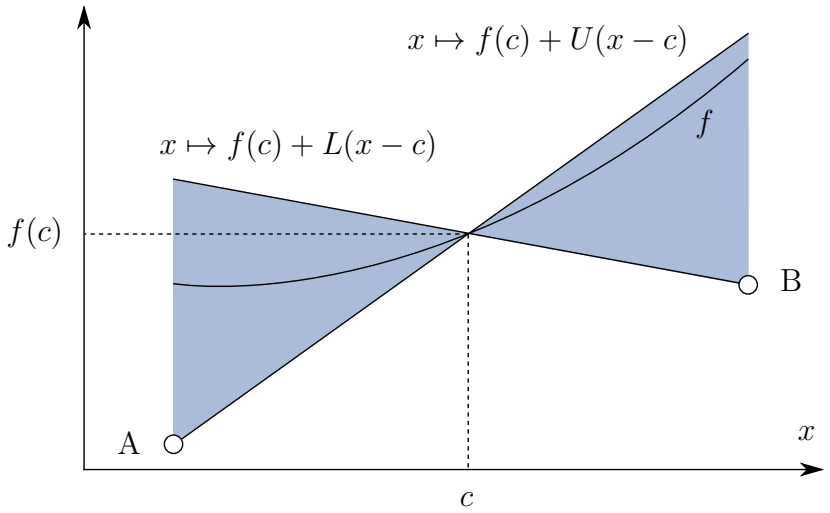


FIGURE A.1 – Forme de Taylor

En notant $[L, U] := F'(X)$, l'encadrement de $f(X)$ par la forme de Taylor s'écrit :

$$\begin{aligned} F_T(X, c) &= [\min(f(c) + U(\underline{X} - c), f(c) + L(\overline{X} - c)), \\ &\quad \max(f(c) + L(\underline{X} - c), f(c) + U(\overline{X} - c))] \\ &= f(c) + [\min(U(\underline{X} - c), L(\overline{X} - c)), \max(L(\underline{X} - c), U(\overline{X} - c))] \end{aligned} \tag{A.1}$$

Le centre optimal de [Baumann 88] c_B^- pour la borne inférieure (respectivement c_B^+ pour la borne supérieure) maximise la borne gauche (respectivement minimise la borne droite) de l'intervalle dans l'équation A.1. On distingue alors deux cas :

- lorsque f est monotone sur X ($F'(X)$ est de signe constant), la valeur optimale de c est une borne de X ;
- lorsque f n'est pas monotone sur X ($L < 0 < U$), l'expression $\min(U(\underline{X}-c), L(\overline{X}-c))$ est maximale pour c_B^- (respectivement l'expression $\max(L(\underline{X} - c), U(\overline{X} - c))$ est minimale pour c_B^+) lorsque ses opérandes sont égaux. On en déduit les équations :

$$U(\underline{X} - c_B^-) = L(\overline{X} - c_B^-) \tag{A.2}$$

$$L(\underline{X} - c_B^+) = U(\overline{X} - c_B^+) \tag{A.3}$$

  et les expressions analytiques données dans la sous-section 3.2.2.

La méthode des centres optimaux de Baumann s'étend aux fonctions multivariées en appliquant les formules composante par composante.

A.1.2 Formes centrées

Il est possible de remplacer les dérivées partielles par des pentes partielles (définition 28) dans la forme de Taylor, tout en conservant une inclusion rigoureuse de l'image de f [Krawczyk 85].

Définition 28 (Pente, fonction pente) *Soient $f : D \subset \mathbb{R} \to \mathbb{R}$, $x \in D$ et $z \in D$ avec $x \neq z$. On appelle* pente *le réel $s_f(x, z)$ défini par :*

$$s_f(x, z) := \frac{f(x) - f(z)}{x - z} \tag{A.4}$$

Lorsque f est continûment différentiable, $\lim_{x \to z} s_f(x, z) = f'(z)$, auquel cas nous définissons $s_f(z, z) := f'(z)$. On appelle fonction pente *la fonction qui, pour un z fixé, associe à tout x la valeur $s_f(x, z)$.*

Les pentes sont calculables analytiquement pour les fonctions rationnelles. Elles ont été étendues aux fonctions irrationnelles (exp, log) indépendamment par [Ratz 96], [Kolev 97] et [Rump 96], en exploitant des propriétés de convexité ou concavité. [Schnurr 08] présente un résultat plus général lorsque la fonction a un unique point d'inflexion et propose une expression pour les fonctions trigonométriques usuelles.

Par exemple, la pente de $f(x) = \sqrt{x}$ entre deux réels positifs x et z est donnée par :

$$\frac{f(x) - f(z)}{x - z} = \frac{\sqrt{x} - \sqrt{z}}{x - z} = \frac{\sqrt{x} - \sqrt{z}}{(\sqrt{x} - \sqrt{z})(\sqrt{x} + \sqrt{z})} = \frac{1}{\sqrt{x} + \sqrt{z}} \tag{A.5}$$

La forme centrée (définition 29) fournit une meilleure approximation que la forme de Taylor : $S_f(X, z)$ représente un encadrement du taux d'accroissement de f entre z fixé (et n'appartenant pas nécessairement à X) et tout point de X. La dérivée $F'(X)$ représente quant à elle un encadrement du taux d'accroissement de f entre toutes les paires de points de X. Par conséquent, la fonction pente calcule toujours un intervalle au moins aussi étroit que la dérivée.

Définition 29 (Forme centrée) *Soit $S_f(X, z)$ une inclusion de $s_f(X, z) := \{s_f(x, z) \mid x \in X\}$ sur un intervalle X, avec $z \in D$ fixé. Alors la* forme centrée *est définie par :*

$$F_C(X, z) := f(z) + S_f(X, z)(X - z) \tag{A.6}$$

Une comparaison des encadrements fournis par la forme de Taylor et la forme centrée peut être trouvée dans [Zuhe 90, Oliveira 96, Gay 12]. Une expression de la forme centrée d'ordre 2, basée sur la fonction pente d'ordre 2, est donnée dans [Gay 10].

Les inclusions des formes naturelle, de Taylor et centrée sont comparées dans l'exemple 16 pour la fonction $f(x) = x^2 - x$, et leurs ordres de convergence respectifs sont estimés.

Exemple 16 (Comparaison des formes naturelle, de Taylor et centrée) *Soient $X = [-2, 2]$, $f(x) = x^2 - x$, $x \in X$ et $z \in \mathbb{R}$. La fonction pente est donnée par :*

$$s_f(x, z) = \frac{f(x) - f(z)}{x - z} = \frac{x^2 - x - z^2 + z}{x - z} = \frac{(x - z)(x + z - 1)}{x - z} = x + z - 1 \tag{A.7}$$

