PSL UNIVERSITÉ PARIS

FACULTE DES SCIENCES DE TUNIS

UNIVERSITÉ DE TUNIS EL MANAR

THÈSE DE DOCTORAT

DE L'UNIVERSITÉ PSL

Préparée à l'Université Paris-Dauphine

Dans le cadre d'une cotutelle avec l'Université de Tunis El Manar

Extension des programmes génétiques pour l'apprentissage supervisé à partir de très larges bases de données (Big Data)

Extending genetic programming for supervised learning from very large datasets (Big Data)

Soutenue par

Hmida HMIDA

Le 23 octobre 2019

Ecole doctorale n° ED 543

Ecole doctorale de Dauphine

Spécialité

Informatique

Composition du jury :

Cyril, FONLUPT Professeur, Université du Littoral – Côte d'Opale	*Rapporteur*
Nadia, ESSOUSSI Professeure, Université de Tunis	*Rapporteur*
Mohamed Mohsen, GAMMOUDI Professeur, Université de la Manouba	*Président du jury*
Tristan, CAZENAVE Professeur, Université Paris-Dauphine	*Examinateur*
Marta, RUKOZ Professeure, Université Paris Nanterre	*Directrice de thèse*
Amel, BORGI Maîtresse de conférences, Université Tunis El Manar	*Directrice de thèse*
Sana, BEN HAMIDA Maîtresse de conférences, Université Paris Nanterre	*Co-encadrante*

Dauphine | PSL UNIVERSITÉ PARIS

Dédicaces

À la mémoire de mon père Ben Mahmoud,

À la mémoire de mon frère Zouhaier,

À ma mère Fattoum,

À ma femme Monia,

À mes enfants Malek, Adem et Lina,

À mes frères et sœurs Jalel, Maher, Zoubaier, Issam, Mbarka,

Nissaf, Kaouther et Dhia-Elhak,

À tous les membres de ma famille.

Remerciements

الحمد لله الذي بنعمته تتم الصالحات

L'accomplissement de ce travail a été possible grâce à plusieurs personnes qui ont eu une influence positive à travers leurs contributions scientifiques ou morales. C'est avec plaisir et reconnaissance que je leurs réserve ces lignes.

Je souhaite remercier mes directrices de thèse qui m'ont guidé tout au long de ces années. Je remercie Amel BORGI pour ces remarques pertinentes, sa rigueur scientifique et son suivi exemplaire. Je remercie vivement Marta RUKOZ, qui a accepté de codiriger cette thèse, et m'a poussé toujours vers la perfection à travers nos longues discussions.

J'adresse un remerciement particulier à Sana BEN HAMIDA pour sa confiance en me proposant ce sujet. Je la remercie aussi pour son co-encadrement, sa disponibilité toutes ces années et ses nombreuses idées et propositions qui ont imprégné ce travail.

Je remercie Cyril FONLUPT et Nadia ESSOUSSI, qui ont accepté d'être rapporteurs de ma thèse, pour l'intérêt qu'ils ont porté à mes travaux et leurs précieuses suggestions. Je remercie également Mohamed Mohsen GAMMOUDI et Tristan CAZENAVE d'être examinateurs de ce travail. Vous m'avez fait honneur en faisant partie de mon jury.

Je remercie aussi les membres des laboratoires LAMSADE et LIPAH dans lesquels ce travail a vu le jour. Malgré le peu de temps que j'ai passé dans leurs locaux respectifs, j'ai pu profiter pendant mes courts séjours de de la bonne ambiance qui y règne ainsi que de l'assistance et la disponibilité du personnel. Merci à tous les occupants du bureau C605 et particulièrement Mohamed Khalil LABIDI. Je remercie Mohammad Amin FARVARDIN et Olivier ROUYER spécialement pour leur aide avec Spark. Sans oublier tout le personnel administratif pour les tracasseries que j'ai pu leur causer.

J'exprime ma profonde gratitude à Adel TOMBARI, Mansour MRABET, Ali HAJJI et leurs familles qui m'ont accueilli durant mes séjours à Paris.

Je remercie du fond de mon cœur ma femme Monia, qui grâce à l'amour qu'elle me porte, m'a poussé à aller jusqu'au bout de ces travaux, et je lui souhaite à elle aussi plein de succès dans sa carrière.

Enfin, je remercie chaleureusement tous les membres de ma famille, pour leur amour, leur soutien et leur réconfort.

Table des matières

Table des figures **iv**

Liste des tableaux **vii**

Liste des abréviations **xi**

Introduction générale **1**
Contexte et problématique 1
Objectifs et contributions 4
Organisation de la thèse 6
Liste des publications 6

1 La Programmation Génétique **9**
1.1 Introduction 10
1.2 Les Algorithmes Évolutionnaires 10
1.2.1 Boucle évolutionnaire 11
1.2.2 Terminologie 13
1.2.3 Exploration et exploitation avec EA 13
1.2.4 Types d'EA 14
1.2.5 Représentations et opérateurs génétiques 16
1.2.6 Le Darwinisme Artificiel 22
1.3 Concepts de base de la Programmation Génétique 25
1.3.1 La représentation fonctionnelle 25
1.3.2 Création de la population initiale 26
1.3.3 Croisement 28
1.3.4 Mutations 29
1.4 Résolution par GP 30
1.4.1 L'ensemble des terminaux et l'ensemble des fonctions 31
1.4.2 La fonction de fitness 33
1.4.3 Paramétrage de la GP 35
1.4.4 Le problème du coût de calcul 36
1.5 Variantes de GP 37
1.5.1 GP basée sur les graphes (GGP) 37
1.5.2 GP linéaire (LGP) 42

1.5.3 GP basée sur les piles (SGP) . . . 44
1.6 Applications de la GP . . . 45
1.6.1 Domaines d'application . . . 45
1.6.2 Les « Humies» . . . 46
1.6.3 Art évolutionnaire . . . 47
1.7 Conclusion . . . 47

2 GP pour l'apprentissage automatique 49
2.1 Introduction . . . 50
2.2 Apprentissage automatique . . . 50
2.2.1 Définitions . . . 50
2.2.2 Apprentissage supervisé . . . 51
2.2.3 Apprentissage non-supervisé . . . 52
2.2.4 Apprentissage semi-supervisé . . . 52
2.2.5 Apprentissage par renforcement . . . 52
2.2.6 Apprentissage actif (active learning) . . . 52
2.2.7 Problème de sur-apprentissage . . . 53
2.3 Adéquation de GP pour la classification supervisée . . . 55
2.3.1 Problème de classification . . . 55
2.3.2 Approches d'apprentissage automatique par GP . . . 55
2.3.3 Application de la GP pour un problème de classification . . . 56
2.4 Apprentissage par GP à partir du Big Data . . . 60
2.4.1 Défis de l'apprentissage à partir du Big Data . . . 60
2.4.2 Approches pour remédier au coût d'apprentissage . . . 61
2.4.3 Mise à l'échelle par la manipulation des données, traitements et algorithmes . . . 62
2.4.4 Mise à l'échelle par le paradigme d'apprentissage . . . 63
2.4.5 Approches étudiées . . . 63
2.5 Conclusion . . . 64

3 Échantillonnage pour la GP 65
3.1 Introduction . . . 65
3.2 Échantillonnage dans la boucle GP . . . 66
3.3 Taxonomie . . . 67
3.3.1 Fréquence d'échantillonnage . . . 68
3.3.2 Stratégie d'échantillonnage . . . 68
3.3.3 Quantité d'échantillonnage . . . 68
3.3.4 Classification des approches d'échantillonnage . . . 69
3.4 Les approches de l'échantillonnage . . . 70
3.4.1 Échantillonnage statique . . . 70
3.4.2 Échantillonnage actif . . . 72
3.4.3 Discussion . . . 84
3.5 Étude comparative . . . 86
3.5.1 Librairie évolutionnaire ECJ . . . 86

3.5.2 Base d'apprentissage . . . 86
3.5.3 Configuration GP . . . 88
3.5.4 Implémentation des algorithmes d'échantillonnage . . . 88
3.6 Résultats . . . 91
3.7 Discussion . . . 93
3.8 Conclusion . . . 95

4 Extension des méthodes d'échantillonnage 97
4.1 Introduction . . . 98
4.2 Échantillonnage hiérarchique avec TBS . . . 98
4.2.1 Contexte . . . 98
4.2.2 RSS-TBS . . . 99
4.2.3 BUSS-RSS-TBS . . . 99
4.2.4 Étude expérimentale . . . 100
4.2.5 Discussion . . . 103
4.3 Échantillonnage hiérarchique pour classifier les bosons de Higgs . . . 104
4.3.1 Contexte . . . 104
4.3.2 La base Higgs et travaux antérieurs . . . 105
4.3.3 Extension de CGP avec l'échantillonnage actif . . . 106
4.3.4 Paramètres expérimentaux . . . 108
4.3.5 Résultats et discussion . . . 109
4.4 Échantillonnage adaptatif . . . 110
4.4.1 Contexte et motivations . . . 110
4.4.2 Contrôler la fréquence d'échantillonnage . . . 111
4.4.3 Étude expérimentale . . . 115
4.4.4 Résultats . . . 116
4.4.5 Discussion . . . 120
4.5 Conclusion . . . 121

5 GP sous Spark 123
5.1 Introduction . . . 123
5.2 Environnement distribué Spark . . . 124
5.2.1 MapReduce et Hadoop . . . 124
5.2.2 Spark . . . 128
5.2.3 YARN . . . 129
5.2.4 EA et environnements distribués . . . 130
5.3 DEAP sous Spark . . . 132
5.3.1 Modèle d'implémentation . . . 133
5.3.2 Échantillonnage avec Spark . . . 135
5.4 Étude expérimentale . . . 137
5.4.1 Environnement . . . 137
5.4.2 Paramètres GP . . . 138
5.5 Résultats et discussion . . . 139
5.5.1 Temps d'apprentissage . . . 139

5.5.2 Qualité de l'apprentissage 141
5.6 Conclusion 144

Conclusion générale **147**
Bilan 147
Perspectives 148

Bibliographie **151**

A Attributs de la base KDD'99 **171**

B Attributs de la base HIGGS **175**

Table des figures

1.1 Cycle d'un EA. 12
1.2 Population, chromosome et allèle. 15
1.3 Exemple d'un automate à états finis. 16
1.4 Exemple d'une solution GP en LISP. 17
1.5 Les croisements à échange de fragments 19
1.6 BLX-ϕ. 20
1.7 La mutation 1 bit. 20
1.8 Roue de loterie. 23
1.9 Stochastic Universal Sampling Selection. 23
1.10 Exemple d'arbres syntaxiques obtenus par GP. 26
1.11 Méthode *Grow*. 27
1.12 Méthode *Full*. 28
1.13 Exemple de croisement de deux arbres. 28
1.14 Exemple de la mutation par insertion d'un sous-arbre. 29
1.15 Exemple de la mutation par promotion d'un arbre. 30
1.16 Exemple de mutation d'un arbre par remplacement d'un nœud. . . 30
1.17 Exemple de mutation d'une branche d'un arbre. 30
1.18 Exemple d'interprétation d'un arbre. 34
1.19 Représentations en graphe (a) et arbre (b). 38
1.20 Structure de base d'un programme PADO. 39
1.21 Graphe général de CGP. 40
1.22 Génotype d'un programme CGP. 41
1.23 Génotype et phénotype d'un programme CGP. 41
1.24 Représentation typique d'un programme en LGP. 42
1.25 Exemple de programme LGP avec la notation C. 43
1.26 Croisement dans LGP. 44
1.27 Antenne pour la mission ST5 de la NASA. 48
1.28 Rendu artistique avec CGP. 48

2.1 Sur-apprentissage et sous-apprentissage. 54
2.2 Classifieur obtenu après 10 générations. 58
2.3 Classifieur obtenu après 30 itérations. 59
2.4 Manipulations et phases d'analyse de données. 62

3.1 Évaluation dans un problème de classification. 67
3.2 Échantillonnage statique et actif. 69
3.3 Taxonomie des approches d'échantillonnage pour GP. 71
3.4 Exemple de croisement de données dans IDI. 77
3.5 Échantillonnage Hiérarchique. 82
3.6 Mesures moyennes sur l'ensemble d'apprentissage. 92
3.7 Mesures du meilleur individu sur la base de test. 92

4.1 Échantillonnage hiérarchique avec TBS. 101
4.2 Mesures moyennes sur la base de test. 103
4.3 Mesures du meilleur individu sur la base de test. 104
4.4 Échantillonnage actif et adaptatif. 113
4.5 Variation du temps d'apprentissage moyen selon la fréquence de ré-échantillonnage. 117
4.6 Variation de l'accuracy du meilleur individu selon la fréquence de ré-échantillonnage. 118
4.7 Variation du FPR du meilleur individu selon la fréquence de ré-échantillonnage 118
4.8 Variation de temps d'apprentissage moyen selon la stratégie de contrôle de la fréquence de ré-échantillonnage. 119
4.9 Variation de l'accuracy et FPR selon la stratégie de contrôle de la fréquence de ré-échantillonnage. 120

5.1 Exemple MapReduce 125
5.2 Modèle d'exécution de MapReduce sur un cluster 126
5.3 MapReduce vs Spark. 128
5.4 Vue d'ensemble d'un cluster Spark. 129
5.5 Modèle d'exécution de Spark avec YARN 130
5.6 DEAP avec SCOOP. 132
5.7 Redéfinition de la fonction *evaluate* avec une base d'apprentissage parallélisée. 134
5.8 Réécriture de *map* pour DEAP avec une population parallélisée. . . . 134
5.9 Évaluation parallèle de population avec DEAP sous Spark. 135
5.10 Diagramme global de l'évaluation modifiée. 136
5.11 Échantillonnage avec Spark. 137
5.12 Temps d'apprentissage moyen à partir de la base Higgs sous Spark. 139
5.13 Temps moyen d'apprentissage pour RSS (10000 instances). 140
5.14 Temps moyen d'apprentissage pour RSS (10000 instances) avec variation de la taille de la population et du nombre de générations. . . 141
5.15 Effet de la taille de l'échantillon RSS sur le temps. 142
5.16 Temps moyen d'apprentissage RSS contre FSS. 142
5.17 Performance du meilleur individu obtenu par FSS. 143
5.18 Taux de détection de RSS contre FSS. 143
5.19 Effet de la taille de l'échantillon RSS sur la qualité de l'apprentissage.144

Liste des tableaux

1.1 Exemple d'un jeu d'instructions LGP. 43
1.2 Exemple d'un programme SGP. 44
1.3 Quelques succès de GP dans les « Humies». 47

2.1 Matrice de confusion. 56

3.1 Nombre et tailles des échantillons pour les méthodes balancées. . . . 80
3.2 Classification des méthodes d'échantillonnage selon leurs catégories et décisions conceptuelles. 84
3.3 Bases d'apprentissage du KDD-99. 87
3.4 Ensembles de terminaux et fonctions. 88
3.5 Paramètres CGP. 89
3.6 Paramètres spécifiques des algorithmes d'échantillonnage. 90
3.7 Mesures du temps (en secondes). 93

4.1 Paramètres CGP. 101
4.2 Paramètres des méthodes d'échantillonnage. 102
4.3 Mesures du temps (en secondes). 103
4.4 Base d'apprentissage Higgs. 106
4.5 Paramètres CGP. 108
4.6 Paramètres d'échantillonnage. 108
4.7 Mesures de performance sur la base de test. 109
4.8 Résultats de Shashidhara et al. 110
4.9 Paramètres des algorithmes d'échantillonnage. 116

5.1 Comparaison de frameworks Big Data. 127
5.2 Configuration matérielle d'un nœud du cluster. 138
5.3 Paramètres GP. 138
5.4 Temps d'exécution avec Spark en secondes. 140

A.1 Description des attributs KDD'99. 171

B.1 Description des attributs de HIGGS. 175

Liste des Algorithmes

1 Interprétation d'un programme génétique 34
2 RSS 73
3 DSS 75
4 IDI 76
5 Mise à jour du graphe TBS 78
6 Sélection TBS 78
7 Calcul du seuil TBS 79
8 Échantillonnage Hiérarchique 83
9 Calcul du seuil modifié 91
10 Construction des blocs de niveau 0 99
11 RSS-TBS 100
12 CGP + RSS-DSS 107
13 Échantillonnage à fréquence fixe. 114
14 Fréquence d'échantillonnage déterministe. 114
15 Fréquence d'échantillonnage adaptative 115

Liste des abréviations

AM Application Master

AMS Approximate Median Significance

ATLAS A Toroidal LHC ApparatuS

BB-DSS Balanced Block DSS

BOSS Basic Over-Sampling Selection

BRSS Balanced RSS

BUSS Basic Under-Sampling Selection

CGP Cartesian GP

DEAP Distributed Evolutionary Algorithm in Python

DNN Deep Neural Networks

DSS Dynamic Subset Selection

EA Evolutionary Algorithms

ECJ Evolutionary Computation in Java

EP Evolutionnary Programming

ES Evolution Strategies

FPR False Positive Rate

FRS Fixed Random Selection

FSS Full Subset Selection

GA Genetic Algorithms

GE Grammatical Evolution

GFS Google File System

GGGP Grammar Guided GP

GGP Graph based GP

GP Genetic Programming

GPGPU General Purpose Graphics Processing Units

HDFS Hadoop Distributed File System

HSS Historical Subset Selection

IDC International Data Corporation

IDI Incremental Data Inheritance

IRS Incremental Radom Selection

IS Interleaved Sampling

JVM Java Virtual Machine

LGP Linear GP

LHC Large Hardon Collider

PADO Parallel Algorithm Discovery and Orchestration

PDGP Parallel Distributed GP

RAT Rational Allocation of Trials

RDD Resilient Distributed Datasets

RSS Random Subset Selection

RWS Roulette Wheel Selection

SBS Static Balanced Sampling

SCOOP Scalable COncurrent Operations in Python

SGP Stack based GP

SS Stochastic Sampling

SUS Stochastic Universal Sampling

SVM Support Vector Machine

TBS Topology Based Sampling

TGP Tree based GP

TPR True Positive Rate

YARN Yet Another Ressource Negociator

Introduction générale

Sommaire

Contexte et problématique 1
Objectifs et contributions 4
Organisation de la thèse 6
Liste des publications 6

Contexte et problématique

Depuis quelques années, avec l'avènement du phénomène **Big Data**[1], les méthodes d'extraction de connaissances basées sur l'apprentissage automatique doivent faire face à de nouveaux besoins liés au développement du web et à l'explosion des masses de données. Plusieurs définitions du Big Data ont été élaborées en s'appuyant sur un ensemble d'attributs ou dimensions (connus sous le nom les 3V, 5V, 6V, . . .) [Wu *et al.* 2016, Mittal *et al.* 2018]. Le groupe Gartner a préconisé une définition du Big Data autour de 3V [Laney 2001] à savoir :

- **V**olume : qui qualifie l'immense quantité de données générées par les machines, les réseaux, les interactions des utilisateurs . . .
- **V**élocité : qui caractérise la vitesse de génération des données et de leur transfert (flux de données en continu et en temps réel).
- **V**ariété : qui concerne les différents formats, types et sources de données qu'elles soient structurées, semi-structurées ou non structurées.