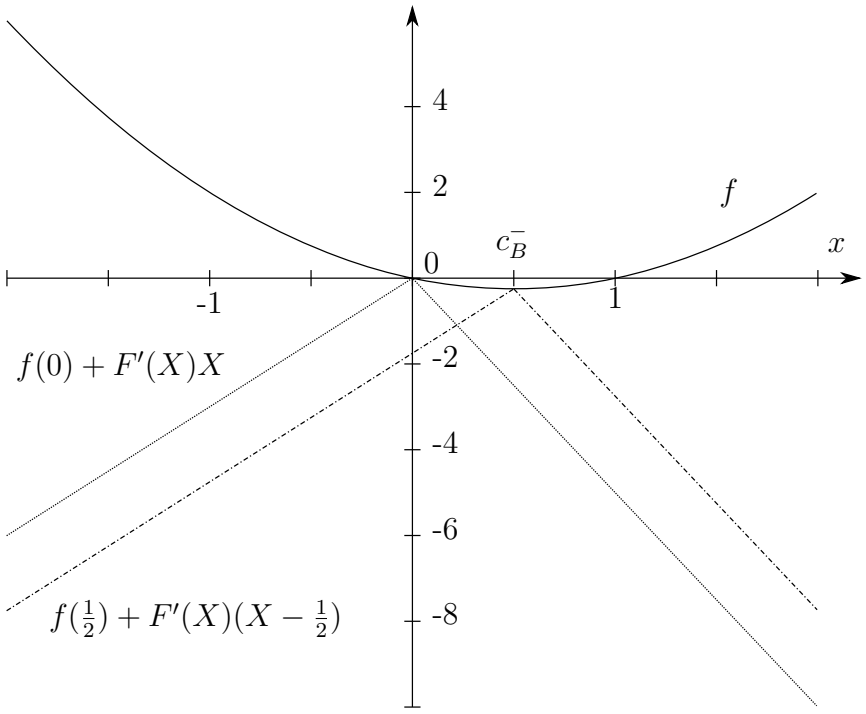
On déduit l'expression de la forme centrée pour f :

$$F_C(X, z) = f(z) + S_f(X, z)(X - z) = z^2 + z + (X + z - 1)(X - z) \tag{A.8}$$

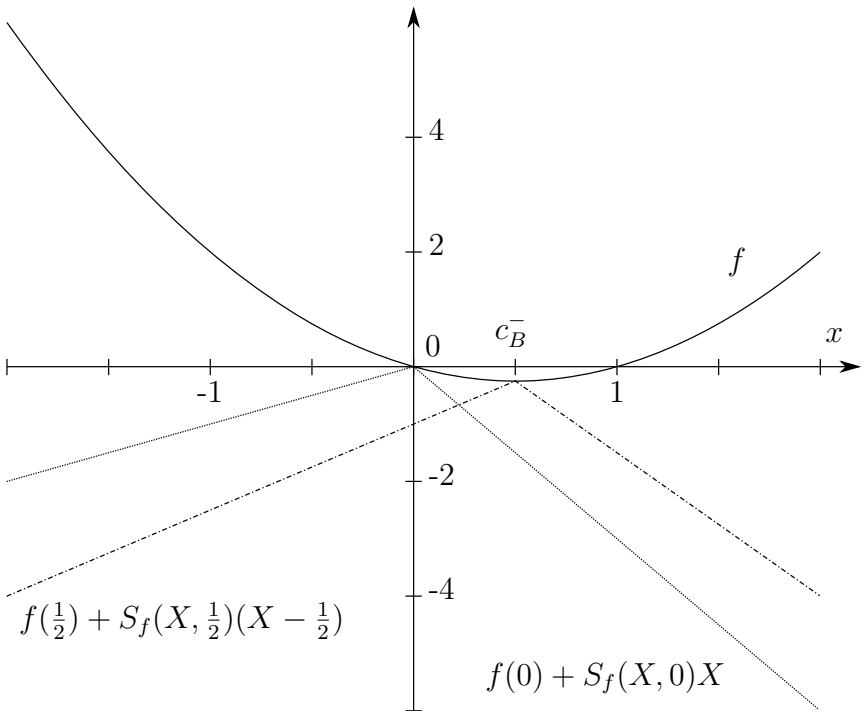
Le tableau A.1 expose pour différentes valeurs de X les encadrements $F_N(X)$ (forme naturelle), $F_T(X, m(X))$ (Taylor), $F_T(X, c_B^-)$ (Taylor avec centre optimal de Baumann), $F_C(X, m(X))$ (forme centrée) et $F_C(X, c_B^-)$ (forme centrée avec centre optimal de Baumann).

On remarque que F_N fournit la meilleure inclusion lorsque X est large. En revanche, plus la largeur de X tend vers 0, plus les formes de Taylor et centrée donnent un encadrement précis. Pour estimer l'ordre de convergence de ces fonctions d'inclusion, réécrivons l'équation 3.14 (pour une fonction d'inclusion quelconque F) :

$$w(F(X)) - w(f(X)) = Cw(X)^\alpha \tag{A.9}$$



(a) Forme de Taylor



(b) Forme centrée

FIGURE A.2 – Comparaison des minorants de $f(x) = x^2 - x$ sur $X = [-2, 2]$ fournis par la forme de Taylor et la forme centrée

TABLE A.1 – Comparaison d'inclusions pour $f(x) = x^2 - x$

X	$f(X)$	$F_N(X)$	$F_T(X, m(X))$	$F_T(X, c_B^-)$	$F_C(X, m(X))$	$F_C(X, c_B^-)$
$[-2, 2]$	$[-0.25, 6]$	$[-2, 6]$	$[-10, 10]$	$[-7.75, 12.25]$	$[-6, 6]$	$[-4, 6]$
$[-1, 2]$	$[-0.25, 2]$	$[-2, 5]$	$[-4.75, 4.25]$	$[-4.75, 4.25]$	$[-2.5, 2]$	$[-2.5, 2]$
$[0, 2]$	$[-0.25, 2]$	$[-2, 4]$	$[-3, 3]$	$[-1.75, 4.25]$	$[-2, 2]$	$[-1, 2]$
$[0, 1]$	$[-0.25, 0]$	$[-1, 1]$	$[-0.75, 0.25]$	$[-0.75, 0.25]$	$[-0.5, 0]$	$[-0.5, 0]$
$[0, 0.6]$	$[-0.25, 0]$	$[-0.6, 0.36]$	$[-0.51, 0.09]$	$[-0.35, 0.25]$	$[-0.42, 0]$	$[-0.3, 0]$

où α est l'ordre de convergence de F et C est une constante positive. Comme suggéré par [Tóth 05], on peut déterminer les valeurs de C et α au moyen d'une régression linéaire en passant au logarithme :

$$\ln(w(F(X)) - w(f(X))) = \ln(C) + \alpha \ln(w(X)) \tag{A.10}$$

On obtient :

- *pour la forme naturelle F_N : $\alpha = 0.67$ et $C = 1.46$*
- *pour la forme de Taylor F_T : $\alpha = 1.95$ et $C = 0.87$*
- *pour la forme centrée F_C : $\alpha = 1.89$ et $C = 0.36$*
- *pour la forme centrée F_C avec centre de Baumann : $\alpha = 2.21$ et $C = 0.19$*

Ces résultats confirment que la forme naturelle a une convergence (approximativement) linéaire, tandis que les formes de Taylor et centrée ont une convergence quadratique.

A.1.3 Formes bicentrée et kite

[Neumaier 90] a introduit une variante de la forme de Taylor, F_{lbvf} (*linear boundary value form*), dans laquelle les bornes de l'intervalle X sont choisies comme points de développement. Les deux minorations affines intersectées définissent une inclusion convexe de f sur X.

La fonction d'inclusion kite F_{kite} a été définie dans [Vinkó 04] comme la combinaison des deux formes F_T et F_{lbvf}. Elle fournit un minorant de l'image de f toujours au moins aussi bonne que F_T et F_{lbvf}, et en pratique généralement bien meilleure. La forme kite a été étendue aux fonctions multivariées [Messine 98, Hansen 07] et aux multisections [Lagouanelle 04].

La figure A.3 illustre la relaxation concave $\underline{F_T}$ et la relaxation convexe $\underline{F_{lbvf}}$ s'intersectant aux points R et V. La forme de Taylor $F_T(X, c)$ atteint sa borne gauche en M, soit $\underline{F_T(X, c)} = \min(y_M, y_N)$. La forme lbvf atteint sa borne gauche au point S : $\underline{F_{lbvf}}(X, c) = y_S$. L'inclusion donnée par la forme kite F_{kite}, à l'intérieur du cône convexe de F_{lbvf} et à l'extérieur du cône concave de F_T, est colorée en mauve. Sa borne gauche est $\underline{F_{kite}(X, c)} = \min(y_R, y_V)$.

A.2 Arithmétique affine

L'AA [Comba 93] est une méthode numérique qui calcule un encadrement de l'image d'une fonction sur un intervalle. A la différence de l'AI, l'AA conserve les dépendances

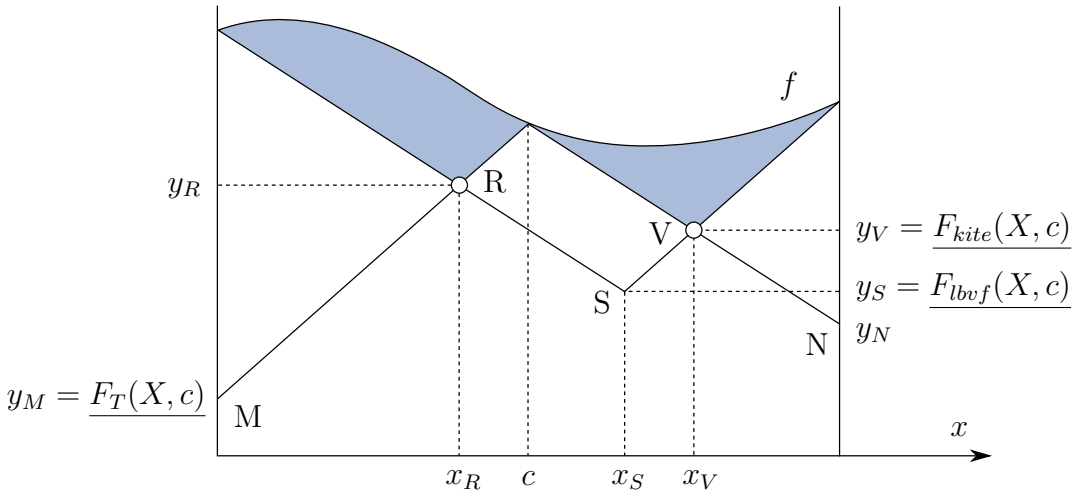


FIGURE A.3 – Fonctions d'inclusion de Taylor, *lbvf* et kite

linéaires entre variables et sous-expressions. En AA, une quantité x est représentée par une forme affine $\hat{x}$:

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i \epsilon_i \tag{A.11}$$

Les variables symboliques ϵ_i représentent des sources indépendantes d'incertitude ou d'erreur (erreurs d'arrondis, approximations des opérations élémentaires) et varient dans le domaine $[-1, 1]$. Les coefficients réels x_i déterminent l'amplitude et le signe des ϵ_i. Lorsque deux formes affines $\hat{x}$ et $\hat{y}$ partagent une variable symbolique ϵ_i, une dépendance partielle existe entre les deux quantités x et y, dans le sens où leur domaine conjoint est un polytope inclus dans le produit cartésien de leurs domaines séparés.

[Andrade 94] décrivent la conversion entre un intervalle X et sa forme affine $\hat{x}$:

$$\hat{x} = m(X) + rad(X)\epsilon_k \tag{A.12}$$

où $\epsilon_k \in [-1, 1]$ est une nouvelle variable symbolique. Inversement, le domaine des valeurs possibles d'une forme affine $\hat{x} = x_0 + \sum_{i=1}^{n} x_i \epsilon_i$ est :

$$X = [x_0 - \xi, x_0 + \xi], \quad \text{où } \xi = \sum_{i=1}^{n} |x_i| \tag{A.13}$$

Les opérations affines préservent l'ensemble des informations qui peuvent être déduites des quantités affines (exemple 17).

Exemple 17 (Opérations affines) *Soient $X = [-2, 0]$ et $Y = [2, 6]$. Les formes affines correspondantes sont $\hat{x} = -1 + \epsilon_1$ et $\hat{y} = 4 + 2\epsilon_2$. On peut vérifier que les opérations affines :*

$$\hat{x} - \hat{x} = 0 \tag{A.14}$$

et :

$$\begin{aligned}(2\hat{x}+\hat{y})-\hat{x} &= 3+\epsilon_1+2\epsilon_2 \in [0,6] \\ &= \{(2x+y)-x \mid x \in X, y \in Y\} \\ &\subset [-2,8] = (2X+Y)-X\end{aligned} \tag{A.15}$$

fournissent l'image exacte.

Les fonctions élémentaires non affines f sont remplacées par des approximations affines au premier ordre $\hat{f}$. L'erreur d'approximation est stockée dans une nouvelle variable symbolique $\epsilon_k \in [-1,1]$:

$$\hat{f}(\hat{x}) = \zeta + \alpha\hat{x} + \delta\epsilon_k \tag{A.16}$$

Le coefficient δ correspond à un majorant de la valeur absolue entre f et $\hat{f}$ pour toutes les valeurs de ϵ_i. ϵ_k est considérée comme indépendante des variables symboliques $(\epsilon_1, \ldots, \epsilon_n)$, bien qu'elle soit en réalité une fonction de ces variables. L'utilisation d'opérations non affines f introduit donc une perte d'informations.

[Stolfi 97] ont proposé deux méthodes d'approximation affine : l'approximation de Tchebychev minimise l'erreur absolue maximale, tandis que l'approximation min-range minimise le diamètre de l'image (figure A.4). Le diamètre des deux approximations varie quadratiquement avec le diamètre des variables d'entrée [Comba 93].

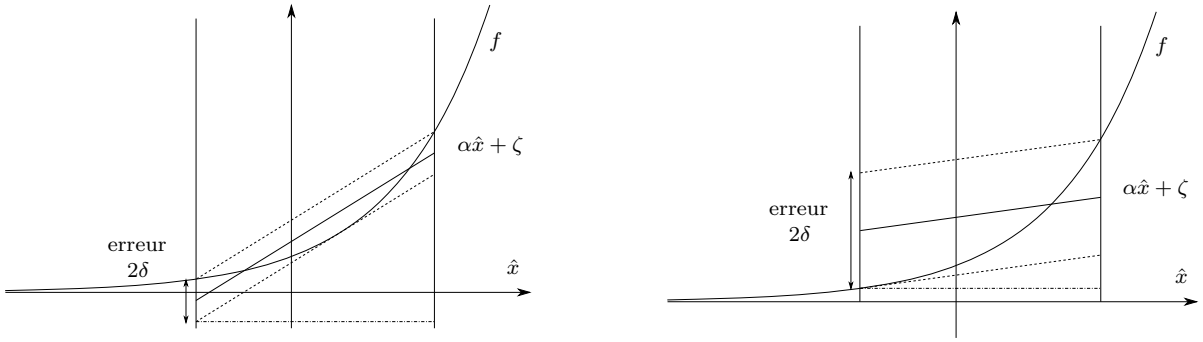


FIGURE A.4 – Approximations affines de $f(x) = \exp(x)$ sur $X = [-1, 1.5]$ par les méthodes de Tchebychev (gauche) et min-range (droite)