IBM[2] a introduit, dans un premier temps, un 4ème attribut qui est la **V**éracité qui traduit la qualité des données en terme d'exactitude. Puis, l'attribut **V**aleur est aussi ajouté aux 4V pour mettre l'accent sur la capacité de transformer les données collectées en valeur à travers leur analyse. En outre, Microsoft donne une autre définition autour de 6V [Wu *et al.* 2016] qui reprend les 3V du Gartner et y ajoute

1. Terme introduit officiellement en 2013 dans le dictionnaire *Oxford English Dictionary* (Source : https://www.kdnuggets.com/2017/02/origins-big-data.html)
2. Source : https://www.ibmbigdatahub.com/infographic/extracting-business-value-4-vs-big-data

Véracité, **V**ariabilité et **V**isibilité. La variabilité reflète la complexité des données au sens du nombre de variables. La visibilité exprime la nécessité de mettre ensemble les données provenant des différentes sources pour qu'elles soient visibles à tout processus d'analyse.

Ces différents attributs ont fait émerger de nouvelles exigences quant aux outils et techniques utilisés pour exploiter ces données notamment par les approches d'apprentissage automatique. Parmi ces approches nous pouvons citer les méta-heuristiques, comme les algorithmes évolutionnaires, connues par leur capacité à traiter plusieurs types de problème.

La Programmation Génétique (GP) [Koza 1992] est une variante des algorithmes évolutionnaires et est au cœur de notre travail. Elle est basée sur la représentation arborescente, dans sa version standard, mais aussi linéaire (Linear GP) ou celle de graphes (Cartesian GP). La GP est considérée comme un solveur universel grâce à une grande flexibilité lui permettant de modéliser les problèmes les plus complexes. La GP a l'avantage de fournir des modèles interprétables par rapport aux autres approches classiques créant des modèles en boîte noire (black-box model).

La GP, depuis son introduction et sous ses différentes variations, a réalisé plusieurs succès dans la résolution de problèmes réels de différentes catégories. Parmi les récentes applications de GP, nous pouvons citer : la génération et la correction automatique de programmes [Petke *et al.* 2014, Barr *et al.* 2015, Yang *et al.* 2018], la conception de circuits numériques [Vasícek & Sekanina 2015], détection d'anomalies ou de fraudes [Cao *et al.* 2016, Yoo *et al.* 2017], aide au diagnostic et traitement médical [Ain *et al.* 2017, Lones *et al.* 2017, Kumari *et al.* 2018], traitement d'images [Khan *et al.* 2019], finance et secteur bancaire [Gite *et al.* 2017, Kotecha & Garg 2016], robots autonomes [Kilyen & Letia 2018, Clemente *et al.* 2018] . . . Plusieurs problèmes se ramènent à un cas de classification. La détection d'anomalies, par exemple, consiste à décider si des événements présentent un mal-fonctionnement ou pas. Le diagnostic médical aussi revient à détecter la présence d'une maladie.

Les résultats obtenus par la GP ont motivé notre choix de cette méta-heuristique comme outil pouvant apporter des réponses aux problèmes rencontrés dans l'ère du Big Data, et plus précisément celui de la classification par apprentissage supervisé. Cette tâche consiste à créer un modèle capable de prédire la classe (fraude ou non, maladie ou non, . . .) en se basant sur un ensemble d'exemples fournis en entrée. Ces exemples sont décrits par un ensemble d'attributs et sont de classe connue, et constituent la base d'apprentissage. Ce modèle, ou classifieur, est donc créé à travers l'examen de plusieurs exemples de chacune des classes du problème étudié.

Comme tous les algorithmes évolutionnaires, la GP commence par une population de solutions (individus) générée de manière aléatoire sur laquelle sera appliquée la boucle évolutionnaire. Cette dernière consiste à évaluer les différentes solutions selon une fonction de mesure de qualité (fitness) puis à appliquer les opérateurs génétiques sur les individus sélectionnés jusqu'à la satisfaction

d'une condition d'arrêt pouvant être une solution satisfaisante ou un nombre de générations (itérations) cible.

Dans le cas du problème de classification, chaque individu est un classifieur. Un individu est représenté par une expression construite à partir d'une liste d'opérations prédéfinies, dont les opérandes sont puisés dans la base d'apprentissage. La classe est prédite selon la valeur du résultat de l'expression. Pour évaluer la performance d'un classifieur, cette expression est évaluée pour chaque exemple de la base d'apprentissage.

Cette phase d'évaluation représente le talon d'Achille pour la GP surtout avec les problèmes d'apprentissage à partir de données de très grande taille. Le temps consommé par cette phase n'est autre que le produit du nombre d'individus de la population par la complexité de chaque individu multiplié par le nombre d'enregistrements dans la base d'apprentissage multiplié aussi par le nombre de générations. Le volume est l'un des attributs communs entre les différentes définitions du Big Data. Il fait référence à la taille ou quantité croissante des données à manipuler dont une conséquence directe est l'accroissement du temps requis pour les traiter. Afin de répondre à cette contrainte, les approches d'apprentissage automatique existantes ont besoin d'être adaptées. Les techniques les plus utilisées pour traiter les grands volumes de données sont l'échantillonnage des données et la parallélisation des traitements.

Plusieurs techniques d'échantillonnage statique et dynamique ont été appliquées avec GP. Cependant, elles ont été étudiées dans des contextes différents (problème d'application, objectif de l'échantillonnage) et généralement avec des bases d'apprentissage de petite taille par rapport aux nouvelles bases. Ce thème est l'objet du premier volet de notre travail. Il est consacré à l'étude de plusieurs approches d'échantillonnage pour ensuite proposer de nouvelles méthodes.

Par ailleurs, de nouvelles technologies autour du Big Data ont vu le jour, nous les retrouvons dans « le paysage du Big Data et de l'Intelligence Artificielle »[3]. Elles offrent plusieurs solutions aux différents défis posés par le Big Data.

Ainsi, MapReduce [Dean & Ghemawat 2004], est un nouveau modèle de traitement des données qui convient au traitement par lot. Implémenté dans l'écosystème Hadoop, il propose une solution pour la distribution des traitements sur de grands volumes de données. Citons également Spark qui apporte un nouveau framework assez performant pour les traitements itératifs ayant un moteur d'exécution opérant en mémoire (*in-memory processing*). La GP, comme les algorithmes évolutionnaires, est de nature hautement itérative et s'intègre parfaitement aux applications possibles de Spark. C'est le deuxième volet de cette thèse dans lequel nous proposons une parallélisation de GP sous Spark.

Dans cette thèse, nous étudions donc l'extension de la GP pour profiter de son potentiel dans le nouveau contexte du Big Data, sans déployer d'importantes ressources, via d'une part l'échantillonnage des données et d'autre part la parallé-

3. Voir *Great Power, Great Responsibility : The 2018 Big Data & AI Landscape* : http://mattturck.com/bigdata2018/

lisation des évaluations en répondant aux interrogations :

- Quelles techniques d'échantillonnage sont utilisées avec GP ?
- Quel type d'échantillonnage convient à ce nouveau contexte ?
- Comment adapter ces techniques pour ce nouveau contexte ?
- Comment adapter GP pour profiter des nouveaux outils et frameworks du paysage Big Data ?

Objectifs et contributions

Dans ce contexte, l'objet de cette thèse est d'apporter des améliorations et des modifications à la GP de base afin de rendre possible son application aux problèmes d'apprentissage à partir de larges bases de données tout en préservant la qualité de l'apprentissage.

Dans un premier volet, nous nous intéressons à l'échantillonnage afin de diminuer la taille de la base d'apprentissage. L'idée est d'introduire des composantes dans l'algorithme lui permettant de sélectionner les données de manière adaptative à l'évolution de l'apprentissage. Nous commençons, alors, par étudier l'échantillonnage de la base d'apprentissage en sous-ensembles de taille réduite ou échantillons. Un tour d'horizon des publications scientifiques dans le domaine révèle l'existence de deux approches d'échantillonnage : l'échantillonnage statique, où un seul échantillon représentatif est utilisé pour tout le processus d'apprentissage, et l'échantillonnage dynamique, où l'échantillon est modifié régulièrement en sélectionnant des nouvelles lignes de données d'une manière aléatoire ou proportionnellement à leurs âges et leurs degrés de difficulté (difficulté de résolution).

L'échantillonnage statique aboutit souvent à une solution locale, spécialement si l'échantillon utilisé ne représente pas toutes les catégories ou classes des données de la base initiale. C'est pour remédier à ce problème que l'échantillonnage dynamique a été proposé. Cependant, ce dernier présente aussi des inconvénients : comme les données sont remplacées à toutes les générations, cela ne laisse pas assez de temps à la population de s'adapter à l'échantillon en cours et de résoudre tous les cas présents. Nous proposons une nouvelle approche appelée « échantillonnage adaptatif » pour remédier à ces inconvénients. Son principe est de donner la possibilité à la GP de décider quand effectuer une nouvelle sélection des données et des cas à remplacer. Il doit prendre en compte la performance de la population et l'historique de l'apprentissage effectué.

Le second volet concerne l'implémentation de GP dans un écosystème distribué en exploitant les librairies existantes. Cette implémentation nécessite l'examen de la distribution des cas d'apprentissage et aussi des individus de la population.

Les travaux menés dans le cadre de cette thèse ont abouti aux contributions suivantes :

- Une revue étendue des méthodes d'échantillonnage pour les base d'apprentissage de grande taille : l'examen minutieux des travaux dans ce domaine

nous a permis de proposer une taxonomie des méthodes d'échantillonnage des bases d'apprentissage et d'établir des critère et choix conceptuels. Ces derniers ont guidé la sélection d'un ensemble d'algorithmes à appliquer sur un problème de classification par GP sur une grande base de données (problème de détection d'intrusion KDD'99). Une étude expérimentale a aussi été menée pour comparer ces techniques [Hmida *et al.* 2016b].

– L'application de l'échantillonnage hiérarchique comme un moyen pour résoudre le problème de temps de calcul engendré par le grand volume de données à traiter par la GP. Ceci est validé sur deux plans :

  ✓ L'extension d'une méthode d'échantillonnage dynamique qui ne peut pas traiter de larges bases de données (selon les résultats de l'étude comparative). Nous avons proposé, pour étendre la méthode TBS (Topology Based Selection), de la combiner avec RSS et BUSS dans deux configurations appelées RSS-TBS et BUSS-RSS-TBS [Hmida *et al.* 2016a].

  ✓ L'application de la GP cartésienne pour traiter un problème de classification Big Data, celui de la détection des bosons de Higgs (dans [Hmida *et al.* 2018]).

– La proposition d'une nouvelle catégorie d'échantillonnage qui est l'échantillonnage adaptatif. C'est une méthode qui offre à GP des mécanismes de contrôle de la génération de nouveaux échantillons en surveillant l'évolution de l'apprentissage. L'objectif de notre contribution est d'adapter l'échantillonnage en fréquence (nombre d'échantillons), quantité (taille des échantillons) et stratégie (méthode de sélection des exemples d'apprentissage) au processus d'apprentissage. L'expérimentation est faite sur le contrôle de la fréquence d'échantillonnage selon trois schémas à savoir : fixe, déterministe et adaptatif. Dans le premier schéma, la fréquence de l'échantillonnage est fixée avant le run et demeure inchangée. Le second se base sur une fonction qui détermine les différentes valeurs à prendre tout au long du run. Enfin, le schéma adaptatif qui s'appuie sur un indicateur pour décider de changer d'échantillon [Hmida *et al.* 2019b].

– L'adaptation d'une implémentation existante de GP dans la librairie DEAP pour le contexte de Spark afin de profiter du moteur d'exécution distribué en se basant sur la distribution de la base d'apprentissage. Nous nous appuyons sur les RDD (Resilient Distributed Datasets) de Spark et la réécriture de l'évaluation. Cette réécriture optimise l'utilisation des RDD en limitant le nombre de phase Reduce par génération GP. La nouvelle implémentation est aussi enrichie par l'échantillonnage aléatoire avec une étude de l'effet de la variation de la taille de la population ainsi que la taille de l'échantillon sur le temps et la qualité de l'apprentissage [Hmida *et al.* 2019a].

Organisation de la thèse

Le reste de ce manuscrit est structuré en 5 chapitres et une conclusion.

Le chapitre 1 présente les fondements théoriques des domaines concernés par cette thèse. Il introduit les algorithmes évolutionnaires d'une manière générale ainsi que les principes du darwinisme artificiel. Ensuite, il se focalise sur la GP, ses variantes et ses applications pour montrer son potentiel ainsi que ses limites.

Le chapitre 2 introduit l'apprentissage automatique et montre comment la GP est adaptée à ce domaine. Puis, nous évoquons les principaux défis à confronter dans le contexte Big Data, plus particulièrement avec GP, ainsi que les approches leur apportant des réponses.

Le chapitre 3 dresse une revue assez étendue des différents algorithmes d'échantillonnage de la base d'apprentissage utilisés avec GP. Une taxonomie de ces algorithmes est aussi proposée. Ensuite une étude expérimentale est conduite pour comparer les performances de GP avec les algorithmes d'échantillonnage sur un problème à large base de données qui est celui de la détection d'intrusion KDD'99.

Le chapitre 4 se base sur les résultats du chapitre 3, pour introduire nos contributions. Il s'agit d'une part d'adapter les méthodes d'échantillonnage qui posent un problème de temps face aux larges bases de données et d'autre part de permettre l'application de la GP sur un problème Big Data (Higgs). Notre contribution intitulée l'*échantillonnage adaptatif* est présentée d'un point de vue théorique puis expérimental.

Le chapitre 5 expose le modèle utilisé pour adapter une implémentation existante de GP et la déployer au-dessus du moteur d'exécution distribué Spark. Puis, l'échantillonnage est aussi introduit dans ce modèle. Les résultats d'une étude expérimentale sont également présentés et discutés.

Enfin, la conclusion générale donne une synthèse des travaux et contributions effectués durant cette thèse ainsi que l'ouverture de nouvelles voies de recherche à investiguer.

Liste des publications

[1] Hmida Hmida, Sana Ben Hamida, Amel Borgi et Marta Rukoz. *A new adaptive sampling approach for Genetic Programming*. In Proceedings of the 3rd International Conference on Intelligent Computing in Data Sciences, ICDS 2019, October 28-30, 2019, Marrakech, Morocco, 2019. IEEE. (accepté)

[2] Hmida Hmida, Sana Ben Hamida, Amel Borgi et Marta Rukoz. *Genetic Programming over Spark for Higgs Boson Classification*. In Witold Abramowicz et Rafael Corchuelo, editeurs, Business Information Systems - 22nd International Conference, BIS 2019, Seville, Spain, June 26-28, 2019, Proceedings, Part I, volume 353 of *Lecture Notes in Business Information Processing*, pages 300–312. Springer, 2019 (Classe B)

[3] Hmida Hmida, Sana Ben Hamida, Amel Borgi et Marta Rukoz. *Scale Genetic Programming for large Data Sets: Case of Higgs Bosons Classification*. In Robert J. Howlett *et al.*, editeurs, Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference KES-2018, Belgrade, Serbia, 3-5 September 2018., volume 126 of *Procedia Computer Science*, pages 302–311. Elsevier, 2018 (Classe B)

[4] Hmida Hmida, Sana Ben Hamida, Amel Borgi et Marta Rukoz. *Hierarchical Data Topology Based Selection for Large Scale Learning*. In 2016 Intl IEEE Conference on Cloud and Big Data Computing, Toulouse, France, July 18-21, 2016, pages 1221–1226. IEEE, 2016

[5] Hmida Hmida, Sana Ben Hamida, Amel Borgi et Marta Rukoz. *Sampling Methods in Genetic Programming Learners from Large Datasets: A Comparative Study*. In Plamen Angelov *et al.*, editeurs, Advances in Big Data - Proceedings of the 2nd INNS Conference on Big Data, October 23-25, 2016, Thessaloniki, Greece, volume 529 of *Advances in Intelligent Systems and Computing*, pages 50–60, 2016

CHAPITRE 1

La Programmation Génétique

Sommaire

1.1 Introduction 10
1.2 Les Algorithmes Évolutionnaires 10
1.2.1 Boucle évolutionnaire 11
1.2.2 Terminologie 13
1.2.3 Exploration et exploitation avec EA 13
1.2.4 Types d'EA 14
1.2.5 Représentations et opérateurs génétiques 16
1.2.6 Le Darwinisme Artificiel 22
1.3 Concepts de base de la Programmation Génétique 25
1.3.1 La représentation fonctionnelle 25
1.3.2 Création de la population initiale 26
1.3.3 Croisement 28
1.3.4 Mutations 29
1.4 Résolution par GP 30
1.4.1 L'ensemble des terminaux et l'ensemble des fonctions 31
1.4.2 La fonction de fitness 33
1.4.3 Paramétrage de la GP 35
1.4.4 Le problème du coût de calcul 36
1.5 Variantes de GP 37
1.5.1 GP basée sur les graphes (GGP) 37
1.5.2 GP linéaire (LGP) 42
1.5.3 GP basée sur les piles (SGP) 44
1.6 Applications de la GP 45
1.6.1 Domaines d'application 45
1.6.2 Les « Humies» 46
1.6.3 Art évolutionnaire 47

1.7 Conclusion 47

1.1 Introduction

La Programmation Génétique (GP) proposée par John Koza dans [Koza 1992] est un système d'inférence automatisée de programmes qui s'améliorent de manière itérative, aussi appelée *automatic programming*. Un programme est une expression structurée composée d'un nombre variable de symboles, qui doit pouvoir être interprétée ou compilée puis exécutée par une machine.