[Messine 02] note cependant deux choses :

- les variables symboliques introduites à chaque opération non-affine sont complètement indépendantes. Dans sa forme affine AF1, le nombre de variables symboliques ϵ_i est fixé au nombre de variables n du problème, et toutes les erreurs d'approximation sont stockées dans un seul terme $x_{n+1}\epsilon_{n+1}$, où $x_{n+1} \in \mathbb{R}_+$ et $\epsilon_{n+1} \in [-1,1]$.
- les approximations affines de la multiplication et des puissances paires contiennent des termes d'erreurs positifs (en ϵ_i^2), négatifs (en $-\epsilon_i^2$) et de signe indéterminé (en $\epsilon_i\epsilon_j$, $i \neq j$). La forme affine AF2 étend AF1 en séparant les erreurs d'approximation en

trois termes, respectivement positif $x_+\epsilon_+$, négatif $x_-\epsilon_-$ et de signe indéterminé $x_\pm\epsilon_\pm$:

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i\epsilon_i + x_\pm\epsilon_\pm + x_+\epsilon_+ + x_-\epsilon_- \tag{A.17}$$

avec $(x_\pm, x_+, x_-) \in \mathbb{R}^3_+$, $\epsilon_\pm \in [-1, 1]$, $\epsilon_+ \in [0, 1]$ et $\epsilon_- \in [-1, 0]$. Le détail de la multiplication pour AF2 est donné dans [Ninin 10a].

Pour rendre l'AA robuste aux erreurs d'arrondis, [Andrade 94] calcule un majorant de l'erreur d'arrondi pour chaque coefficient x_i de la forme affine, et l'ajoute au terme ϵ_{n+1}. Une alternative est d'utiliser des coefficients intervalles dans la forme affine, bornant ainsi automatiquement les erreurs d'arrondis par AI [Messine 02].

Dans la forme mixte de [de Figueiredo 97, Ninin 10a], chaque quantité est représentée par une forme affine et un intervalle. Les opérations élémentaires sont effectuées dans les deux arithmétiques et les domaines obtenus sont intersectés dans le but d'améliorer les approximations affines.

A.3 Postprocessing d'un programme linéaire

Cette section détaille le postprocessing proposé par [Neumaier 04] afin d'obtenir un minorant rigoureux d'un problème linéaire à partir d'une solution optimale calculée par AF. Rappelons la forme standard d'un problème linéaire :

$$\begin{array}{lll} (\mathcal{P}) & \min\limits_{\boldsymbol{x}\in\mathbb{R}^n} & \boldsymbol{c}^\top\boldsymbol{x} \\ & \text{s.c.} & A\boldsymbol{x} = \boldsymbol{b} \end{array} \tag{A.18}$$

Lorsque $\underline{\boldsymbol{b}} = -\infty$ (c'est-à-dire lorsque le problème n'est soumis qu'à des contraintes d'inégalité), le problème dual s'écrit :

$$\begin{array}{lll} (\mathcal{D}) & \max\limits_{\boldsymbol{\lambda}\in\mathbb{R}^m} & \overline{\boldsymbol{b}}^\top\boldsymbol{\lambda} \\ & \text{s.c.} & A^\top\boldsymbol{\lambda} = \boldsymbol{c} \\ & & \boldsymbol{\lambda} \leq 0 \end{array} \tag{A.19}$$

Notons $\boldsymbol{\lambda}^*$ la solution optimale de $(\mathcal{D})$. La sous-section A.3.1 détaille le calcul d'un minorant fiable du problème linéaire. La sous-section A.3.2 permet de déterminer la non réalisabilité du problème primal $(\mathcal{P})$.

A.3.1 Minorant du problème primal

Calculons un encadrement du résidu des contraintes par AI :

$$A^\top\boldsymbol{\lambda}^* - \boldsymbol{c} \in \boldsymbol{r} = [\underline{r}, \overline{r}] \tag{A.20}$$

Alors pour tout $\boldsymbol{x} \in \mathbb{R}^n$, sa valeur objectif s'exprime comme :

$$\boldsymbol{c}^\top \boldsymbol{x} \in (A^\top \boldsymbol{\lambda}^* - \boldsymbol{r})^\top \boldsymbol{x} \in \boldsymbol{\lambda}^{*\top} A\boldsymbol{x} - \boldsymbol{r}^\top \boldsymbol{x} \in \boldsymbol{\lambda}^{*\top} \boldsymbol{b} - \boldsymbol{r}^\top \boldsymbol{x} \tag{A.21}$$

On en déduit un minorant rigoureux de l'image de f sur $\boldsymbol{X}$:

$$f(\boldsymbol{X}) \geq \underline{F(\underline{X}) + \boldsymbol{\lambda}^{*\top} \boldsymbol{b} - \boldsymbol{r}^\top \boldsymbol{x} - \sum_{i=1}^{n} \frac{\partial F}{\partial x_i}(\boldsymbol{X}) \cdot \underline{X_i}} \tag{A.22}$$

A.3.2 Non réalisabilité du problème primal

Calculons un encadrement rigoureux du résidu par AI :

$$A^\top \boldsymbol{\lambda}^* \in \boldsymbol{r} = [\underline{r}, \overline{r}] \tag{A.23}$$

Pour tout $\boldsymbol{x} \in \mathbb{R}^n$ réalisable, on a :

$$0 \in (\boldsymbol{r} - A^\top \boldsymbol{\lambda}^*)^\top \boldsymbol{x} \in \boldsymbol{r}^\top \boldsymbol{x} - \boldsymbol{\lambda}^{*\top} A\boldsymbol{x} \in \boldsymbol{r}^\top \boldsymbol{x} - \boldsymbol{\lambda}^{*\top} \boldsymbol{b} \tag{A.24}$$

La non réalisabilité du problème primal est alors garantie lorsque :

$$0 \notin \boldsymbol{r}^\top \boldsymbol{x} - \boldsymbol{\lambda}^{*\top} \boldsymbol{b} \tag{A.25}$$

Bibliographie

[Alba 05] Enrique Alba. Parallel metaheuristics : a new class of algorithms, volume 47. John Wiley & Sons, 2005. *Cité p.* 66

[Alliot 97] J-M. Alliot, J-F. Bosc, N. Durand et L. Maugis. *CATS : A complete Air Traffic Simulator*. 16th Digital Avionics Systems Conference, 1997. *Cité p.* 113, 116

[Alliot 12a] J.-M. Alliot, N. Durand, D. Gianazza et J.-B. Gotteland. *Finding and Proving the Optimum : Cooperative Stochastic and Deterministic Search*. 20th European Conference on Artificial Intelligence (ECAI 2012), August 27-31, 2012, Montpellier, France, 2012. *Cité p.* 68, 98, 109, 149

[Alliot 12b] Jean-Marc Alliot, Jean-Baptiste Gotteland, Charlie Vanaret, Nicolas Durand et David Gianazza. *Implementing an interval computation library for OCaml on x86/amd64 architectures*. Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming, 2012. *Cité p.* 29

[Andrade 94] MVA Andrade, JLD Comba et J Stolfi. *Affine arithmetic*. Interval 94, pages 5–10, 1994. *Cité p.* 158, 160

[Araya 10] Ignacio Araya, Gilles Trombettoni et Bertrand Neveu. *Exploiting monotonicity in interval constraint propagation*. Proc. AAAI, pages 9–14, 2010. *Cité p.* 37, 51

[Araya 12] Ignacio Araya, Gilles Trombettoni et Bertrand Neveu. *A contractor based on convex interval taylor*. Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems, pages 1–16. Springer, 2012. *Cité p.* 60, 74, 75, 92, 93, 94, 95, 151

[Baumann 88] Eckart Baumann. *Optimal centered forms*. BIT Numerical Mathematics, vol. 28, pages 80–87, 1988. *Cité p.* 35, 154

[Belotti 09] Pietro Belotti, Jon Lee, Leo Liberti, Francois Margot et Andreas Wächter. *Branching and bounds tighteningtechniques for non-convex MINLP*. Optimization Methods & Software, vol. 24, no. 4-5, pages 597–634, 2009. *Cité p.* 50, 96

[Benhamou 94] Frédéric Benhamou, David A. McAllester et Pascal Van Hentenryck. *CLP(Intervals) Revisited*. ILPS, pages 124–138, 1994. *Cité p.* 52

[Benhamou 99] F. Benhamou, F. Goualard, L. Granvilliers et J-F. Puget. *Revising Hull and Box Consistency*. International Conference on Logic Programming, pages 230–244. MIT press, 1999. *Cité p.* 50

[Benhamou 00] Frédéric Benhamou et Frédéric Goualard. *Universally quantified interval constraints*. Principles and Practice of Constraint Programming–CP 2000, pages 67–82. Springer, 2000. *Cité p.* 132, 133

[Blum 11] Christian Blum, Jakob Puchinger, Günther R Raidl et Andrea Roli. *Hybrid metaheuristics in combinatorial optimization : A survey*. Applied Soft Computing, vol. 11, no. 6, pages 4135–4151, 2011. *Cité p.* 68

[Brönnimann 06] Hervé Brönnimann, Guillaume Melquiond et Sylvain Pion. *The design of the Boost interval arithmetic library*. Theoretical Computer Science, vol. 351, no. 1, pages 111–118, 2006. *Cité p.* 28

[Cafieri 10] S. Cafieri, P. Brisset et N. Durand. *A mixed-integer optimization model for air traffic deconfliction*. Proceedings of the Toulouse Global Optimization Workshop, pages 27–30, 2010. *Cité p.* 113

[Caprani 80] Ole Caprani et Kaj Madsen. *Mean value forms in interval analysis*. Computing, vol. 25, no. 2, pages 147–154, 1980. *Cité p.* 34

[Chabert 09a] G. Chabert et L. Jaulin. *Contractor programming*. Artificial Intelligence, vol. 173, pages 1079–1100, 2009. *Cité p.* 48, 49, 92

[Chabert 09b] Gilles Chabert et Luc Jaulin. *Hull consistency under monotonicity*. Principles and Practice of Constraint Programming-CP 2009, pages 188–195. Springer, 2009. *Cité p.* 52

[Cleary 87] John G Cleary. *Logical arithmetic*. Future Computing Systems, vol. 2, pages 125–149, 1987. *Cité p.* 50

[Collavizza 99a] Hélene Collavizza, François Delobel et Michel Rueher. *Comparing partial consistencies*. Reliable computing, vol. 5, no. 3, pages 213–228, 1999. *Cité p.* 52

[Collavizza 99b] Hélène Collavizza, François Delobel et Michel Rueher. *Extending consistent domains of numeric CSP*. IJCAI, volume 99, pages 406–413, 1999. *Cité p.* 133

[Comba 93] João Luiz Dihl Comba et Jorge Stolfi. *Affine Arithmetic and its Applications to Computer Graphics*. Proceedings of SIBGRAPI'93 - VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, pages 9–18, 1993. *Cité p.* 157, 159

[Cotta 95] C Cotta, JF Aldana, AJ Nebro et JM Troya. *Hybridizing genetic algorithms with branch and bound techniques for the resolution of the TSP*. Artificial Neural Nets and Genetic Algorithms, pages 277–280. Springer, 1995. *Cité p.* 68

[Cotta 03] Carlos Cotta et José M Troya. *Embedding branch and bound within evolutionary algorithms*. Applied Intelligence, vol. 18, no. 2, pages 137–153, 2003. *Cité p.* 67

[Csendes 97] Tibor Csendes et Dietmar Ratz. *Subdivision direction selection in interval methods for global optimization.* SIAM Journal on Numerical Analysis, vol. 34, no. 3, pages 922–938, 1997. *Cité p.* 41

[de Figueiredo 97] Luiz Henrique de Figueiredo, Ronald Van Iwaarden et Jorge Stolfi. *Fast interval branch-and-bound methods for unconstrained global optimization with affine arithmetic.* Rapport technique IC-9708, Institute of Computing, Univ. of Campinas, 1997. *Cité p.* 160

[Delahaye 10] D Delahaye, C Peyronne, M Mongeau et S Puechmorel. *Aircraft Conflict Resolution by Genetic Algorithm and B-Spline Approximation.* Proceedings of ENRI Int. Workshop on ATM/CNS. Tokyo, Japan, volume 4, 2010. *Cité p.* 113, 114

[Du 94] Kaisheng Du et R Baker Kearfott. *The cluster problem in multivariate global optimization.* Journal of Global Optimization, vol. 5, no. 3, pages 253–265, 1994. *Cité p.* 42

[Durand 96] N. Durand, J.M. Alliot et J. Noailles. *Automatic aircraft conflict resolution using Genetic Algorithms.* Proceedings of the Symposium on Applied Computing, 1996. *Cité p.* 113, 114, 116, 141

[Durand 00] N. Durand, J-M. Alliot et Geraud Granger. *FACES : a Free flight Autonomous and Coordinated Embarked Solver.* ATC Quarterly, 2000. *Cité p.* 113

[Feltl 04] Harald Feltl et Günther R Raidl. *An improved hybrid genetic algorithm for the generalized assignment problem.* Proceedings of the 2004 ACM symposium on Applied computing, pages 990–995. ACM, 2004. *Cité p.* 67

[Feo 89] Thomas A Feo et Mauricio GC Resende. *A probabilistic heuristic for a computationally difficult set covering problem.* Operations research letters, vol. 8, no. 2, pages 67–71, 1989. *Cité p.* 12

[Focacci 03] Filippo Focacci, François Laburthe et Andrea Lodi. *Local search and constraint programming.* Handbook of metaheuristics, pages 369–403. Springer, 2003. *Cité p.* 87

[Gallardo 07] José E Gallardo, Carlos Cotta et Antonio J Fernández. *On the hybridization of memetic algorithms with branch-and-bound techniques.* Systems, Man, and Cybernetics, Part B : Cybernetics, IEEE Transactions on, vol. 37, no. 1, pages 77–83, 2007. *Cité p.* 68

[Gay 10] David M Gay. *Bounds from Slopes.* Rapport technique, Sandia National Laboratories, Albuquerque, 2010. *Cité p.* 155

[Gay 12] David Gay. *Using Expression Graphs in Optimization Algorithms.* Mixed Integer Nonlinear Programming, page 247, 2012. *Cité p.* 155

[Glover 90] Fred Glover. *Tabu search : A tutorial.* Interfaces, vol. 20, no. 4, pages 74–94, 1990. *Cité p.* 12

[Goldberg 87] David E Goldberg et Jon Richardson. *Genetic algorithms with sharing for multimodal function optimization*. Genetic algorithms and their applications : Proceedings of the Second International Conference on Genetic Algorithms, pages 41–49. Hillsdale, NJ : Lawrence Erlbaum, 1987. *Cité p.* 21