La résolution automatisée de problèmes est au centre des concepts d'intelligence artificielle, d'apprentissage automatique, c'est ce que Turing appelle « *machine intelligence* » (traduit en intelligence artificielle). Dans [Samuel 1983], Samuel énonce l'objectif de l'intelligence artificielle :

> « to get machines to exhibit behaviour, which if done by humans, would be assumed to involve the use of intelligence. »

C'est dans ce cadre que s'inscrit la Programmation Génétique comme une métaheuristique de résolution de problèmes indépendante du problème d'application. Elle accède à cette résolution pour ces problèmes à partir de l'énoncé de haut niveau de l'objectif à atteindre.

La suite de ce chapitre commence par une introduction de la famille à laquelle appartient la GP qui est les algorithmes évolutionnaires (section 1.2), puis présente les concepts de base de la GP allant de la représentation standard sous-jacente à la description des opérations (section 1.3) jusqu'à la formulation d'un problème en GP (section 1.4) ainsi que les problèmes de choix des paramètres GP et celui des performances de calcul de la GP. La section 1.5 présente les variantes de GP les plus connues pour poursuivre, dans la section 1.6, avec les applications les plus marquantes de la GP.

1.2 Les Algorithmes Évolutionnaires

Cette section introduit les algorithmes évolutionnaires [Yu & Gen 2010, Simon 2013, Pétrowski & Ben-Hamida 2017] (*Evolutionary Algorithms :* **EA**) qui sont la famille des algorithmes qui ont puisé leurs techniques dans la théorie de l'évolution naturelle.

Les EA sont introduits par Fogel en 1965 [Fogel *et al.* 1965], Rechenberg en 1973 [Rechenberg 1973], Holland en 1975 [Holland 1975], et popularisé par Goldberg en 1989 [Goldberg 1989].

Les EA font partie des algorithmes bio-inspirés ou inspirés de la nature dans la mesure où ils imitent des processus naturels afin de résoudre un problème. En effet, les EA retrouvent leur origine dans la théorie de la sélection naturelle

darwinienne en reprenant le principe de « la survie du mieux adapté » pour construire des solutions par améliorations successives et progressives.

Plusieurs branches d'algorithmes évolutionnaires sont apparues. Elles se classent en 4 catégories :

- les algorithmes génétiques (*Genetic Algorithms* : GA) [Holland 1962]
- les stratégies d'évolution (*Evolution Strategies* : ES) [Rechenberg 1973]
- la programmation évolutionnaire (*Evolutionnary Programming* : EP) [Fogel *et al.* 1965, Fogel 1995]
- la programmation génétique (*Genetic Programming* : GP) [Koza 1992]

Certains auteurs, comme dans [Yu & Gen 2010, Simon 2013] ajoutent d'autres catégories à la liste précédente comme l'« intelligence en essaim » (Swarm Intelligence) et les « systèmes immunitaires artificiels » (Artificial Immune Systems). Mais cette classification n'est pas acceptée par d'autres auteurs du domaine EA vu qu'elles ne sont pas basées sur la sélection naturelle.

Le principal avantage des EA, est qu'ils ne posent aucune hypothèse sur le problème à résoudre ce qui a permis leur application à plusieurs problèmes relevant notamment de l'optimisation et de l'apprentissage automatique.

Les caractéristiques communes aux différentes classes d'EA peuvent être regroupées aux points suivants :

- Métaheuristique[1] : ces algorithmes consistent à guider des heuristiques[2] dans l'exploration et l'exploitation de l'espace des solutions pour trouver de façon efficace une solution proche de l'optimal [Osman & Laporte 1996]. Ils sont paramétrables pour résoudre divers types de problèmes.
- Á base de population : un EA améliore ou fait évoluer un groupe de solutions potentielles de manière simultanée appelé **population**. Chaque solution est appelée **individu**.
- Guidées par une fitness : « la survie du mieux adapté » nécessite une évaluation ou quantification du degré d'adaptation de chaque individu (la fonction de fitness). C'est le calcul de cette valeur qui va guider l'évolution de la population.
- Pilotées par les mécanismes de variation : l'amélioration itérative des solutions est opérée en appliquant différentes opérations de variation permettant d'explorer l'espace de recherche. Ces opérations simulent les phénomènes génétiques naturels : mutation, croisement, etc.

1.2.1 Boucle évolutionnaire

Les algorithmes évolutionnaires se traduisent par la répétition d'un ensemble d'étapes qu'on appelle communément la boucle évolutionnaire. Chaque cycle (ou

1. vient des mots grecs méta (au delà) et heuriskein (trouver ou découvrir)
2. Heuristique : une technique de résolution spécifique à un problème qui est basée sur des règles intuitives ou de « bon sens » .

itération) est appelé une **génération**. L'objectif de ces itérations est de chercher une solution optimale au problème traité. La recherche est effectuée dans l'espace des états Ω. Chaque élément de Ω est un **individu**, noté I. Une **population** est donc un ensemble de N éléments (individus) de Ω : $P = (I_1, I_2, \cdots, I_N)$.

La mesure du degré d'adaptation de chaque individu constitue la **fonction de performance** (en anglais **fitness**) $F : \Omega \rightarrow R$ où R est un sous-ensemble de $\mathbb{R}$. L'objectif des EA est de faire évoluer une population P dans le but de trouver l'optimum I^*. Pour ce faire, à chaque génération t, les individus de la population P(t) sont mutés et croisés avec des probabilités prédéfinies respectivement p_m et p_c, et ce sont les plus aptes qui survivent pour la génération suivante $t+1$, constituant la nouvelle population $P(t+1)$. Ce processus est répété pendant un certain nombre de **générations** jusqu'à ce qu'une **condition d'arrêt** soit satisfaite.

Les étapes principales communes aux différents algorithmes d'évolution sont décrites dans la figure 1.1.

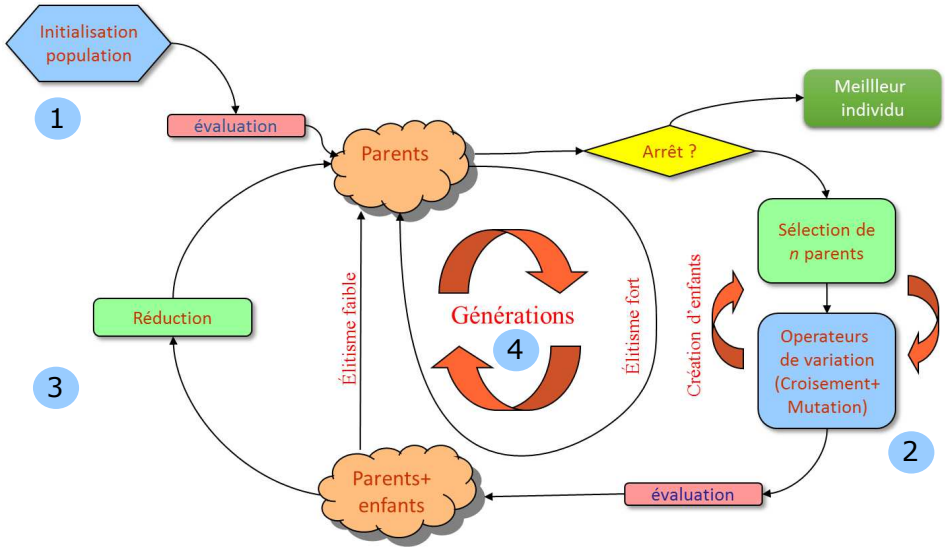


FIGURE 1.1 – Cycle d'un EA.
(Source : MOOC « Optimisation Stochastique Évolutionnaire » de Pierre Collet)

1. Le processus évolutionnaire est enclenché par la génération aléatoire d'une population initiale $P(0)$ suivie par l'évaluation de la performance de chaque individu.
2. La population précédente est auscultée pour sélectionner n parents (individus) selon leur performance. Ils sont mutés et croisés entre eux pour créer de nouveaux individus appelés enfants dont on évalue la performance.
3. La nouvelle population est constituée en sélectionnant les meilleurs parmi les enfants ou l'ensemble des parents et des enfants (selon le schéma d'évolution choisi). Cette phase de **réduction** permet de garder une taille constante de la population. Elle est aussi appelée sélection environnementale.

4. La boucle évolutionnaire réitère ensuite les étapes 2 et 3, jusqu'à ce qu'on arrive à la convergence ou à la satisfaction d'un des critères d'arrêt

1.2.2 Terminologie

Les EA n'ont pas seulement imité la sélection naturelle mais ils ont aussi emprunté son vocabulaire. Ce paragraphe énumère les différents termes utilisés en donnant leurs définitions succinctement.

Génotype/Phénotype C'est la forme que prend un individu appartenant à l'ensemble Ω, elle est utilisée lors de l'application des opérations de variation. Un génotype, composé de **gènes**, est la transcription d'un phénotype qui est une solution du problème sur lequel est définie la fitness. Par exemple, dans un problème de contrôle de robot, un génotype peut être un vecteur réel, et il représente une séquence d'actions du robot.

Codage/Décodage L'espace des phénotypes et celui des génotypes sont le plus souvent différents. Le codage est la traduction d'un phénotype en génotype. Le décodage est la fonction réciproque. L'opération de décodage est exécutée à chaque évaluation d'un individu.

Croisement Une recombinaison des génotypes de deux individus de la population pour créer un ou deux enfants qui propagent une partie du patrimoine génétique de leurs parents. Elle peut être réalisée de manière aléatoire ou déterministe.

Mutation C'est une modification aléatoire ou déterministe d'un ou plusieurs gènes d'un génotype.

Sélection C'est le tirage d'un nombre d'individus à partir de la population avec ou sans remise. Cette opération est exécutée deux fois dans la boucle évolutionnaire pour deux objectifs différents. Dans le premier cas, elle détermine les individus qui vont se reproduire par application des opérateurs génétiques, le plus souvent selon leur performance et sans remise. C'est la **sélection parentale**. Dans le second cas, il s'agit d'une sélection, souvent avec remise, pour le remplacement des parents appelée **sélection environnementale**. Elle est effectuée sur les enfants et éventuellement leurs parents. Cette dernière est indépendante de la représentation des individus. Elle confère à l'EA un caractère **élitiste** ou **générationnel**.

Élitisme Un EA élitiste garantit qu'au moins une copie du meilleur individu sera propagée vers la génération suivante.

Remplacement générationnel Un EA est dit générationnel quand toute la population est remplacée par les enfants sans préserver aucun parent.

1.2.3 Exploration et exploitation avec EA

L'exploration et l'exploitation constituent les pierres angulaires de toute méthode de résolution de problème par recherche [Eiben & Schippers 1998] et en

particulier les EA. La clé de réussite réside dans l'atteinte d'un équilibre entre ces deux notions. Dans [Crepinsek *et al.* 2013], une revue d'une centaine de travaux en EA recense les différentes méthodes pour arriver à cet équilibre.

Exploration : Elle consiste à visiter de nouveaux points de l'espace de recherche Ω et maintient ainsi la diversité. Elle est liée à la recherche globale.

Exploitation : Elles considérée comme une recherche locale en exploitant les solutions précédentes pour visiter leurs voisinages.

Ces deux mécanismes sont complémentaires : l'exploitation toute seule conduit à un optimum local tandis qu'une exploration excessive se résume à une **marche aléatoire**.

En EA, l'idée commune est que les opérations de sélection et de croisement sont des étapes d'exploitation, alors que l'initialisation et la mutation sont des étapes d'exploration. Toutefois, la mutation peut devenir un opérateur d'exploitation si les perturbations générées sont très petites. De plus, on peut perdre la diversité de la population au cours de la procédure de sélection des survivants pour la génération suivante.

1.2.4 Types d'EA

La famille des EA englobe plusieurs types, ayant chacun un ensemble de caractéristiques qui les distingue, et qui sont : les algorithmes génétiques, les stratégies d'évolution, la programmation évolutionnaire et la programmation génétique. Nous donnons, dans les paragraphes qui suivent, un aperçu de ces types avec un bref historique.

1.2.4.1 Les Algorithmes Génétiques : GA

C'est la catégorie la plus notoire des EA et la plus utilisée. Ils ont été développés dans les années soixante par Holland [Holland 1962] mais leurs origines remontent aux milieu des années cinquante [Barricelli 1954,Fraser 1957]. L'espace de recherche pour ces algorithmes est sous forme de chaînes binaires ($\{0,1\}^n$). Le génotype est appelé aussi **chromosome** et la valeur de chaque gène est appelée **allèle** (voir figure 1.2).

Ce n'est que pendant les années quatre-vingt-dix, précisément avec l'apparition de l'ouvrage de référence écrit par Goldberg [Goldberg 1989], que les GA se sont fait connaître dans la communauté scientifique.

1.2.4.2 Les Stratégies d'Évolution : ES

Les Stratégies d'Évolution ont été introduites à l'université technique de Berlin par 3 étudiants qui essayaient de résoudre un problème de conception aérodynamique en introduisant des modifications aléatoires au meilleur modèle [Schwefel 1965]. Leur domaine d'application de prédilection est la résolution de problèmes

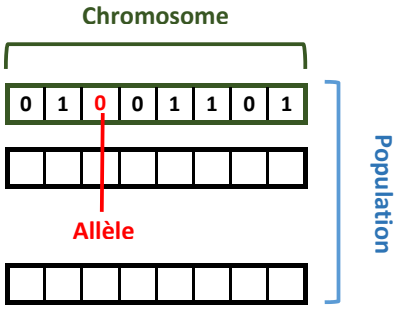


FIGURE 1.2 – Population, chromosome et allèle.

d'optimisation numérique dans l'espace des vecteurs réels. Dans les premières versions de ces algorithmes, les principales opérations sont la sélection et la mutation ; le croisement vient en second lieu.

Plusieurs schémas décrivent comment la sélection et le croisement sont réalisées et sont regroupés dans la notation mnémonique $(\mu/\rho \overset{+}{,} \lambda) - ES$ [Hansen *et al.* 2015], où μ, ρ et λ sont trois entiers positifs représentant respectivement la taille de la population (nombre de parents), le nombre de parents à croiser et le nombre d'enfants générés. Le symbole $+$ indique que la sélection est faite sur l'ensemble des parents et enfants. Avec la virgule, les parents sont écartés de la sélection. Parmi ces scénarios nous pouvons citer :

- $\boldsymbol{(1+1)-ES}$ **:** La population est composé d'un unique individu auquel est appliquée une mutation gaussienne dont la déviation standard est adaptée selon la règle $1/5$ par Rechenberg [Rechenberg 1973].
- $\boldsymbol{(\mu+1)-ES}$ **:** μ parents sont croisés entre eux pour produire un seul enfant sur lequel une mutation est appliquée. De l'ensemble de $\mu+1$ individus, le plus mauvais est supprimé.
- $\boldsymbol{(\mu+\lambda)-ES}$ **:** De μ parents, λ enfants sont générés ($\lambda \geq \mu$). Les μ meilleurs sont gardés pour la génération suivante.
- $\boldsymbol{(\mu,\lambda)-ES}$ **:** Identique au schéma précédent mais les parents sont supprimés et la sélection est appliquée aux λ enfants.
- $\boldsymbol{(\mu/\rho,\lambda)}$ **:** Identique à $(\mu,\lambda)-ES$ mais pour créer un enfant ρ parents sont croisés.

1.2.4.3 La Programmation Évolutionnaire : EP

La Programmation Évolutionnaire a été développée par L.J. Fogel [Fogel *et al.* 1965] en Californie dans les années 60, dans l'espace des automates à états finis (figure 1.3), pour la résolution de problèmes de prédiction. La table de transition des automates est modifiée grâce à des mutations aléatoires uniformes dans

l'alphabet discret correspondant. L'évaluation de la performance des individus correspond au nombre de symboles prédits correctement. Chaque automate de la population parente génère un enfant par mutation, et les meilleures solutions entre les parents et les enfants sont sélectionnées pour survivre, ce qui correspond à la stratégie $(\mu + \mu)$ dans la terminologie des ES.

Les EP ont été ensuite développées et leur domaine a été élargi par D.B. Fogel, afin qu'il puissent travailler dans l'espace réel [Fogel 1992a, Fogel 1992b], où la sélection déterministe est remplacée par un tournoi stochastique.

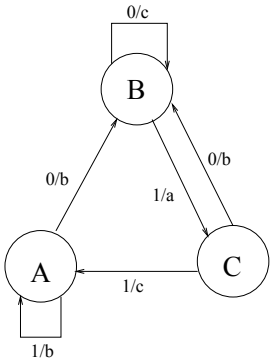


FIGURE 1.3 – Exemple d'un automate à états finis ayant trois états différents $S = \{A, B, C\}$, un alphabet d'entrée $I = \{0, 1\}$, et un alphabet de sortie $O = \{a, b, c\}$. Chaque arête entre deux états indique une transition possible, et la fonction de transition $\delta : S \times I \rightarrow S \times O$ est spécifiée par les labels au niveau des arêtes ayant la forme i/o, signifiant que $\delta((s_i, i)) = (s_j, o)$.

1.2.4.4 La Programmation Génétique : GP

C'est Cramer [Cramer 1985a] qui est le premier à avoir utilisé des structures arborescentes dans un algorithme génétique. Mais l'adoption de cette représentation pour définir la programmation génétique comme un nouvel algorithme évolutionnaire a été faite par John Koza en 1992 [Koza 1992]. Son objectif initial était de faire évoluer des sous-programmes du langage LISP (figure 1.4). Grâce à l'ouvrage de Koza [Koza 1992], l'utilisation de GP s'est étendue pour la résolution de nombreux types de problèmes dont les solutions peuvent être représentées par des structures arborescentes, comme les représentations fonctionnelles linéaires [Nordin 1994], les graphes [Teller & Veloso 1995, Ryan *et al.* 1998], les structures moléculaires, ...