[Goldberg 89] David E Goldberg. *Genetic algorithms in search, optimization and machine learning*. Reading : Addison-Wesley, 1989. *Cité p.* 18

[Goldberg 91] David Goldberg. *What every computer scientist should know about floating-point arithmetic*. ACM Computing Surveys (CSUR), vol. 23, no. 1, pages 5–48, 1991. *Cité p.* 28

[Goldsztejn 09] Alexandre Goldsztejn, Claude Michel et Michel Rueher. *Efficient handling of universally quantified inequalities*. Constraints, vol. 14, no. 1, pages 117–135, 2009. *Cité p.* 132, 133

[Goualard 03] Frédéric Goualard. *Gaol, Not Just Another Interval Library*. http ://www.sourceforge.net/projects/gaol, 2003. *Cité p.* 28

[Goualard 08] Frédéric Goualard. *Interval Extensions of Multivalued Inverse Functions*. ACM Transactions on Mathematical Software, vol. 5, 2008. *Cité p.* 50

[Hansen 68] Eldon R Hansen. *On solving systems of equations using interval arithmetic*. Mathematics of Computation, pages 374–384, 1968. *Cité p.* 34, 74, 151

[Hansen 92] E. Hansen. Global optimization using interval analysis. Dekker, 1992. *Cité p.* 30, 53, 151

[Hansen 07] Pierre Hansen, Jean-Louis Lagouanelle et Frédéric Messine. *Comparison between Baumann and admissible simplex forms in interval analysis*. Journal of Global Optimization, vol. 37, no. 2, pages 215–228, 2007. *Cité p.* 157

[Hanson 68] Richard J Hanson. *Interval arithmetic as a closed arithmetic system on a computer*. Jet Propulsion Laboratory report, vol. 197, 1968. *Cité p.* 30

[Hoare 71] M.R. Hoare et P. Pal. *Physical cluster mechanics : Statics and energy surfaces for monatomic systems*. Advances in Physics, vol. 20, no. 84, pages 161–196, 1971. *Cité p.* 145

[Holland 75] J. H. Holland. Adaptation in natural and artificial systems. University of Michigan Press, 1975. *Cité p.* 18

[Hooke 61] Robert Hooke et To A Jeeves. *"Direct Search"Solution of Numerical and Statistical Problems*. Journal of the ACM (JACM), vol. 8, no. 2, pages 212–229, 1961. *Cité p.* 12

[Ichida 79] K Ichida et Y Fujii. *An interval arithmetic method for global optimization*. Computing, vol. 23, no. 1, pages 85–97, 1979. *Cité p.* 41

[Isaacson 97] D. R. Isaacson et H. Erzberger. *Design of a conflict detection algorithm for the Center/TRACON automation system.* Digital Avionics Systems Conference, 1997. *Cité p.* 113

[Jeffreys 99] Harold Jeffreys et Bertha Jeffreys. Methods of mathematical physics. Cambridge university press, 1999. *Cité p.* 34

[Jones 24] J. E. Jones. *On the Determination of Molecular Fields. I. From the Variation of the Viscosity of a Gas with Temperature.* Proceedings of the Royal Society of London. Series A, vol. 106, no. 738, pages 441–462, 1924. *Cité p.* 143

[Jourdan 09] Laetitia Jourdan, Matthieu Basseur et E-G Talbi. *Hybridizing exact methods and metaheuristics : A taxonomy.* European Journal of Operational Research, vol. 199, no. 3, pages 620–629, 2009. *Cité p.* 66

[Kahan 68] WM Kahan. *A more complete interval arithmetic.* Lecture notes for a summer course at the University of Michigan, 1968. *Cité p.* 30

[Kang 02] Lishan Kang, Zhou Kang, Yan Li et Hugo de Garis. *A Two Level Evolutionary Modeling System For Financial Data.* GECCO, pages 1113–1118, 2002. *Cité p.* 98

[Karush 39] William Karush. *Minima of functions of several variables with inequalities as side constraints.* PhD thesis, Master's thesis, Dept. of Mathematics, Univ. of Chicago, 1939. *Cité p.* 8

[Keane 94] Andy Keane. *Bump, A Hard(?) Problem.* http ://www.southampton.ac.uk/˜ajk/bump.html, 1994. *Cité p.* 96

[Kearfott 93] Baker Kearfott et Kaisheng Du. *The cluster problem in global optimization : The univariate case.* Validation Numerics, pages 117–127. Springer, 1993. *Cité p.* 42

[Kearfott 96a] R. Baker Kearfott. *Interval Extensions of Non-Smooth Functions for Global Optimization and Nonlinear Systems Solvers.* Computing, vol. 57, pages 57–149, 1996. *Cité p.* 98

[Kearfott 96b] R Baker Kearfott. Rigorous global search : continuous problems. Springer, 1996. *Cité p.* 73, 92

[Kearfott 05] R Baker Kearfott et Siriporn Hongthong. *Validated linear relaxations and preprocessing : some experiments.* SIAM Journal on Optimization, vol. 16, no. 2, pages 418–433, 2005. *Cité p.* 92

[Kennedy 95] J. Kennedy et R. Eberhart. *Particle Swarm Optimization.* Proceedings of IEEE International Conference on Neural Networks, pages 1942–1948, 1995. *Cité p.* 22

[Kirkpatrick 83] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi*et al. Optimization by simmulated annealing.* science, vol. 220, no. 4598, pages 671–680, 1983. *Cité p.* 12

[Knüppel 94] Olaf Knüppel. *PROFIL/BIAS - a fast interval library.* Computing, vol. 53, no. 3-4, pages 277–287, 1994. *Cité p.* 28

[Kolev 97] Lubomir V Kolev. *Use of interval slopes for the irrational part of factorable functions.* Reliable Computing, vol. 3, no. 1, pages 83–93, 1997. *Cité p.* 154

[Krawczyk 82] R. Krawczyk et K. Nickel. *The centered form in interval arithmetics - Quadratic convergence and inclusion isotonicity.* Computing, vol. 28, no. 2, pages 117–137, 1982. *Cité p.* 34

[Krawczyk 85] Rudolf Krawczyk et Arnold Neumaier. *Interval slopes for rational functions and associated centered forms.* SIAM Journal on Numerical Analysis, vol. 22, no. 3, pages 604–616, 1985. *Cité p.* 154

[Lagouanelle 04] Jean-Louis Lagouanelle et Gérard Soubry. *Optimal multisections in interval branch-and-bound methods of global optimization.* Journal of Global Optimization, vol. 30, no. 1, pages 23–38, 2004. *Cité p.* 157

[Lawler 66] Eugene L Lawler et David E Wood. *Branch-and-bound methods : A survey.* Operations research, vol. 14, no. 4, pages 699–719, 1966. *Cité p.* 38

[Leary 97] R.H. Leary. *Global Optima of Lennard-Jones Clusters.* J. of Global Optimization, vol. 11, no. 1, pages 35–53, 1997. *Cité p.* 145

[Lebbah 05a] Yahia Lebbah, Claude Michel et Michel Rueher. *Efficient pruning technique based on linear relaxations.* Global Optimization and Constraint Satisfaction, pages 1–14. Springer, 2005. *Cité p.* 64

[Lebbah 05b] Yahia Lebbah, Claude Michel, Michel Rueher, David Daney et Jean-Pierre Merlet. *Efficient and safe global constraints for handling numerical constraint systems.* SIAM Journal on Numerical Analysis, vol. 42, no. 5, pages 2076–2097, 2005. *Cité p.* 73

[Lerch 01] M. Lerch, G. Tischler, J. Wolff von Gudenberg, W. Hofschuster et W. Krämer. *FILIB++ Interval Library.* http ://www.math.uni-wuppertal.de/org/WRST/software/filib.html, 2001. *Cité p.* 28

[Lhomme 93] Olivier Lhomme. *Consistency techniques for numeric CSPs.* IJCAI, volume 93, pages 232–238. Citeseer, 1993. *Cité p.* 56

[Locatelli 03] Marco Locatelli et Fabio Schoen. *Efficient Algorithms for Large Scale Global Optimization : Lennard-Jones Clusters.* Comput. Optim. Appl., vol. 26, no. 2, pages 173–190, 2003. *Cité p.* 145

[Mackworth 77] Alan K Mackworth. *Consistency in networks of relations.* Artificial intelligence, vol. 8, no. 1, pages 99–118, 1977. *Cité p.* 43

[Mentzer 91] Stuart G Mentzer. *LP-form inclusion functions for global optimization.* Computers & Mathematics with Applications, vol. 21, no. 6, pages 51–65, 1991. *Cité p.* 60

[Messine 97] Frédéric Messine. *Méthodes d'optimisation globale basées sur l'analyse d'intervalles pour la résolution de problemes avec contraintes*. PhD thesis, INPT-ENSEEIHT, Toulouse, 1997. *Cité p.* 37, 50

[Messine 98] Frédéric Messine et Jean-Louis Lagouanelle. *Enclosure methods for multivariate differentiable functions and application to global optimization*. Journal of Universal Computer Science, vol. 4, no. 6, pages 589–603, 1998. *Cité p.* 157

[Messine 02] Frédéric Messine. *Extensions of Affine Arithmetic : Application to Unconstrained Global Optimization*. Journal of Universal Computer Science, vol. 8, no. 11, pages 992–1015, 2002. *Cité p.* 159, 160

[Michalewicz 96a] Zbigniew Michalewicz. Genetic algorithms + data structures = evolution programs. Springer, 1996. *Cité p.* 20, 96

[Michalewicz 96b] Zbigniew Michalewicz et Marc Schoenauer. *Evolutionary algorithms for constrained parameter optimization problems*. Evolutionary computation, vol. 4, no. 1, pages 1–32, 1996. *Cité p.* 17

[Microsystems 01] SUN Microsystems. C++ interval arithmetic programming manual. SUN, Palo Alto, California, 2001. *Cité p.* 28

[Mimram 04] Samuel Mimram. *ocaml-glpk*. http ://ocaml-glpk.sourceforge.net/, 2004. *Cité p.* 75

[Mishra 06] S. K. Mishra. *Some new test functions for global optimization and performance of repulsive particle swarm method*. Rapport technique, University Library of Munich, Germany, 2006. *Cité p.* 98

[Moore 66] R. E. Moore. Interval analysis. Prentice-Hall, 1966. *Cité p.* 28, 32, 34

[Moore 76] Ramon E Moore. *On computing the range of a rational function of n variables over a bounded region*. Computing, vol. 16, no. 1, pages 1–15, 1976. *Cité p.* 39

[Moscato 04] Pablo Moscato, Carlos Cotta et Alexandre Mendes. *Memetic algorithms*. New optimization techniques in engineering, pages 53–85. Springer, 2004. *Cité p.* 67

[Nelder 65] John A Nelder et Roger Mead. *A simplex method for function minimization*. The computer journal, vol. 7, no. 4, pages 308–313, 1965. *Cité p.* 12

[Neumaier 90] Arnold Neumaier. Interval methods for systems of equations, volume 37. Cambridge university press, 1990. *Cité p.* 53, 83, 157

[Neumaier 04] Arnold Neumaier et Oleg Shcherbina. *Safe bounds in linear and mixed-integer linear programming*. Mathematical Programming, vol. 99, no. 2, pages 283–296, 2004. *Cité p.* 75, 160

[Neumaier 05] Arnold Neumaier, Oleg Shcherbina, Waltraud Huyer et Tamás Vinkó. *A comparison of complete global optimization solvers*. Mathematical programming, vol. 103, no. 2, pages 335–356, 2005. *Cité p.* 12

[Ninin 10a] Jordan Ninin. *Optimisation Globale basée sur l'Analyse d'Intervalles : Relaxation Affine et Limitation de la Mémoire*. PhD thesis, Institut National Polytechnique de Toulouse-INPT, 2010. *Cité p.* 160

[Ninin 10b] Jordan Ninin, Pierre Hansen et Frédéric Messine. A reliable affine relaxation method for global optimization. Groupe d'études et de recherche en analyse des décisions, 2010. *Cité p.* 37, 61, 73, 92, 93, 94

[Northby 87] J. A. Northby. *Structure and binding of Lennard-Jones clusters : 13 <= N <= 147*. The Journal of Chemical Physics, vol. 87, no. 10, pages 6166–6177, 1987. *Cité p.* 145

[Oliveira 96] João B Oliveira. *New slope methods for sharper interval functions and a note on Fischer's acceleration method*. Reliable Computing, vol. 2, no. 3, pages 299–320, 1996. *Cité p.* 155

[Oplatková 08] Zuzana Oplatková. *Metaevolution - Synthesis of Evolutionary Algorithms by Means of Symbolic Regression*. PhD thesis, Tomas Bata University in Zlín, 2008. *Cité p.* 98

[Pallottino 02] L. Pallottino, E. Feron et A. Bicchi. *Conflict resolution problems for air traffic management systems solved with mixed integer programming*. IEEE transactions on intelligent transportation systems, vol. 3, Mars 2002. *Cité p.* 113, 114

[Pohl 10] Jan Pohl, Václav Jirsík et Petr Honzík. *Stochastic Optimization Algorithm with Probability Vector in Mathematical Function Minimization and Travelling Salesman Problem*. WSEAS Trans. Info. Sci. and App., vol. 7, pages 975–984, 2010. *Cité p.* 98

[Price 06] K. Price, R. Storn et J. Lampinen. Differential evolution - a practical approach to global optimization. Natural Computing. Springer-Verlag, 2006. *Cité p.* 17, 24, 25

[Puchinger 05] Jakob Puchinger et Günther R Raidl. *Combining metaheuristics and exact algorithms in combinatorial optimization : A survey and classification*. Artificial intelligence and knowledge engineering applications : a bioinspired approach, pages 41–53. Springer, 2005. *Cité p.* 66

[Raidl 06] Günther R Raidl. *A unified view on hybrid metaheuristics*. Hybrid Metaheuristics, pages 1–12. Springer, 2006. *Cité p.* 66

[Ratz 96] Dietmar Ratz. An optimized interval slope arithmetic and its application. Citeseer, 1996. *Cité p.* 154

[Revol 02] Nathalie Revol et Fabrice Rouillier. *Motivations for an arbitrary precision interval arithmetic and the MPFI library*. Rapport de recherche, INRIA, 2002. *Cité p.* 28

[Rogalsky 00] T Rogalsky, S Kocabiyik et RW Derksen. *Differential evolution in aerodynamic optimization*. Canadian Aeronautics and Space Journal, vol. 46, no. 4, pages 183–190, 2000. *Cité p.* 23

[Rump 88] S. M. Rump. *Algorithms for verified inclusions - theory and practice.* Reliability in computing, pages 109–126. Academic Press Professional, Inc., 1988. *Cité p.* 28