L'ensemble des travaux de cette thèse s'appuie sur cette famille des algorithmes évolutionnaires. La GP est disséquée dans la suite de ce chapitre.

1.2.5 Représentations et opérateurs génétiques

Le succès d'un EA est étroitement lié au choix du génotype et de la représentation (correspondance génotype-phénotype) et aussi aux différentes opérations à

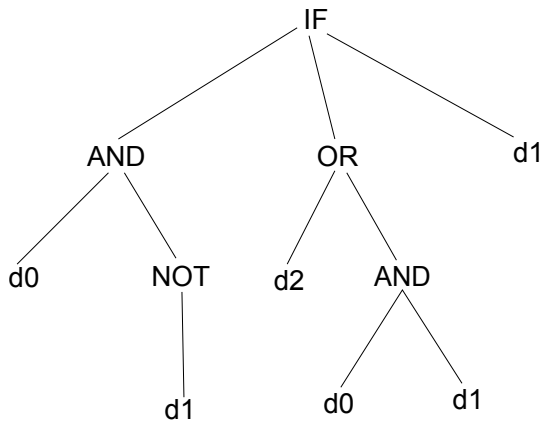


FIGURE 1.4 – Exemple d'une solution GP en LISP : $\{d0, d1, d2\}$ est un ensemble d'instructions constituant les terminaux, et $\{if, and, or\}$ sont des expressions LISP constituant les nœuds.

appliquer sur les génotypes pour générer les enfants. Le choix d'une représentation limite l'ensemble des opérations applicables.

Deux principales approches sont décrites dans [Rothlauf 2006]. La **représentation indirecte** qui est la plus générale et la plus répandue. Elle se base sur le codage et décodage vers ou depuis une structure de données standard (e.g. vecteur binaire). Et la **représentation directe** dans laquelle les solutions sont codées en utilisant des concepts du problème à résoudre. Cette dernière représentation élimine les opérations de décodage mais nécessite de définir des opérations génétiques spécifiques.

Un large nombre de représentations et d'opérations ont été définies [Back *et al.* 1997, Larrañaga *et al.* 1999, Rothlauf 2006, Simon 2013, Pawlak *et al.* 2015, Umbarkar & Sheth 2015, Pétrowski & Ben-Hamida 2017]. Nous donnons dans ce qui suit quelques représentations usuelles et des variantes des opérations de croisement et mutation.

1.2.5.1 La représentation binaire

Le codage binaire est le cadre général des GA traditionnels [Goldberg 1989]. Chaque individu I est représenté par un vecteur binaire (ou chaîne de bits), où chaque élément prend la valeur 0 ou 1 :

$$I = (a_1, \cdots, a_l) \in \{0,1\}^l,$$

où l est la taille du vecteur (nombre de bits). L'espace de recherche Ω (espace phénotypique) est alors $\{0,1\}^l$ avec 2^l génotypes différents.

Cette représentation s'adapte bien à des problèmes où les solutions potentielles ont une représentation binaire canonique, comme les problèmes booléens. Elle s'applique aussi pour des problèmes d'optimisation paramétrique continue (f :

$\mathbb{R}^n \rightarrow \mathbb{R}$, n est le nombre de paramètres), mais il est nécessaire de définir une technique de codage adéquate de $\mathbb{R}^n$ vers $\{0,1\}^l$.

1.2.5.2 La représentation entière

On peut définir cette représentation comme la représentation binaire mais les valeurs des éléments sont puisées dans un alphabet plus grand. La taille de l'espace de recherche est de Ξ^l où $\Xi > 2$ est le nombre de symboles de l'alphabet. La plupart des problèmes résolus avec cette représentation peuvent être traduits dans la représentation binaire qui est recommandée alors puisqu'elle possède un alphabet plus petit. C'est la représentation favorisée dans l'optimisation combinatoire évolutionnaire.

1.2.5.3 La représentation réelle

Avec la représentation réelle, l'espace de recherche est l'espace réel : $\Omega = \mathbb{R}^n$ (variables non bornées) ou une partie de l'espace réel : $\Omega = S$, avec $S \subset \mathbb{R}^n$ (variables bornées).

Cette représentation a été introduite initialement pour les stratégies d'évolution [Schwefel 1977], mais son utilisation s'est étendue rapidement aux autres types d'algorithmes évolutionnaires.

1.2.5.4 Le croisement

Le but du croisement est de combiner des propriétés intéressantes pour la résolution du problème détectées dans certains individus. Le choix de ces individus est la mission de l'opération de sélection. L'application du croisement nécessite un nombre de parents égal ou supérieur à 2 et donne naissance à 1 ou 2 enfants. Dans [Surry & Radcliffe 1996], quelques recommandations quant à la conception d'opérations d'exploration sont proposées dont la plus importante est d'utiliser une métrique. Par exemple, la règle qui stipule que la distance entre l'enfant ($x^{'}$) et ses parents(x_1 et x_2) doit être inférieure ou égale à la distance entre les parents formulée dans 1.1.

$$max(d(x^{'}, x_1), d(x^{'}, x_2)) \leq d(x_1, x_2). \tag{1.1}$$

La distance dépend de la représentation et peut être, par exemple, une distance euclidienne, de Hamming ou de Levenshtein.

Croisements à échange de fragments Ce sont les variantes de croisement les plus utilisées et applicables sur les différentes représentations. Leur rôle consiste à combiner les génotypes de deux individus pour en obtenir deux nouveaux, en échangeant un ou plusieurs fragments des deux génotypes selon un nombre de points de coupures (n points est la généralisation des croisement à 1 et 2 points, voir figure 1.5). On distingue plusieurs croisements possibles, dont les plus utilisés sont :

- *Le croisement à 1 point* [Holland 1962] : un seul fragment est échangé selon un point de coupure choisi aléatoirement,
- *Le croisement à 2 points* [Spears & De Jong 1991, De Jong & Spears 1992] : deux fragments sont échangés selon 2 points de coupure choisis aléatoirement,
- *Le croisement Uniforme* [Syswerda 1989] : on échange les fragments à chaque position indépendamment avec une probabilité de 0.5. Il peut être vu comme un croisement multi-points dont le nombre de coupures est déterminé aléatoirement au cours de l'opération.

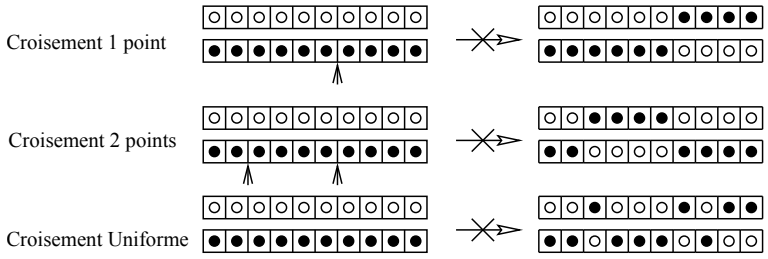


FIGURE 1.5 – Les croisements à échange de fragments : à 1 point, à 2 points et le croisement uniforme.

Croisement intermédiaire Il s'agit de combiner linéairement les deux fragments choisis à partir des parents. Cette forme n'est pas adaptée à la représentation binaire et convient à celle qui est réelle.

$$\forall i \in \{1, \cdots, n\},\ x'_i = \alpha x_{S,i} + (1-\alpha) x_{T,i}$$

où α est une variable aléatoire uniforme appartenant à l'intervalle $[0,1]$ ($\alpha = \mathcal{U}[0,1]$) et S et T sont deux individus sélectionnés pour l'ensemble des composantes de $\overrightarrow{x}$ de taille n.

Une autre variante du croisement intermédiaire est le croisement BLX-ϕ (« *Blend Crossover* »), proposé par Eshelman et Schaffer [Eshelman & Schaffer 1993], qui a la propriété de pouvoir élargir le domaine de définition de l'enfant, en utilisant un intervalle plus large pour α. En effet, α n'est plus compris entre 0 et 1, mais entre $(-\phi)$ et $(1+\phi)$, avec $0 < \phi < 1$ (figure 1.6). On note que le croisement BLX-0 correspond au croisement arithmétique traditionnel.

1.2.5.5 La mutation

L'opération de mutation est une modification aléatoire du génotype d'un individu en y introduisant une légère variation. Elle est appliquée avec une faible probabilité p_m généralement aux enfants issus d'un croisement.

La variation et la probabilité peuvent être constants ou s'adapter selon les précédentes mutations.

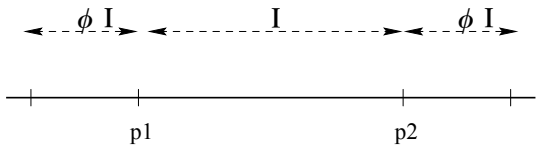


FIGURE 1.6 – BLX-ϕ.

La mutation binaire Le génotype étant constitué de bits, la mutation est simplement une inversion aléatoire de bits. Les mutations les plus utilisées sont :

- **La mutation stochastique (*bit flip*)** [Goldberg 1989, Kallel 1999] : inverse chaque bit indépendamment avec une probabilité $\frac{c}{l}$, avec $c > 0$ et l est la taille du vecteur. C'est la mutation la plus employée dans la représentation binaire.
- **La mutation 1 bit** : inverse le symbole d'un bit choisi au hasard avec une probabilité p_m [Kallel 1999] (figure 1.7).



FIGURE 1.7 – *La mutation 1 bit : Le symbole du* 5^{me} *bit a été inversé.*

La mutation réelle Une grande variété de stratégies de mutation ont été proposées pour la représentation réelle [Reed *et al.* 1967, Rechenberg 1973, Schwefel 1981, Schwefel 1987, Michalewicz 1996].

Nous présentons ci-dessous les mutations uniformes et les mutations Gaussiennes.

- **Mutation uniforme** : Des opérateurs de mutation de vecteurs réels ont été proposés par Michalewicz dans [Michalewicz 1996], dédiés à des problèmes contraints (variables bornés et/ou soumises à des contraintes linéaires). Ce sont des opérateurs unaires, qui à partir d'un parent $\overrightarrow{x} = (x_1, \cdots, x_k, \cdots, x_n)$, génèrent un enfant $\overrightarrow{x}' = (x'_1, \cdots, x'_k, \cdots, x'_n)$, où x'_k est l'élément muté et k dans l'intervalle $[1, n]$. Pour calculer x'_k, ils utilisent le domaine admissible des variables du rang k $[l(k), u(k)]$, défini à partir des bornes de l'espace de recherche et des contraintes du problème s'il y en a, à condition qu'elles soient linéaires. Avec la mutation uniforme, x'_k prend une valeur aléatoire (distribution de probabilité uniforme) dans l'intervalle $[l(k), u(k)]$.

$$r \leftarrow U[0,1]$$
$$x'_k \leftarrow \begin{cases} x_k & si \;\; r \geq p_m \\ U[l(k), u(k)] & si \;\; r < p_m \end{cases}$$

La mutation uniforme centrée en x_k : C'est une variation de la mutation uniforme. Avec cette mutation, x'_k prend une valeur selon une probabilité uniforme dans un intervalle centré sur son ancienne valeur.

$$r \leftarrow U[0,1]$$
$$x'_k \leftarrow \begin{cases} x_k & si \;\; r \geq p_m \\ U[x_k - \alpha, x_k + \alpha] & si \;\; r < p_m \end{cases}$$

Le paramètre α contrôle l'importance de la variation de la mutation tout en restant dans le domaine de recherche.

$$\alpha = min(x_k - l(k), u(k) - x_k)$$

– **Les mutations Gaussiennes** : Elles sont basées sur l'ajout d'un bruit Gaussien aux différentes composantes du vecteur $\overrightarrow{x}$. La difficulté de cette approche est l'ajustement de la déviation standard σ du bruit généré. Une grande valeur de σ favorise l'exploration de l'espace de recherche au détriment de l'exploitation des meilleures solutions et une petite valeur a un effet inverse. La valeur de σ peut être fixée de façon statique ou adaptative au cours de l'évolution, comme la règle des $1/5$ proposé par Rechenberg en 73 [Rechenberg 1973] ou la mutation Log-Normale proposée par Schwefel en 81 [Schwefel 1981].

✓ *La mutation Gaussienne statique :* Elle modifie toutes les composantes d'un individu en ajoutant un bruit Gaussien :

$$\forall i \in \{1, \cdots, n\}, \;\; x'_i = x_i + N(0, \sigma)$$

où σ est unique pour toute la population. Il est défini par l'utilisateur et sa valeur reste inchangée au long de l'évolution.

✓ *La mutation Gaussienne adaptative (la règle des 1/5) :*

Rechenberg propose de mettre à jour la valeur de σ toutes les générations. Il a défini la règle des $1/5$ [Rechenberg 1973], qui mesure la probabilité p_{sm} des mutations réussies (performance de I' est meilleure que celle de I), et ajuste σ à la génération t comme suit :

$$\sigma(t) = \begin{cases} \sigma(t-1) \times fact & si \;\; p_{sm} < 0.2 \\ \sigma(t-1)/fact & si \;\; p_{sm} > 0.2 \\ \sigma(t-1) & si \;\; p_{sm} = 0.2 \end{cases}$$

où $0 < fact < 1$ est le taux d'adaptation de σ. Selon une étude faite par Schwefel en [Schwefel 1981], une bonne valeur de $fact$ est 0.83. De plus, cette règle peut être appliquée toute les k générations au lieu de toutes les générations (k paramètre de la méthode).

1.2.6 Le Darwinisme Artificiel

Le Darwinisme dans un algorithme évolutionnaire est appelé dans deux étapes différentes au cours d'une génération donnée : l'étape de sélection parentale afin de sélectionner les parents qui vont se reproduire et l'étape de remplacement, qui remplace les individus non sélectionnés par ceux désignés à survivre, constituant ainsi la nouvelle population (figure 1.1).

La sélection est un opérateur principal utilisé dans la programmation évolutionnaire, dont le premier rôle est de favoriser les meilleurs individus dans la population. Il sélectionne les individus relativement performants pour se reproduire. Il doit aider l'algorithme à converger, et en même temps, maintenir la diversité de la population, afin d'éviter la convergence prématurée. Plusieurs stratégies ont été mises en place, dont nous présentons les plus utilisées.

1.2.6.1 La sélection parentale

La roulette La première sélection mise en place pour les GA est le tirage à la roulette, ou *Roulette Wheel Selection* (*RWS*) introduite par Goldberg [Goldberg 1989]. C'est une méthode stochastique qui exploite la métaphore d'une roulette de casino, qui compterait autant de cases que d'individus dans la population. La largeur de la case d'un individu $\overrightarrow{x_i}$ est proportionnelle à sa performance $f(\overrightarrow{x_i})$: $\frac{f(\overrightarrow{x_i})}{\sum_j^N f(\overrightarrow{x_j})}$. La roue étant lancée, l'individu sélectionné est désigné par l'arrêt de la roue sur sa case (figure 1.8).

L'espérance n_i du nombre de copies d'un élément $\overrightarrow{x_i}$ de la population courante est donnée par l'expression :

$$n_i = \frac{N}{\sum_j^N f(\overrightarrow{x_j})} f(\overrightarrow{x_i})$$

où N est le nombre d'individus. L'espérance maximale $Max(n_i)$ dans l'ensemble des éléments de la population est appelée la **pression sélective**.

Cette méthode favorise les meilleurs individus, mais tous les individus ont toujours des chances d'être sélectionnés. Cependant, elle peut causer la perte de la diversité de la population si la pression sélective (ou n_i du meilleur) est élevée. De plus, sa variance et son coût sont élevés.

Afin de diminuer la variance, Baker a proposé en 1987 la sélection à la roulette avec reste stochastique, ou *Stochastic Universal Sampling* (SUS) [Baker 1987]. Avec cette approche, d'une façon similaire à *RWS*, on considère une roulette partitionnée en autant de cases que d'individus, où chaque case est de taille proportionnelle à la performance. Mais cette fois, les individus sélectionnés sont désignés par un ensemble de points équidistants (figure 1.9). Le nombre effectif de copies de l'individu $\overrightarrow{x_i}$ sera la partie entière inférieure ou supérieure de son espérance n_i :

$$E(\frac{N.f(\overrightarrow{x_i})}{\sum_{i=j}^N f(\overrightarrow{x_j})})$$

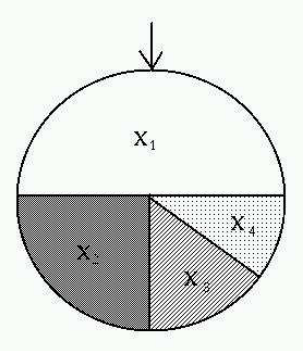


FIGURE 1.8 – Roue de loterie pour une population de 4 individus avec $f(x_1) = 50, f(x_2) = 25, f(x_3) = 15, f(x_4) = 10$. Pour tirer un élément, on lance la roue, et si elle s'arrête sur la case i, x_i est sélectionné.

Le bruit de sélection étant plus faible, la dérive génétique se manifeste moins qu'avec la méthode *RWS*.

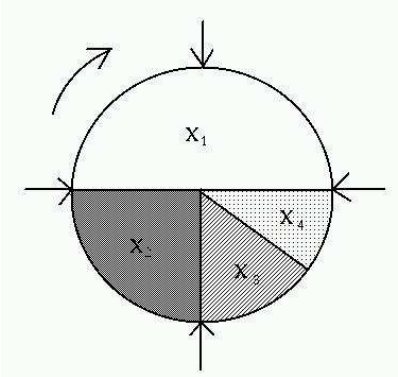


FIGURE 1.9 – Stochastic Universal Sampling Selection : pour sélectionner des individus, on lance la roue, et quand elle s'arrête, chaque individu dont la case est pointée par un marqueur est sélectionné.