[Rump 96] S. M. Rump. *Expansion and estimation of the range of nonlinear functions.* Mathematics of Computation of the American Mathematical Society, vol. 65, no. 216, pages 1503–1512, 1996. *Cité p.* 154

[Ryoo 95] Hong S Ryoo et Nikolaos V Sahinidis. *Global optimization of nonconvex NLPs and MINLPs with applications in process design.* Computers & Chemical Engineering, vol. 19, no. 5, pages 551–566, 1995. *Cité p.* 39

[Sahinidis 96] Nikolaos V Sahinidis. *BARON : A general purpose global optimization software package.* Journal of Global Optimization, vol. 8, no. 2, pages 201–205, 1996. *Cité p.* 107

[Schichl 05] Hermann Schichl et Arnold Neumaier. *Interval analysis on directed acyclic graphs for global optimization.* Journal of Global Optimization, vol. 33, no. 4, pages 541–562, 2005. *Cité p.* 61, 62

[Schnurr 08] Marco Schnurr. *Computing slope enclosures by exploiting a unique point of inflection.* Applied Mathematics and Computation, vol. 204, no. 1, pages 249–256, 2008. *Cité p.* 154

[Skelboe 74] Stig Skelboe. *Computation of rational interval functions.* BIT Numerical Mathematics, vol. 14, no. 1, pages 87–95, 1974. *Cité p.* 39

[Sloane 95] N.J.A. Sloane, R.H. Hardin, T.D.S. Duff et J.H. Conway. *Minimal-energy clusters of hard spheres.* Discrete & Computational Geometry, vol. 14, pages 237–259, 1995. *Cité p.* 145

[Slowik 08] Adam Slowik et Michal Bialko. *Training of artificial neural networks using differential evolution algorithm.* Human System Interactions, 2008 Conference on, pages 60–65. IEEE, 2008. *Cité p.* 23

[Sotiropoulos 97] DG Sotiropoulos, EC Stavropoulos et MN Vrahatis. *A new hybrid genetic algorithm for global optimization.* Nonlinear Analysis : Theory, Methods & Applications, vol. 30, no. 7, pages 4529–4538, 1997. *Cité p.* 67

[Sotiropoulos 05] DG Sotiropoulos et TN Grapsa. *Optimal centers in branch-and-prune algorithms for univariate global optimization.* Applied mathematics and computation, vol. 169, no. 1, pages 247–277, 2005. *Cité p.* 43

[Stolfi 97] Jorge Stolfi et Luiz Henrique De Figueiredo. *Self-Validated Numerical Methods and Applications.* Monograph for 21st Brazilian Mathematics Colloquium, 1997. *Cité p.* 159

[Storn 97] R. Storn et K. Price. *Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces.* Journal of Global Optimization, pages 341–359, 1997. *Cité p.* 23

[Talbi 02] E-G Talbi. *A taxonomy of hybrid metaheuristics.* Journal of heuristics, vol. 8, no. 5, pages 541–564, 2002. *Cité p.* 66

[Talbi 09] El-Ghazali Talbi. Metaheuristics : from design to implementation, volume 74. John Wiley & Sons, 2009. *Cité p.* 17

[Tao 07] Jili Tao et Ning Wang. *DNA computing based RNA genetic algorithm with applications in parameter estimation of chemical engineering processes.* Computers & Chemical Engineering, vol. 31, no. 12, pages 1602 – 1618, 2007. *Cité p.* 98

[Tóth 05] Boglárka Tóth et Tibor Csendes. *Empirical investigation of the convergence speed of inclusion functions in a global optimization context.* Reliable Computing, vol. 11, no. 4, pages 253–273, 2005. *Cité p.* 157

[Trombettoni 07] Gilles Trombettoni et Gilles Chabert. *Constructive interval disjunction.* Principles and Practice of Constraint Programming–CP 2007, pages 635–650. Springer, 2007. *Cité p.* 58

[Trombettoni 11] Gilles Trombettoni, Ignacio Araya, Bertrand Neveu et Gilles Chabert. *Inner Regions and Interval Linearizations for Global Optimization.* AAAI, 2011. *Cité p.* 73

[Van Hentenryck 97] Pascal Van Hentenryck. Numerica : a modeling language for global optimization. MIT press, 1997. *Cité p.* 43

[Vanaret 12] Charlie Vanaret, David Gianazza, Nicolas Durand et Jean-Baptiste Gotteland. *Benchmarking conflict resolution algorithms.* 5th International Conference on Research in Air Transportation, ICRAT, Berkeley, USA, 2012. *Cité p.* 113, 114

[Vanaret 13] Charlie Vanaret, Jean-Baptiste Gotteland, Nicolas Durand et Jean-Marc Alliot. *Preventing Premature Convergence and Proving the Optimality in Evolutionary Algorithms.* International Conference on Artificial Evolution (EA-2013), pages 84–94, 2013. *Cité p.* 69

[Vanaret 14] Charlie Vanaret, Jean-Baptiste Gotteland et Nicolas Durand. *Nouvelle formulation pour la résolution de conflits aériens.* 15ème congrès annuel de la Société française de recherche opé rationnelle et d'aide à la décision (ROADEF 2014), 2014. *Cité p.* 129

[Vavasis 94] Stephen A. Vavasis. *Open problems.* Journal of Global Optimization, vol. 4, pages 343–344, 1994. *Cité p.* 145

[Vinkó 04] Tamás Vinkó, Jean-Louis Lagouanelle et Tibor Csendes. *A new inclusion function for optimization : Kite–the one dimensional case.* Journal of Global Optimization, vol. 30, no. 4, pages 435–456, 2004. *Cité p.* 157

[Wales 97] David J. Wales et Jonathan P. K. Doye. *Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters*

Containing up to 110 Atoms. The Journal of Physical Chemistry A, vol. 101, no. 28, pages 5111–5116, 1997. *Cité p.* 145

[Whitley 96] D. Whitley, K. Mathias, S. Rana et J. Dzubera. *Evaluating Evolutionary Algorithms*. Artificial Intelligence, vol. 85, pages 245–276, 1996. *Cité p.* 96

[Yamamura 98] Kiyotaka Yamamura, Hitomi Kawata et Ai Tokue. *Interval solution of nonlinear equations using linear programming*. BIT Numerical Mathematics, vol. 38, no. 1, pages 186–199, 1998. *Cité p.* 43, 60

[Yamamura 02] Kiyotaka Yamamura et Shigeru Tanaka. *Finding all solutions of systems of nonlinear equations using the dual simplex method*. BIT Numerical Mathematics, vol. 42, no. 1, pages 214–230, 2002. *Cité p.* 60

[Yin 93] Xiaodong Yin et Noel Germay. *A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization*. Artificial neural nets and genetic algorithms, pages 450–457. Springer, 1993. *Cité p.* 21

[Zeghal 98] K. Zeghal. *A Comparison of Different Approaches based on Force Fields for Coordination among Multiple Mobiles*. Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1998. *Cité p.* 113

[Zhang 07] Xiaowei Zhang et Sanyang Liu. *A new interval-genetic algorithm*. Natural Computation, 2007. ICNC 2007. Third International Conference on, volume 4, pages 193–197. IEEE, 2007. *Cité p.* 67

[Zuhe 90] Shen Zuhe et Michael A Wolfe. *On interval enclosures using slope arithmetic*. Applied Mathematics and Computation, vol. 39, no. 1, pages 89–105, 1990. *Cité p.* 155