Le *Ranking* Une autre variante de la sélection par roulette est la roulette par le rang, conçue également pour lutter contre les éventuels super-individus. La largeur de la case d'un individu donné est proportionnelle à son rang dans la population (triée par ordre décroissant de la performance). Les valeurs de la fitness sont alors sans importance pour la sélection, seul le classement des individus compte.

$$\text{largeur de la case i : } \frac{Rang(\overrightarrow{x_i})}{N \times (N-1)/2}.$$

Le tournoi Comme pour le *Ranking*, la sélection par tournoi ne fait pas intervenir les valeurs des fitness, mais seulement des comparaisons entre les individus. Son principe est de choisir un groupe de q individus aléatoirement dans la population, de sélectionner d'une manière déterministe le meilleur dans ce groupe, et de recommencer l'opération jusqu'à l'obtention du nombre d'individus requis. Au cours d'une génération, il y a autant de tournois que d'individus à sélectionner. La pression de sélection est ajustée par le nombre de participants à un tournoi (q). Un q élevé conduit à une forte pression de sélection.

L'avantage de cette technique de sélection est qu'elle est paramétrable par la valeur de q, peu sensible aux erreurs sur f et n'est pas chère à mettre en œuvre et à exécuter. Par contre, sa variance est élevée [De Jong & Sarma 1995].

Il existe plusieurs variantes de la sélection avec tournoi, dont on cite le tournoi de Boltzmann [de la Maza & Tidor 1993], qui assure que la distribution des valeurs d'adaptation d'une population soit proche d'une distribution de Boltzmann. Pour une compétition entre une solution courante $\overrightarrow{x_i}$ et une solution alternative $\overrightarrow{x_j}$, $\overrightarrow{x_i}$ gagne avec la probabilité $\frac{1}{1+e^{(f(\overrightarrow{x_i})-f(\overrightarrow{x_j}))/T}}$, où T est la température de sélection.

1.2.6.2 La sélection environnementale ou remplacement

Le remplacement déterministe Le remplacement déterministe est utilisé dans les stratégies d'évolution. Son caractère purement déterministe lui donne un rôle clef dans l'évolution vu qu'il guide la recherche vers les zones des meilleurs individus. Il opère en sélectionnant les μ ($1 < \mu \leq \lambda$) meilleures solutions parmi :

- l'union des μ parents et λ enfants : schéma appelé $(\mu + \lambda)$-ES,
- l'ensemble des λ enfants : schéma appelé (μ, λ)-ES.

Le remplacement $(\mu + \lambda)$ est élitiste et il garantit une amélioration monotone de la performance de la population, mais il s'adapte mal à un éventuel changement d'environnement. Par contre, avec le remplacement (μ, λ), la meilleure performance peut décroître, mais l'algorithme est plus flexible à des changements d'environnement. De plus, la régression des meilleures performances peut aider le processus de recherche à sortir des régions d'attraction des optima locaux et continuer l'exploration ailleurs. D'où la recommandation du schéma (μ, λ)-ES par Schwefel [Schwefel 1981].

Le remplacement générationnel Une autre technique de remplacement est le remplacement générationnel, utilisé principalement dans les GA standards. Avec cette approche, la population des enfants remplace entièrement la population parente. La durée de vie d'un individu est alors d'une seule génération. Dans ses études sur la performance de cette technique, De Jong [De Jong 1975] a proposé de définir un paramètre G (*Generation Gap*), spécifiant le taux de remplacement à chaque génération, afin d'augmenter la durée de vie des meilleurs individus. $G = 1$ correspond au remplacement générationnel total. Une étude empirique

faite par Grefenstette en 1986 [Grefenstette 1986] montre que des taux élevés de G donnent des meilleurs résultats que des taux faibles.

1.3 Concepts de base de la Programmation Génétique

1.3.1 La représentation fonctionnelle

La première utilisation des structures arborescentes dynamiques dans un algorithme génétique a été proposée par Cramer en 1985 [Cramer 1985b], dans le but de faire évoluer des sous-programmes séquentiels d'un langage algorithmique simple. Le moteur d'évolution utilisé était le *Steady-State Genetic Algorithm (SSGA)*, dont la tâche n'était pas de rechercher les valeurs optimales d'un problème posé, mais de découvrir le programme informatique qui pourrait le résoudre.

John Koza a adopté cette représentation en 1992 [Koza 1992] pour définir la Programmation Génétique ou *Genetic Programming* (GP) comme un nouvel algorithme évolutionnaire. Son objectif initial était de faire évoluer des sous-programmes du langage LISP (figure 1.10 (a)). Il a montré empiriquement que son approche permet de découvrir des programmes pertinents pour un grand nombre d'exemples d'applications avec une efficacité notablement plus élevée que ce que permettrait le hasard. La conception d'objets complexes comme des circuits électroniques en est un exemple. Grâce à l'ouvrage de Koza [Koza 1992], l'utilisation de la Programmation Génétique s'est étendue pour la résolution de nombreux types de problèmes dont les solutions peuvent être représentées par des structures arborescentes, comme les représentations fonctionnelles linéaires [Nordin 1994] (figure 1.10(b)), les graphes [Teller & Veloso 1995, Ryan *et al.* 1998], les structures moléculaires [Wasiewicz & Mulawka 2001],...

Dans la représentation fonctionnelle, une solution est une fonction (f), construite à partir de :

1. un ensemble de symboles terminaux ou feuilles $\mathcal{T}$ qui peuvent être des variables, des constantes universelles, des fonctions sans arguments (rand(), time(), ...).
2. un ensemble de symboles non terminaux ou nœuds ($\mathcal{N}$) qui peuvent être des opérateurs : $*, -, +$, des fonctions avec arguments : sin, cos, ...

En tant que structures d'arbres syntaxiques, les *Programmes Génétiques* requièrent la donnée de l'ensemble des nœuds et des symboles terminaux, décrivant alors l'espace des solutions possibles du problème à résoudre.

L'ensemble des nœuds et des feuilles doivent respecter les propriétés de clôture et de suffisance (voir 1.4.1). La propriété de suffisance exige que les séries de symboles terminaux et non terminaux soient capables de représenter une solution au problème posé, alors que la propriété de clôture implique que chaque nœud

doit accepter, comme argument, n'importe quels type et valeur qui peuvent être produits par un symbole terminal ou non terminal.

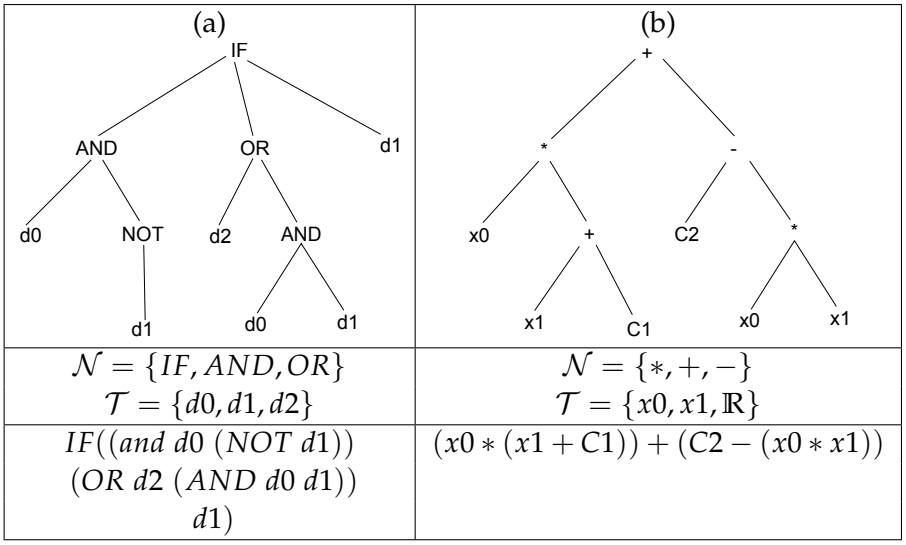


FIGURE 1.10 – Exemple d'arbres syntaxiques obtenus par Programmation Génétique (a) où l'espace exploré est l'espace des programmes LISP, et en représentation fonctionnelle linéaire (b) où l'espace exploré est celui des polynômes réels à 2 variables.

Les structures des individus avec la Programmation Génétique sont très différentes de celles qui ont été évoquées précédemment pour d'autres représentations. Les arborescences doivent notamment disposer d'un mécanisme de régulation de leurs tailles. Dans le cas contraire, elles vont avoir tendance à croître indéfiniment au cours des générations (*bloating*) [Purohit *et al.* 2013], consommant ainsi inutilement de plus en plus de mémoire et de puissance de calcul. Le mécanisme de régulation peut être simplement implanté par la donnée d'une profondeur d'arbre maximale, ou alors un nombre maximum de nœuds que les opérateurs génétiques doivent respecter.

1.3.2 Création de la population initiale

Avec la représentation arborescente, la création de la population initiale ne répond pas aux mêmes règles qu'avec les représentations binaire et réelle. Chaque arbre est construit en deux étapes : d'abord les nœuds ensuite les feuilles. Cependant, la forme de l'arbre dépend de l'approche adoptée. On dénombre principalement trois méthodes :

– « *Grow* » : les arbres sont de formes irrégulières ; à chaque étape, la sélection se fait d'une façon uniforme dans les ensembles des nœuds et des terminaux, tout en respectant la profondeur maximale (figure 1.11).

– « *Full* » : les arbres sont équilibrés et pleins ; pour un nœud donné, un terminal n'est choisi que lorsque l'on est à la profondeur maximale (figure 1.12).
– « *Ramped Half and Half* » : afin de garantir une bonne diversité de la forme et la taille des arbres, Koza [Koza 1992] a proposé de combiner les deux méthodes « Full » et « Grow » en permettant la création d'un nombre égal d'arbres de profondeurs régulièrement échelonnées qui varient entre 2 et la profondeur maximale en alternant les deux premières méthodes. C'est la méthode préférée actuellement.

Les arbres des figures 1.11 et 1.12 sont construits à partir de l'ensemble des nœuds terminaux $\mathcal{T}$ et celui des fonctions $\mathcal{N}$ suivants :

$$\mathcal{T} = \{x, y, 1, 0\},\ \mathcal{N} = \{+, -, *, /\}$$

Avec la méthode *Grow*, jusqu'à la profondeur maximale, chaque nœud est choisi parmi l'ensemble complet des éléments primitifs (fonctions et terminaux) pouvant être des nœuds ou des feuilles. Dès le franchissement de la profondeur limite, il suffit de choisir uniquement des terminaux (feuilles). La figure 1.11 illustre la construction d'un arbre de profondeur maximale 2 avec la méthode *Grow*. Un terminal a été tiré comme premier argument de la racine + à l'étape t=2, ce qui interrompt le processus de construction de la branche avant la profondeur maximale. L'autre argument est une fonction soustraction - dont les arguments sont obligatoirement tirés parmi les terminaux (1 et y), car la profondeur maximale de 2 est atteinte.

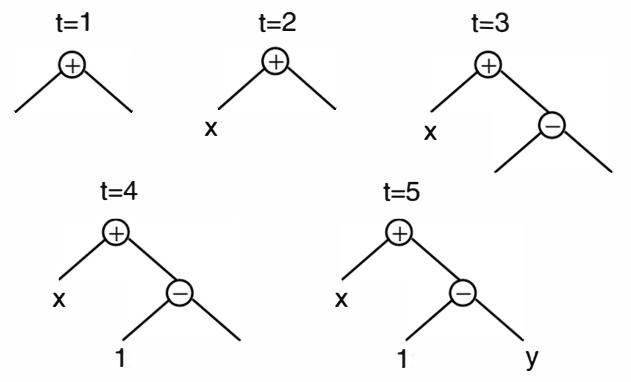


FIGURE 1.11 – Création d'un arbre contenant 5 nœuds avec une profondeur maximale égale à 2 en utilisant la méthode *Grow*

Quant à la méthode *Full*, elle produit des arbres dont tous les terminaux ont la même profondeur, aussi appelés arbres complets. Les nœuds sont tirés aléatoirement parmi les nœuds internes (non terminaux) possibles, ce jusqu'à la profondeur maximale spécifiée. Les nœuds suivants sont alors tirés uniquement parmi les terminaux possibles. La figure 1.12 montre le déroulement de la méthode

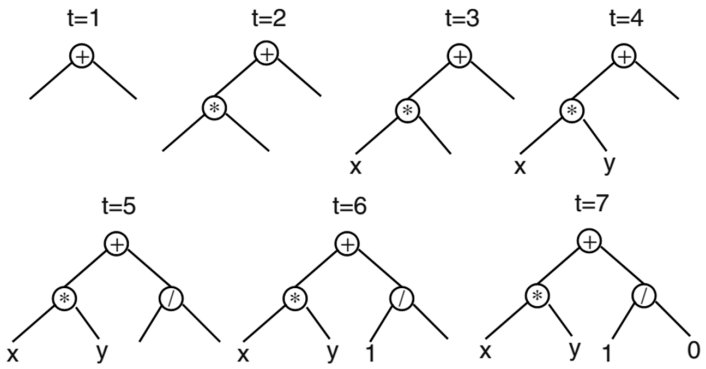


FIGURE 1.12 – Création d'un arbre contenant 5 nœuds avec une profondeur maximale égale à 2 en utilisant la méthode *Full*

full sur un arbre complet de profondeur 2. Concernant le choix des nœuds qui suivent + et *, il faut les choisir parmi des feuilles terminales, sinon l'arbre dépasse la profondeur spécifiée. Le nouveau nœud introduit aux étapes 3, 4, 6, et 7 est donc choisi parmi les terminaux (x, y, 1 et 0).

1.3.3 Croisement

Typiquement, la stratégie de croisement consiste en un échange de deux sous-arbres des deux individus à croiser, sélectionnés *a priori* parmi les plus performants, donc contenant potentiellement des sous-structures intéressantes. Le choix des sous-arbres à échanger est généralement fait par tirage uniforme. La figure 1.13 illustre un exemple de croisement.

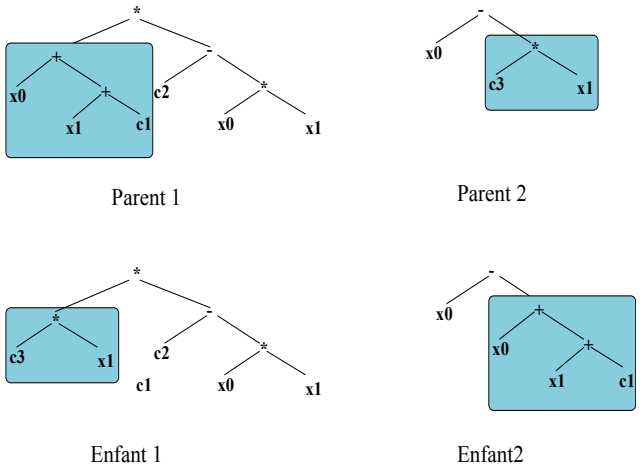


FIGURE 1.13 – Exemple de croisement de deux arbres.

Ce principe général, présenté par Cramer en 1985 [Cramer 1985b], peut être

affiné selon divers points de vue. D'abord, il est nécessaire de respecter la limite de taille assignée aux arbres de la population, afin qu'ils ne deviennent pas inutilement gigantesques. Si les points de croisement choisis ne la respectent pas, alors le croisement ne pourra avoir lieu en l'état. L'attitude adoptée dans ce cas est un paramètre du croisement. Elle pourra être au moins l'une des deux suivantes :

- sélectionner un nouveau couple de parents et tenter de refaire un croisement jusqu'à ce qu'on trouve des descendants qui respectent la contrainte de taille ;
- ou encore choisir des points de croisement différents sur les deux parents sélectionnés, jusqu'à ce qu'on obtienne des descendants satisfaisants.

Une pratique courante est de sélectionner le point de croisement avec les proportions de 10% parmi les feuilles et 90% parmi les nœuds non terminaux.

1.3.4 Mutations

La Programmation Génétique « traditionnelle » proposée par Koza [Koza 1992] n'utilise pas d'opérateurs de mutation. Pour assurer l'accès à toutes les primitives du langage de recherche (e.g. LISP) et assurer la diversité génétique, on utilise des populations de très grande taille, pour qu'elles contiennent le maximum d'information. C'est en 1996 que la mutation a été introduite par Angeline [Angeline 1996] dans le but de réduire les tailles de populations.

On distingue de multiples sortes de mutations du fait de la complexité des structures arborescentes, dont les capacités exploratoires peuvent être locales ou, à l'inverse, de grande envergure. Parmi les différentes mutations, les plus courantes sont :

- *la mutation par insertion* (*grow mutation*) : on ajoute un nœud et des feuilles complémentaires n'importe où dans l'arbre (figure 1.14).

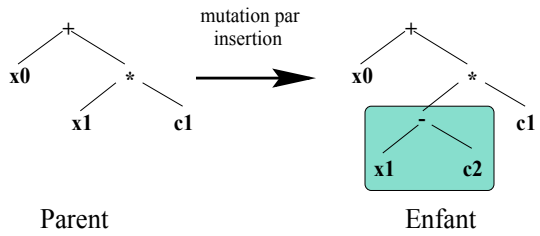


FIGURE 1.14 – Exemple de la mutation par insertion d'un sous-arbre.

- *la mutation par promotion* (*shrink mutation*) : un nœud interne est ôté et l'un de ses fils remonte prendre sa place (figure 1.15).
- *la mutation d'un nœud* (*cycle mutation*) : un nœud (interne ou terminal) de l'arbre est remplacé par un autre nœud de même arité (figure 1.16).
- *la mutation d'une branche* : on élague une branche de l'arbre et on la remplace par un sous-arbre généré aléatoirement (figure 1.17).

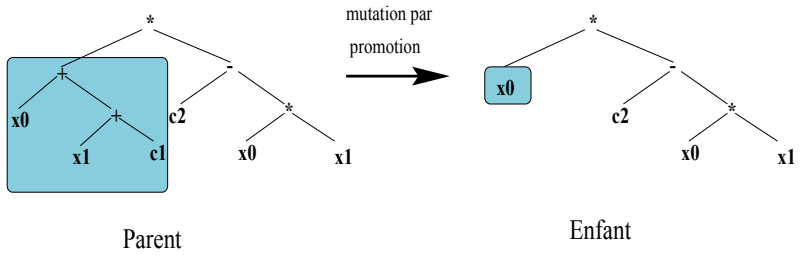


FIGURE 1.15 – Exemple de la mutation par promotion d'un arbre.

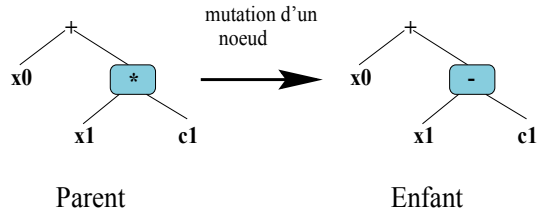


FIGURE 1.16 – Exemple de mutation d'un arbre par remplacement d'un nœud.

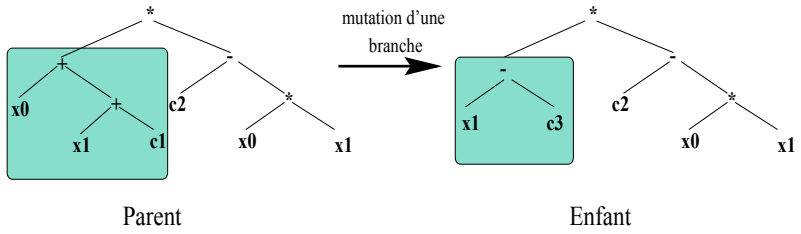


FIGURE 1.17 – Exemple de mutation d'une branche d'un arbre.

Dans le cas où les terminaux peuvent prendre des valeurs numériques, d'autres mutations ont été introduites, dont :

- la mutation des constantes : quelques constantes sont modifiées selon une loi Gaussienne ou uniforme [Angeline 1996],
- la mutation optimisée des constantes : on applique quelques itérations d'un *stochastic hill climber* en vue d'affiner les constantes [Schoenauer *et al.* 1996].

1.4 Résolution par GP

Afin d'appliquer la GP pour résoudre un problème, il faut procéder à un nombre de choix pour répondre aux questions suivantes :

1. Quel est l'ensemble de terminaux ?
2. Quel est l'ensemble de fonctions ? (nœuds non terminaux)

3. Comment appliquer les opérateurs génétiques ?
4. Quelle fonction de performance ou de fitness utiliser pour évaluer les individus ?
5. Quand arrêter les itérations ?

Ces choix sont déterminants pour la GP et dépendent du problème traité. Ci-après quelques directives [Koza 1992] pour bien mener cette étape préparatoire.

1.4.1 L'ensemble des terminaux et l'ensemble des fonctions

Les terminaux d'un programme génétique représentent les valeurs possibles des feuilles de l'arbre. Ils peuvent être répertoriés en :

- les entrées du programme : elles apparaissent généralement sous la forme de variables nommées,
- les fonctions sans argument ou d'arité 0 : ces fonctions peuvent renvoyer des valeurs différentes à chacune de leurs occurrences dans l'arbre et/ou à chaque exécution ; citons le cas de la fonction de tirage réel uniforme rand() ou encore le cas du signal d'un capteur,
- les constantes : les terminaux peuvent prendre des valeurs prédéfinies, souvent générées lors de la création du programme ou lors de sa mutation et appelés constantes aléatoires éphémères (Ephemeral Random Constant ou ERC).

L'ensemble des fonctions constitue, avec l'ensemble des terminaux, les ingrédients de base pour la création automatique de programmes à faire évoluer. Ces programmes sont généralement des expressions et formules exprimées dans des langages spécifiques au domaine du problème posé. L'ensemble des fonctions choisies dépend de la nature du domaine du problème. Par exemple pour un problème numérique simple, on peut se contenter d'un ensemble de fonctions arithmétiques (+, -, *, /). Si par exemple on cherche comment écrire un programme de contrôle d'un robot aspirateur autonome, on peut inclure les fonctions de commandes (paramétrables) suivantes : tourner(angle) ou avancer(distance). Selon Koza [Koza 1992], les ensembles de fonctions et de terminaux doivent être choisis de manière à satisfaire deux conditions : la clôture (closure) et la suffisance.

1.4.1.1 La clôture

La *clôture* peut être définie par la satisfaction de deux conditions à savoir :

1. l'unicité du type de données que retournent les terminaux et les nœuds internes,
2. l'infaillibilité de l'évaluation d'un programme.

Lors de l'application de l'opérateur de croisement (cf. section 1.3.3), le choix du point de croisement est réalisé aléatoirement parmi les terminaux et les fonctions. Par conséquent, chaque nœud peut être placé n'importe où dans l'arbre ce qui veut dire qu'il sera un argument d'une primitive quelconque. Pour cette raison, toutes les fonctions doivent retourner une valeur de même type qui est aussi celui des terminaux. On contraint par exemple des opérateurs arithmétiques +, -, * et / à retourner une valeur entière et accepter deux arguments entiers.

Dans le but de répondre à cette la contrainte d'unicité de type, une conversion est parfois nécessaire. Par exemple, il est possible de convertir des nombres en booléens en transformant les nombres positifs en « vrai » et les nombres négatifs en « faux ». La propriété de clôture semble contraindre l'expression du problème posé par l'utilisateur ; cependant il est aisé de restructurer les fonctions concernées. Prenons par exemple une instruction conditionnelle comme « IF-THEN-ELSE ». Cette instruction dépend de trois arguments : la condition, la valeur retournée si la condition est vraie et l'autre valeur retournée dans le cas contraire. L'argument condition est clairement de type booléen, ce qui suggère une incompatibilité avec d'autres fonctions telles que l'addition +. La conversion présentée plus haut est une solution, mais on préfère le plus souvent transformer cette fonction *si* à trois arguments en une fonction « IFLTZ » (If Less Than Zero).

On peut toutefois contourner la clause d'unicité de type à l'aide d'opérateurs génétiques qui tiennent compte explicitement des types associés aux nœuds et branches. De tels opérateurs utilisent directement l'information de type pour créer de nouveaux branchements cohérents car ils sont tenus de ne produire que des programmes syntaxiquement corrects. Pour que la mutation d'un programme soit correcte, l'opérateur génétique est par exemple contraint de générer un sous-arbre qui retourne le même type que celui du sous-arbre muté. Ce principe s'appelle le typage fort [Montana 1995], où toutes les fonctions n'acceptent pas et ne retournent pas le même type de valeurs. Cela peut se gérer par une série de tentatives jusqu'à ce que le type de retour soit correct.

D'autre part, le principe de clôture garantit l'infaillibilité de l'étape d'évaluation « Evaluation Safety », ce qui est indispensable sachant que des fonctions peuvent échouer pendant l'exécution d'un programme et déclencher une exception, principalement quand les arguments ne correspondent pas à une pré-condition. En effet, il est possible qu'un programme créé durant la phase d'exploration de l'espace de recherche exécute une division par zéro. Ce problème est contourné en modifiant la spécification des primitives. Pour les opérateurs arithmétiques et les fonctions mathématiques qui ont un domaine de définition restreint, on emploie en GP une version dite protégée, qui ne produit pas les exceptions classiques, comme celles de la division par zéro ou de la racine carrée d'un nombre négatif. La version protégée d'une fonction vérifie d'abord que la valeur des arguments d'entrée ne pose aucun problème potentiel, puis calcule le résultat de la fonction en question ; en cas de problème détecté, on retourne une valeur prédéfinie. La division protégée ou protected division, souvent notée %, contrôle si le dénominateur est de valeur nulle ; dans ce cas, elle retourne par convention la valeur 1 quelle que soit la valeur

du numérateur. Keijzer, dans [Keijzer 2003], propose d'effectuer une vérification systématique du domaine de définition des entrées de chaque nœud de l'arbre afin d'éviter les discontinuités dommageables à la qualité du résultat final causées par les fonctions protégées.

La capture des exceptions informatiques peut remplacer l'emploi des fonctions protégées. Dans ce cas, un individu dont l'évaluation provoque une exception se voit attribuer une mauvaise valeur de fitness.

1.4.1.2 La suffisance

C'est la propriété qui cherche à garantir que la solution au problème traité peut être exprimée avec l'ensemble de primitives choisi. Cette garantie ne s'obtient malheureusement que si le problème est déjà résolu de manière théorique ou empirique, et si la solution peut s'exprimer par composition des primitives choisies. L'ensemble des primitives AND, OR, NOT, x_1, x_2, . . . , x_n est considéré suffisant pour résoudre n'importe quel problème d'induction booléenne, puisqu'il peut exprimer toute fonction objectif booléenne de n variables d'entrées x_1, x_2, . . . , x_n. En revanche, le jeu +, -, *, /, x, 0, 1, 2 reste insuffisant pour représenter des fonctions transcendantes, comme la fonction objectif exp(x) qui n'est pas représentable de manière exacte avec ces primitives. Si le jeu de primitives a une expressivité insuffisante, la GP se limitera à chercher une approximation de la fonction objectif. Une approximation rationnelle est généralement une solution acceptable et satisfaisante pour l'utilisateur ce qui est préférable à l'ajout de plusieurs primitives qui introduisent un coût supplémentaire en temps de calcul. Ce temps est toutefois considéré acceptable par [Poli *et al.* 2008].

1.4.2 La fonction de fitness

C'est la fonction qui mesure le degré d'adéquation de la solution examinée au problème le plus souvent sous la forme d'une valeur numérique. Elle permet ainsi d'établir un ordre parmi les solutions de l'espace de recherche qui sera la base pour les méthodes de sélection vues dans la section 1.2.6. Selon le problème et la formulation de la fonction objectif, le meilleur individu est celui qui obtient la fitness minimale ou maximale. Cette mesure peut se baser sur :

- l'écart entre le résultat du programme et le résultat souhaité (problème de régression symbolique),
- le temps d'exécution du programme qui peut refléter le coût énergétique, ou le coût financier,
- la précision pour un problème de classification avec le nombre de bonnes prédictions ou hits,
- la conformité à un ensemble de contraintes,
- . . .

En apprentissage automatique, la fitness se déduit directement de l'erreur d'apprentissage, calculée sur les exemples de la base d'apprentissage qu'on appelle aussi cas de fitness ou fitness cases. La GP détermine la fitness d'un programme (individu de la population) en l'exécutant plusieurs fois selon les cas de fitness disponibles. L'algorithme 1 donne le pseudo-code de l'interprétation d'un programme GP. L'application de cet algorithme sur un arbre est donnée dans la figure 1.18. Le parcours des différents nœuds de l'arbre doit être réalisé de telle sorte qu'une fonction n'est exécutée que si ses arguments sont calculés.

Algorithme 1 Interprétation d'un programme génétique

```
1:  Procedure eval(expr)
2:  si expr est une liste alors
3:     proc ← expr(1) %récupérer la racine
4:     n ← arite(proc)
5:     valeur ← proc(eval(expr(2)), eval(expr(3)), ..., eval(expr(n)))
6:  sinon
7:     si expr est une variable ou constante alors
8:        valeur ← expr
9:     sinon
10:       valeur ← expr() %fonction d'arité 0
11:    fin si
12: fin si
13: retour valeur
14: Fin Procedure
```

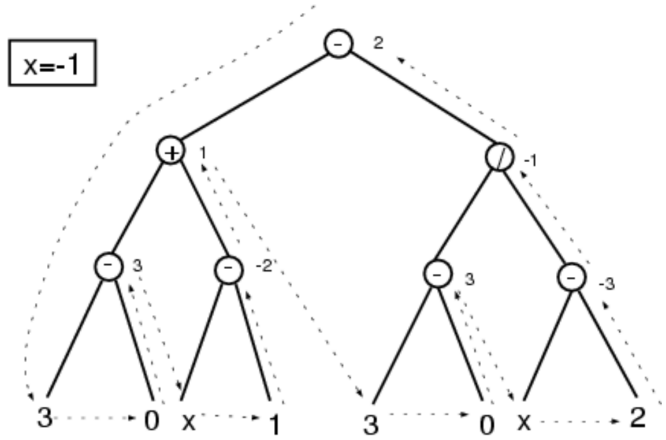


FIGURE 1.18 – Exemple d'interprétation d'un arbre représentant l'expression $(- (+ (- 3\ 0)\ (- x\ 1))\ (/ (- 3\ 0)\ (- x\ 2)))$.

L'étape d'évaluation dans la GP est celle qui s'accapare la plus importante proportion du temps CPU et constitue un obstacle pour l'application de la GP sur

des problèmes de grande taille. C'est pour cette raison que d'autres algorithmes d'interprétation ont vu le jour notamment sur les machines multiprocesseurs ou sur les processeurs multi-core qui introduisent le parallélisme pour accélérer l'étape de calcul de la fitness (voir section 1.4.4). Il existe également, d'autres techniques d'accélération comme l'utilisation de cache de sous-arbres comme dans [Keijzer 2004], ou l'interruption du calcul au-delà d'une borne d'erreur, technique appelée limited error fitness de [Gathercole 1998]. Le chapitre 3, étudie une autre technique d'accélération basée sur la réduction des cas de fitness soumis à chaque individu par des méthodes d'échantillonnage de l'ensemble d'apprentissage.

Les problèmes multi-objectifs sont aussi résolubles par GP via des fonctions de fitness qui combinent autant d'éléments de performance que nécessaire, souvent contradictoires. Ce cas est largement étudié dans [Deb 2001, Fillon & Bartoli 2007].

1.4.3 Paramétrage de la GP

Les choix énumérés dans les paragraphes précédents constituent les premiers paramètres de la GP qui sont très liés au problème à résoudre. À ces éléments, s'ajoutent plusieurs paramètres qui sont partagés avec les différents algorithmes évolutionnaires et qui influencent directement la qualité de la solution obtenue.

Nombre de générations : C'est un nombre entier qui exprime le nombre maximal d'itérations de la boucle évolutionnaire. Il traduit le nombre d'« évolutions » des individus de la population initiale. Donner une valeur trop basse à ce paramètre conduit à une solution aléatoire et qui risque d'être loin de la solution optimale. Une valeur très élevée nécessite un temps plus important et génère des individus qui souffrent du phénomène de « bloating » ou gonflement c'est-à-dire des individus avec une expression trop longue engendrée par les mutations et croisements successifs.

Condition d'arrêt : C'est une condition logique reflétant l'adéquation de la solution et liée à la fonction de fitness utilisée. Le plus souvent, il s'agit d'un seuil de fitness à atteindre. Si cette condition est vérifiée, l'algorithme s'arrête avant d'atteindre le nombre de générations maximal défini.

Taille de la population : Il détermine le nombre d'individus ou solutions candidates au problème. Chaque individu est évalué pour déterminer sa fitness qui sera prise en compte lors de la sélection, croisement, ... La valeur usuelle est de l'ordre de centaines d'individus et elle est limitée par les ressources du système en mémoire. Un nombre important permet d'explorer une zone plus grande de l'espace des solutions.

Probabilités des opérateurs génétiques : Ce sont des ratios qui contrôlent la fréquence d'exécution des mutations et croisement pour générer de nouveaux individus.

Taille maximale d'un individu : Afin de lutter contre le *bloating*, ce paramètre permet de fixer une profondeur maximale de l'arbre qui représente un individu.

Paramètres spécifiques à une variante de GP : Chaque type de GP (cf. 1.5) introduit souvent des paramètres qui ne s'appliquent pas sur les autres variantes. De plus, plusieurs algorithmes ont été élaborés pour les différentes opérations d'initialisation, sélection, croisement et mutation. Le choix d'un algorithme particulier constitue un paramètre supplémentaire.

Le paramétrage de la GP est une tâche importante et ardue. Il a été démontré empiriquement, que de petits changements peuvent provoquer une grande variation dans les résultats obtenus et de grands changements peuvent être sans effet notable. Selon la taxonomie établie par Eiben et al. [Eiben *et al.* 2007], et comme nous l'avons déjà évoqué, les méthodes de paramétrage des algorithmes évolutionnaires sont classés en 2 catégories :

1. Ajustement de paramètres : les valeurs des paramètres sont optimisées en amont.
2. Contrôle de paramètres : l'algorithme commence avec des valeurs non optimales et les améliore au fur et à mesure du processus d'évolution.

L'exploration de l'espace des paramètres faite dans [Sipper *et al.* 2018] a pu montrer qu'un large nombre de combinaisons de valeurs des paramètres couvrant la quasi-totalité de l'espace des paramètres sont optimaux. Ceci s'oppose à l'idée commune qui fixe des valeurs usuelles pour chaque paramètre. L'optimisation de la valeur de ces paramètres a fait l'objet de plusieurs travaux qui sont parfois contradictoires dont une revue est donnée dans [Eiben & Smit 2011].

Il est évident que c'est un problème chronophage et dont la difficulté peut dépasser celle du problème principal. Toutefois, c'est ce qui confère à la GP sa flexibilité et lui permet ainsi de traiter un large éventail de problèmes. Dans ce travail, le paramétrage de la GP n'est pas exploré, nous nous basons sur une pratique très courante qui repose sur l'expérimentation.

1.4.4 Le problème du coût de calcul

La principale faiblesse de GP est son utilisation intensive des ressources mémoire et temps processeur. Ceci constitue un obstacle sur deux perspectives : adopter la GP comme la solution de prédilection d'une part et approfondir le processus évolutionnaire pour l'obtention de meilleures résultats d'autre part.
Les besoins de GP en mémoire et temps processeur dépendent des caractéristiques suivantes :

- la taille des individus (programmes),
- la taille de la population,
- le nombre de générations.

L'utilisation de la mémoire augmente avec la taille des individus et de la population. Par ailleurs, le temps CPU s'accroît avec les 3 précédents facteurs. C'est l'évaluation des individus qui s'accapare quasiment la totalité du temps processeur avec un taux largement supérieur à 80% du temps total. Ce taux peut, dans certains problèmes, atteindre 99% comme dans [Funika & Koperek 2016]. Notre travail se focalise sur ce problème et contribue à le maîtriser en préservant la qualité des solutions.

1.5 Variantes de GP

Bien que la représentation standard telle que définie par Koza est celle arborescente présentée dans la section 1.3.1, d'autres formes de GP ont vu le jour se distinguant les unes des autres essentiellement par le phénotype des individus à faire évoluer par la GP. Parmi ces formes nous évoquons ici quatre familles de GP à savoir :

- GP basée sur les arbres (Tree-based GP : TGP) ; c'est la GP Standard,
- GP basée sur les graphes (Graph-based GP : GGP),
- GP linéaire (Linear GP : LGP),
- GP basée sur les piles (Stack-based GP : SGP).

La première famille a été introduite au début de ce chapitre avec la GP de Koza. Pour cette raison, dans ce qui suit, nous donnons une description succincte des autres variantes de GP avec un accent mis sur la GP cartésienne (CGP) qui appartient à la famille des GGP et que nous avons utilisée lors des expérimentations (chapitre 3).

D'autres classifications qui s'appuient sur des aspects de la GP autre que la représentation peuvent exister. La GP basée sur une grammaire (GGGP : Grammar Guided GP) [McKay *et al.* 2010] est une forme de GP très utilisée. Elle introduit une limitation à l'espace de recherche en imposant des restrictions sur la création d'individus à partir des ensembles de terminaux et fonctions. L'intérêt pour la GGGP a considérablement augmenté avec l'évolution grammaticale (Grammatical Evolution GE) [Ryan *et al.* 1998, Ryan *et al.* 2018]. Les opérateurs génétiques doivent, dans ce cas, préserver la consistance des expressions produites.

1.5.1 GP basée sur les graphes (GGP)

L'idée de cette représentation vient du fait qu'un arbre est un type particulier de graphes. Plusieurs extensions de GP dans ce sens sont proposées par les chercheurs depuis le milieu des années 1990. Nous présentons dans ce qui suit quelques travaux et principalement la GP cartésienne (CGP).

1.5.1.1 Premières représentations

Le premier travail qui a fait évoluer par des algorithmes évolutionnaires des structures de graphes est celui de Louis et al. [Louis & Rawlins 1991]. Ce travail consiste à concevoir des circuits logiques par algorithmes génétiques. Un génotype binaire est alors utilisé pour représenter un réseau de portes logiques numériques. Dans chaque colonne de ce réseau, une porte logique peut être reliée à une porte de la colonne précédente.

Poli a proposé une extension appelée Parallel Distributed GP (PDGP) [Poli 1996, Poli 1999], qui offre la possibilité de réutiliser des résultats partiels comme le montre la figure 1.19 où la représentation basée sur les graphes permet d'éviter la réévaluation du même sous-arbre.

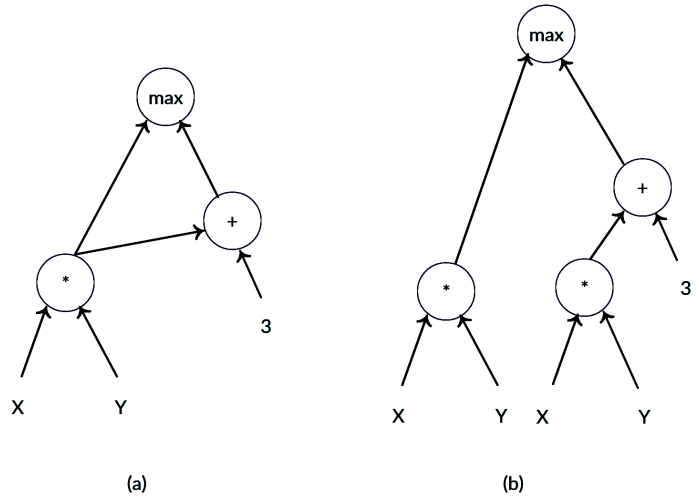


FIGURE 1.19 – Représentations en graphe (a) et arbre (b).

Dans PDGP, les programmes génétiques sont représentés sous forme de graphes orientés où les nœuds sont les fonctions et les terminaux. Les arêtes représentent le flux de données et de contrôle. En associant des labels aux arêtes, PDGP permet de faire évoluer des structures comme les programmes à structure d'arbre, les réseaux de neurones, les réseaux logiques, les automates à états finis et les réseaux à transitions récurrentes.

Une autre extension est proposée par Teller appelée Parallel Algorithm Discovery and Orchestration (PADO) dans [Teller & Veloso 1995, Teller & Veloso 1997]. PADO combine GP et discrimination linéaire pour résoudre des problèmes de reconnaissance d'objets. La figure 1.20 montre un programme dans le système PADO. Dans un programme PADO, chaque nœud est caractérisé par une action et une décision de branchement. Quand un nœud est activé, il commence par exécuter son action puis la décision de branchement qui permet de décider, selon

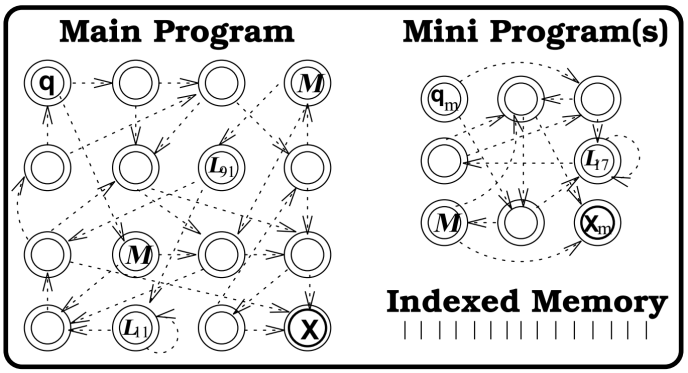


FIGURE 1.20 – Structure de base d'un programme PADO. [Teller & Veloso 1997]

une condition, quel arc activer parmi l'ensemble des arcs sortant de ce nœud.

Dans la figure 1.20, plusieurs types de nœuds sont présents :

- **q** : le nœud de départ et le premier à être exécuté quand ce programme est invoqué,
- **X** : le nœud final, qui lorsqu'il est atteint, est exécuté et le programme s'arrête,
- **M** : appel d'un mini-programme PADO privé qui sera exécuté comme action de ce nœud avant la décision de branchement,
- **L** : appel de programme d'une librairie globale comme les mini-programmes mais qui sont disponibles pour tous les programmes PADO de la population.

1.5.1.2 GP cartésienne (CGP)

La CGP est une variante de GP, créée par Julian Miller [Miller & Thomson 2000], qui représente les programmes génétiques sous la forme de graphes orientés acycliques (voir figure 1.21 [3]). Elle découle du précédent travail de Miller sur la conception de circuits électroniques par algorithmes génétiques dans [Miller *et al.* 1998]. Cette forme de GP est qualifiée de cartésienne vu qu'elle adopte une représentation des programmes génétiques ayant la forme d'une grille bi-dimensionnelle. Dans ce graphe, trois types de nœuds sont présents : les nœuds d'entrée d'un côté (à gauche dans la figure 1.21) qui correspondent à l'ensemble de terminaux de la GP, les nœuds de calcul ou de fonctions dans les colonnes au milieu qui représentent l'ensemble des fonctions de la GP et les nœuds de sortie de l'autre côté contenant les résultats de l'évaluation selon les connexions entre les différents nœuds. Chaque nœud de fonction possède plusieurs entrées et une unique sortie.

CGP utilise un génotype de taille fixe avec des gènes de type entier. Il est représenté sous forme de groupes commençant chacun par l'adresse de la fonction dans

3. Le graphe est modifié par rapport à la version originale pour rectifier certaines erreurs au niveau de la dernière colonne et des nœuds de sortie.

la table des fonctions définies par l'utilisateur appelé « gène de fonction » suivi de plusieurs gènes représentant les entrées de cette fonction et dont le nombre dépend de l'arité de la fonction et sont appelés « gènes de connexion ». Les gènes représentant les nœuds de sorties sont placés à la fin du génotype.

Les connexions dans ce graphe sont régies par les règles suivantes :

- les nœuds d'entrée peuvent être reliés aux entrées des nœuds de fonctions ou directement aux nœuds de sortie,
- les sorties des nœuds de fonction peuvent être dirigées vers les nœuds de sortie ou vers les entrées d'autres nœuds de fonction des colonnes suivantes (selon le paramètre Levels-back),
- les nœuds d'entrée et les sorties des nœuds de fonctions peuvent être connectés de 0 à plusieurs connexions.

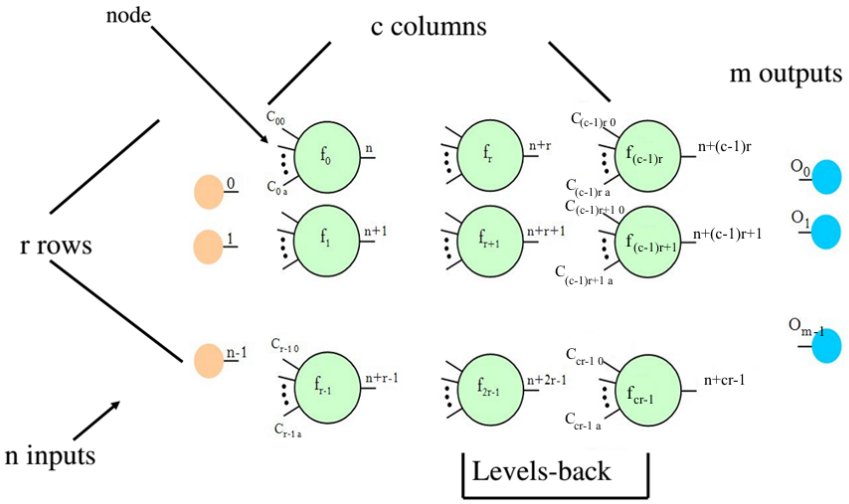


FIGURE 1.21 – Graphe général de CGP [Miller & Harding 2012].

Dans la figure du graphe général de CGP (figure 1.21) :

- n est le nombre d'entrées du programme,
- r est le nombre de lignes,
- c est le nombre de colonnes,
- a est l'arité maximale des fonctions utilisées,
- m est le nombre de sorties.

Les entrées du programme ont les adresses absolues de 0 à $n-1$, suivies par les sorties des nœuds de fonctions avec des adresses attribuées de manière séquentielle et colonne par colonne commençant par n allant jusqu'à $n+cr-1$. Les nœuds de fonctions f_i sont représentés par leur adresses dans la tables des fonctions. Les gènes de connexions C_{ij} sont les entiers représentant les adresses des données allant de 0 jusqu'à l'adresse de la dernière sortie du dernier nœud de

la colonne précédente. Le paramètre *Levels-back* détermine le nombre de colonnes précédentes à partir desquelles un nœud peut chercher ses entrées (en plus des entrées du programme). Par exemple, si sa valeur est égale à 1, chaque nœud du graphe ne peut utiliser que les sorties de la colonne précédente.

La forme générale du génotype d'un programme CGP est présentée dans la figure 1.22.

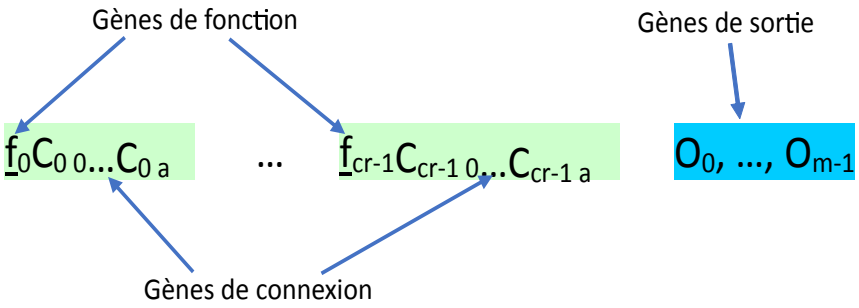


FIGURE 1.22 – Génotype d'un programme CGP.

Exemple :

Soit un programme CGP pour générer des équations mathématiques où les fonctions sont prises à partir de l'ensemble $\{+, -, *, /\}$ codés respectivement aux valeurs entières 0, 1, 2 et 3. Le programme possède comme entrée deux valeurs réelles notées symboliquement par x_0 et x_1 et produit 4 sorties O_A, O_B, O_C et O_D. Pour cet exemple, le graphe CGP est de dimensions 2 lignes x 3 colonnes avec la valeur 2 pour le paramètres levels-back. Un exemple de génotype et le phénotype correspondant sont montrés dans la figure ci-après :

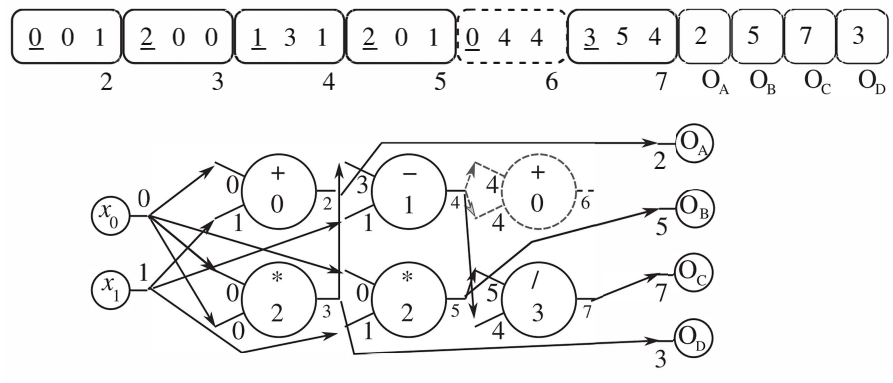


FIGURE 1.23 – Génotype et phénotype d'un programme CGP [Miller & Harding 2012].

Le phénotype correspondant est l'ensemble des 4 équations obtenues en sortie :

$$\begin{aligned} O_A &= x_0 + x_1 \\ O_B &= x_0 * x_1 \\ O_C &= \frac{x_0 * x_1}{X_0^2 - x_1} \\ O_D &= X_0^2 \end{aligned}$$

Cette forme de GP possède les avantages suivants :

- Contrairement aux arbres, elle permet plus qu'un chemin entre une paire de nœuds favorisant ainsi la réutilisation des résultats précédents (voir Figure 1.22).
- Elle est très compétitive avec les autres variantes de GP.
- Elle ne souffre pas d'effet de bloat.
- Elle est facile à implémenter.
- Elle peut avoir plusieurs sorties et par conséquent résoudre plusieurs types de problèmes (e.g. problèmes de classification à plusieurs classes : chaque sortie représente la probabilité de la classe correspondante).

1.5.2 GP linéaire (LGP)

Cette catégorie englobe toute forme de GP qui utilise une liste d'instructions et de terminaux pour représenter les programmes de la population (Figure 1.24).

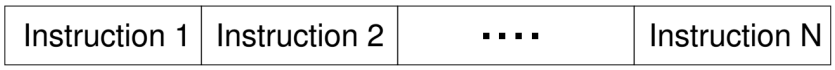


FIGURE 1.24 – Représentation typique d'un programme en LGP.

Ce codage mime la forme de programmes écrits dans un langage de programmation ou celle du code machine. En effet, les instructions considérées peuvent être prises d'un langage de programmation comme dans [Nordin 1994]. Dans ce travail, Nordin et Banzhaf ont étendu ce principe pour représenter des programmes en code machine par l'utilisation des ingrédients de programmes suivants : les tableaux, les opérateurs arithmétiques, les sous-routines, IF-THEN-ELSE, les structures itératives, la récursivité, les fonctions de chaînes et les listes. Ils ont aussi pu intégrer les fonctions du langage C utilisant le jeu d'instructions SPARC sur les stations de travail de SUN.

LGP est conçue et utilisée essentiellement pour deux raisons :

1. l'architecture de Von Neumann définit les programmes comme une liste séquentielle d'instruction où le voisinage exprime une exécution consécutive (sauf les instructions conditionnelles et répétitives), ceci est exprimé dans les travaux [Banzhaf 1993, Openshaw & Turton 1994, Perkis 1994],

2. Les ordinateurs qui vont exécuter les programmes créés par la TGP, utilisent des interpréteurs et des compilateurs pour traduire l'arbre en forme séquentielle. [Nordin 1994, Crepeau 1995, Nordin *et al.* 1999, Eklund 2002] espèrent accélérer l'exécution de leurs programmes GP en utilisant cette forme linéaire qui n'a pas besoin de transformations effectuées par le compilateur.

Typiquement, dans LGP les instructions cherchent leurs entrées dans des registres et mettent leurs résultats aussi dans ces registres. LGP ne distingue pas entre fonctions et terminaux comme le fait la TGP. En plus, dans LGP, la position de l'instruction reflète son ordre d'exécution qui n'est pas le cas dans un arbre TGP.

En utilisant le jeu d'instructions réduit défini dans la tableau 1.1, un programme LGP peut être exprimé dans la notation du langage C comme dans la figure 1.25.

TABLE 1.1 – Exemple d'un jeu d'instructions LGP.

Instructions	
Type	Notation
Opérateurs arithmétiques	$r_i = r_j + r_k$ $r_i = r_j - r_k$ $r_i = r_j \times r_k$ $r_i = r_j / r_k$
Structures conditionnelles	if$(r_j \geq r_k)$ $if(r_j \leq r_k)$ $if(r_j)$
Registres	
r_i	$i \in \{0, 1, \cdots, 7\}$

```
1  void LGP()
2  {
3      double r[8]; //les registres
4      //....
5      r[0] = r[5] + 71;
6      if(r[1]>0)
7          if(r[5]>2)
8              r[4] = r[2] * r[1];
9      r[6] = r[4] + 13;
10     r[7] = r[6] - 2;
11     //....
12 }
13
14
```

FIGURE 1.25 – Exemple de programme LGP avec la notation C.

Les opérateurs génétiques permettent de changer le contenu et la taille des

programmes génétiques. L'opérateur de croisement linéaire typique est illustré par la figure 1.26. Pour chaque parent, un segment de code de taille aléatoire est pris à partir d'un point aléatoire. Le choix du point de croisement et la taille du segment respectent les limites des instructions (l'unité élémentaire est l'instruction). Les deux segments sont alors permutés pour obtenir deux enfants de différentes tailles.

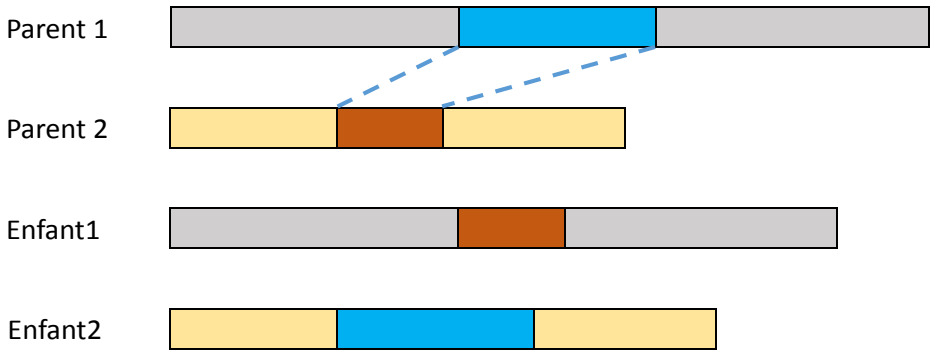


FIGURE 1.26 – Croisement dans LGP.

À l'intérieur de chaque instruction, des *micro-mutations* changent aléatoirement soit le code de l'instruction par un autre à partir du jeu d'instructions prédéfini ou aussi un registre ou une constante par un registre ou constante qui leur sont équivalents. D'autres opérateurs avancés sont définis dans [Brameier 2004, Langdon & Banzhaf 2005].

1.5.3 GP basée sur les piles (SGP)

Perkis [Perkis 1994] a introduit SGP dans le cadre de la recherche d'une représentation qui soit simple et flexible tout en maintenant l'expressivité offerte par GP. Les individus sont toujours des expressions construites à partir des ensembles de terminaux et fonctions prédéfinis mais avec une organisation linéaire associée à une pile d'exécution. Les arguments ne sont pas définis ; les fonctions les puisent dans une pile numérique. Par la suite, le résultat est empilé dans cette pile (Push). L'exemple suivant (tableau 1.2) représente un programme qui résout le problème de régression symbolique $3X^3 + 2X^2 + X$:

TABLE 1.2 – Exemple d'un programme SGP.

$(X)(X)(X)(*)(*)(X)(X)(X)(X)(X)(+)(1)(+)(X)(+)(X)(+)(*)(+)(1)(+)(*)(-)$

L'évaluation de cet individu consiste à parcourir cette expression et pour chaque élément traité :

- Si c'est un terminal (X) ou une constante alors empiler sa valeur dans la pile.

- Si c'est une fonction alors dépiler les arguments du sommet de la pile et empiler le résultat.

Quant aux opérateurs génétiques, ils sont similaires à ceux utilisés avec les algorithmes génétiques.

Ultérieurement, une autre variante appelée PushGP [Spector & Robinson 2002, Spector *et al.* 2005] est définie grâce à la représentation des programmes dans le langage Push. PushGP utilise aussi des piles pour chaque type de données (entiers, réels, booléens, etc.).

1.6 Applications de la GP

1.6.1 Domaines d'application

Opter pour la GP lors de la résolution d'un problème peut être un choix judicieux si ce problème possède quelques ou toutes les caractéristiques suivantes :

- les relations entre les variables explicatives du problème à résoudre sont inconnues (ou mal connues) ;
- l'identification de la taille et la forme des solutions optimales du problème à résoudre est une question à part entière ;
- une quantité de données suffisante pour permettre de décrire le problème à résoudre est disponible ;
- il est possible d'évaluer la performance d'une solution candidate ;
- les méthodes classiques de résolution mathématiques n'ont pas permis de découvrir de solution analytique au problème ;
- une solution partielle, c'est-à-dire complète mais non optimale, du problème à résoudre est acceptable.

Avec plus de 25 années d'existence, les travaux autour de la GP ont pu traiter divers problèmes et couvrir un large spectre de domaines. La collection des bibliographies en informatique [4] recense 13794 publications en GP qui se classe en seconde position par rapport aux publications dans l'IA. Sous ses différentes formes et variantes la GP a été utilisée avec succès comme outil d'apprentissage automatique, d'optimisation ou de programmation automatique et par la suite a été appliquée dans les domaines de ces disciplines comme : les problèmes de classification, de régression symbolique, de conception et ingénierie, de robotique, de finance, de médecine etc. Un florilège des plus marquantes applications de la GP est dressé dans [Gandomi *et al.* 2015].

Bien que nous citons plusieurs domaines d'application de la GP, afin de montrer son potentiel, nous nous intéressons dans ce travail à l'étude de la GP face à un problème de classification. Plusieurs travaux ont été menés dans ce contexte [Espejo

4. https://liinwww.ira.uka.de/bibliography/index.html (visité le 19/01/2019)

et al. 2010] sous la forme d'applications du monde réel comme : le diagnostic médical, la détection de fraudes, la reconnaissance de formes, la détection d'intrusion, etc.

1.6.2 Les « Humies»

Pour montrer la puissance de sa GP, Koza et al. l'ont appliqué sur des problèmes de conception et dimensionnement de circuits électroniques [Koza *et al.* 1999, Koza *et al.* 2003]. Ceci a permis de réinventer 21 brevets et un nouveau brevet [5]. Ensuite, Koza a introduit la notion de « Human Competitive » pour qualifier un résultat obtenu par une intelligence artificielle s'il satisfait au moins un des 8 critères définis dans [Koza *et al.* 2003] et qui sont :

1. Le résultat a fait l'objet d'un brevet dans le passé, est une amélioration d'un brevet ou pourrait faire l'objet d'un nouveau brevet.
2. Le résultat est équivalent ou meilleur qu'un résultat accepté comme un nouveau résultat scientifique à la date de sa publication dans une revue à comité de lecture.
3. Le résultat est équivalent ou meilleur qu'un résultat qui a été placé dans une base de données maintenu par des experts de renommée internationale.
4. Le résultat est publiable en tant que nouveau résultat scientifique, indépendamment du fait qu'il a été obtenu par une machine.
5. Le résultat est égal ou supérieur à la solution la plus récente créée par l'homme à un problème de longue date. Un problème pour lequel il y a eu une succession d'améliorations de solutions créées par l'homme.
6. Le résultat est équivalent ou supérieur à un résultat considéré comme un succès dans son domaine lors de sa découverte.
7. Le résultat résout un problème de difficulté incontestée dans son domaine.
8. Le résultat remporte un concours réglementé impliquant des candidats humains (sous la forme de joueurs humains ou de programmes informatiques écrits par des humains).

Depuis 2004, une compétition est organisée conjointement avec la conférence GECCO (Genetic and Evolutionary Computation COnference) appelée « The Humies ». GP, en concurrence avec les autres algorithmes évolutionnaires, a été récompensée avec le premier prix dans plusieurs éditions. Quelques exemples sont exposés dans le tableau 1.3.

L'une des plus emblématiques réalisations de la GP, est la conception de l'antenne utilisée pour la mission ST5 de la Nasa (figure 1.27).

5. https://patentimages.storage.googleapis.com/8a/5b/e9/b69e6415df8fe1/US6847851.pdf

TABLE 1.3 – Quelques succès de GP dans les « Humies».

Année	Titre
2004	Conception d'une antenne (voir figure 1.27) pour la mission ST5 de la NASA [Lohn *et al.* 2005]
	Création d'algorithmes en informatique quantique [Spector 2004]
2005	Conception de cristaux photoniques bidimensionnels [Preble *et al.* 2005]
2008	Application à l'algèbre de type fini [Spector *et al.* 2008]
2009	Réparation automatique de programmes informatiques [Forrest *et al.* 2009]
2010	Fonction d'énergie pour la prédiction de la structure de protéine [Widera *et al.* 2009]
2012	Conception de jeux [Browne 2012]
2018	Dispositif de surveillance pour les dyskinésies de Parkinson [Lones *et al.* 2017]

1.6.3 Art évolutionnaire

L'association entre informatique et création artistique dans des disciplines comme « l'art génératif» [Whitelaw 2004] est l'un des domaines où la GP et les algorithmes évolutionnaires ont pu s'illustrer comme outils de création artistique automatique (les premiers essais remontent à ceux de Sims en 1991 [Sims 1991]). Des système à base de GP ont réussi à réaliser des œuvres artistiques comme la composition musicale [Hofmann 2015] ou les arts graphiques [Bakurov & Ross 2018] (cf. figure 1.28). Depuis 2012, EvoMUSART est devenue une conférence [6] qui regroupe les travaux concernant l'utilisation des EA dans les tâches artistiques.

1.7 Conclusion

Dans ce chapitre, nous avons présenté l'historique et les concepts de base des EA et de la GP dans sa version standard telle que définie par Koza [Koza 1992]. Cette version est basée sur une représentation arborescente des individus de la population et utilise, par conséquent, des opérateurs génétiques appropriés (mutation et croisement d'arbres).

Plusieurs variantes de la GP ont été définies pour apporter des réponses à des besoins spécifiques lors de la résolution de problèmes. Le nombre de travaux et publications ainsi que des applications pratiques de la GP témoignent de la flexibilité de cette métaheuristique pour être adoptée dans un grand éventail de domaines. Elle a été, en particulier, appliquée dans l'apprentissage automatique

6. Il y a d'autres conférences et journaux spécialisés dans l'art génératif comme Generative Art Conference : http://www.generativeart.com

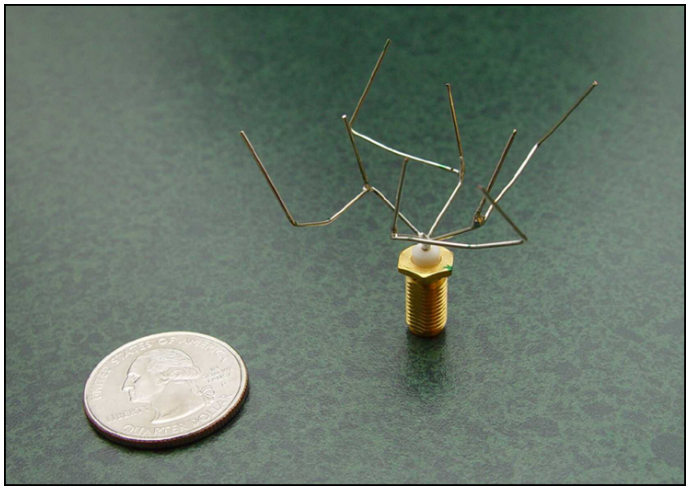


FIGURE 1.27 – Antenne pour la mission ST5 de la NASA.



FIGURE 1.28 – Rendu artistique avec CGP [Bakurov & Ross 2018].

supervisé pour résoudre des problèmes de classification et a permis d'obtenir des résultats très satisfaisants.

Néanmoins, la GP sollicite des ressources importantes à cause du temps de calcul généré par l'étape d'évaluation. Avec la croissance exponentielle du volume de données des bases utilisées dans les problèmes d'apprentissage supervisé, ce temps de calcul devient un obstacle devant son application.

Le chapitre suivant revient sur cette question et permet de positionner notre travail entre GP, apprentissage automatique et Big Data.

CHAPITRE 2

GP pour l'apprentissage automatique

Sommaire

2.1 Introduction 50
2.2 Apprentissage automatique 50
2.2.1 Définitions 50
2.2.2 Apprentissage supervisé 51
2.2.3 Apprentissage non-supervisé 52
2.2.4 Apprentissage semi-supervisé 52
2.2.5 Apprentissage par renforcement 52
2.2.6 Apprentissage actif (active learning) 52
2.2.7 Problème de sur-apprentissage 53
2.3 Adéquation de GP pour la classification supervisée 55
2.3.1 Problème de classification 55
2.3.2 Approches d'apprentissage automatique par GP 55
2.3.3 Application de la GP pour un problème de classification . . . 56
2.4 Apprentissage par GP à partir du Big Data 60
2.4.1 Défis de l'apprentissage à partir du Big Data 60
2.4.2 Approches pour remédier au coût d'apprentissage 61
2.4.3 Mise à l'échelle par la manipulation des données, traitements et algorithmes 62
2.4.4 Mise à l'échelle par le paradigme d'apprentissage 63
2.4.5 Approches étudiées 63
2.5 Conclusion 64

2.1 Introduction

L'apprentissage automatique est l'un des domaines les plus actifs de l'Intelligence Artificielle. Il fournit différents outils pour l'extraction de connaissances à partir des données et il est appliqué dans plusieurs domaines (reconnaissance faciale et vocale, robots et véhicules autonomes, etc.).

Carbonell et al. [Carbonell *et al.* 1983] donnent une classification des techniques d'apprentissage automatique selon la stratégie d'apprentissage adoptée. Les EA et particulièrement GP s'inscrivent dans la catégorie « apprendre à partir d'exemples » [McDermott & O'Reilly 2015].

En effet, chaque individu de la population traitée par un EA peut être considéré comme un exemple de l'espace de recherche et la fonction de fitness mesure à quel point c'est un bon ou mauvais exemple.

De plus, le schéma général d'un AE est très proche des méthodes d'optimisation classiques. L'apprentissage automatique peut être modélisé aussi sous la forme d'un problème d'optimisation.

Parmi les différentes catégories d'EA, c'est la GP qui est la plus adaptée aux problèmes d'apprentissage automatique grâce à sa flexibilité quant à la représentation de l'espace de recherche. Dans ce qui suit, nous montrons comment la GP est appliquée sur un problème particulier de l'apprentissage automatique supervisé qui est celui de la classification (section 2.3) après une introduction succincte à l'apprentissage automatique (section 2.2). La section 2.4, présente les problèmes que soulèvent l'utilisation de la GP pour un problème d'apprentissage supervisé Big Data. Elle énonce les voies choisies dans cette thèse pour résoudre le problème du temps d'apprentissage.

2.2 Apprentissage automatique

2.2.1 Définitions

Le terme « Machine Learning » traduit en apprentissage automatique ou encore artificiel a été inventé par Arthur Samuel [Samuel 1959] en 1959 à l'occasion de la création de son programme de jeu d'échecs et qu'il définit ainsi :

> « A field of study that gives computers the ability to learn without being explicitly programmed. »

Une seconde définition est celle de Mitchell [Mitchell 1997] :

> « A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. »

Tandis que la première définition insiste sur la capacité des machines à réaliser des tâches sans être programmées à le faire, la seconde met l'accent sur l'aspect

incrémental de l'apprentissage d'une tâche lié à une amélioration mesurée sur un ensemble d'exemples. L'apprentissage est un processus guidé par les données appelées « dataset » ou base d'apprentissage. Par conséquent, la qualité des données infère celle de l'apprentissage.

Les algorithmes d'apprentissage automatique sont généralement classés sous les catégories suivantes : apprentissage supervisé, semi-supervisé, non-supervisé et par renforcement. Il est à noter que d'autres paradigmes d'apprentissage sont apparus avec les nouvelles applications et tâches d'apprentissage ainsi que les nouvelles sources de données. Dans [Cornuéjols & Miclet 2010], un panorama de ces paradigmes est donné comme l'apprentissage incrémental ou en ligne (on-line learning), l'apprentissage actif ou les apprentissages collaboratifs. Nous nous intéressons à l'apprentissage actif qui apporte une solution au coût d'apprentissage (cf. section 2.4).

2.2.2 Apprentissage supervisé

Dans ce cas, la base d'apprentissage est constituée d'exemples d'apprentissage étiquetés. Chaque exemple est un couple $(\overrightarrow{X_i}, Y_i)$ où $\overrightarrow{X_i}$ est le vecteur de données en entrée décrit par un nombre m d'attributs. Y_i est un attribut supplémentaire contenant le résultat attendu permettant ainsi de superviser ou guider l'apprentissage à travers des mesures de l'écart par rapport à ces valeurs. L'objectif est d'atteindre un niveau de précision par rapport aux exemples d'apprentissage traités tout en ayant la capacité de généralisation à d'autres exemples ne faisant pas partie de l'ensemble d'apprentissage en évitant le sur-apprentissage (overfitting cf. 2.2.7). Les problèmes appartenant à cette catégorie sont souvent :

- Classification[1] : Il s'agit de déterminer la classe Y_i de chaque exemple X_i parmi un nombre de classes fini. Un exemple d'application est le diagnostic d'une maladie, la détection d'intrusion, la détection de fraude, ... C'est ce type de problème qui est étudié dans cette thèse. Plus de détails sont fournis dans la section suivante.

- Régression : C'est un problème très similaire à celui de la classification. La seule différence réside dans la valeur à prédire qui est, dans ce cas, continue. La prévision de la température ou du niveau d'humidité est un exemple de régression.

- Séries temporelles : C'est un problème largement utilisé en finance. Il consiste à prédire la valeur future d'un attribut en se basant sur plusieurs valeurs prises dans le passé.

1. C'est une traduction du mot anglais *classification*, nous l'utilisons pour désigner la classification supervisée que Cornuéjols et Miclet [Cornuéjols & Miclet 2010] ont traduit en « classement ».

2.2.3 Apprentissage non-supervisé

Si les exemples de la base d'apprentissage ne sont pas étiquetés (pas de valeur cible Y_i), alors il s'agit de l'apprentissage non supervisé. C'est la transformation du vecteur d'entrée vers un autre vecteur ou vers une valeur pour résoudre un problème particulier. Par exemple, dans le problème de détection de points aberrants (outliers), $\overrightarrow{X_i}$ est transformé en une valeur réelle indiquant l'écart par rapport à un exemple de données typique. Dans cette catégorie s'insèrent les problèmes de segmentation (traduction de clustering appelée encore classification non supervisée), de réduction de dimension, de découverte de règles d'association, etc. L'objectif n'est plus de prédire mais de découvrir une structure des données en entrée. Les modèles créés ne peuvent être évalués par rapport à des valeurs attendues ce qui rend leur résolution par GP impossible, vu l'absence de mesure de performance des solutions (voir section 1.4).

2.2.4 Apprentissage semi-supervisé

La différence par rapport à l'apprentissage supervisé réside dans la base d'apprentissage. Elle est, dans le cas semi-supervisé mixte contenant des exemples labellisés (une minorité) et d'autres sans la valeur cible Y_i.

2.2.5 Apprentissage par renforcement

Ce type avancé d'apprentissage remplace l'absence d'une mesure précise de l'erreur (apprentissage supervisé) par un feedback sous la forme d'une récompense (reward) positive ou négative. Ce feedback est obtenu en examinant l'état de l'environnement dynamique dans lequel évolue le système en prenant des actions bien déterminées. L'apprentissage par renforcement est appliqué dans les domaines de la robotique, voitures autonomes, gestion de ressources, etc.

2.2.6 Apprentissage actif (active learning)

L'apprentissage actif [Atlas *et al.* 1990, Cohn *et al.* 1994] peut être défini comme :

> « any form of learning in which the learning program has some control over the inputs on which it trains. »
>
> « toute forme d'apprentissage dans laquelle le programme d'apprentissage possède un contrôle sur les entrées à partir desquelles il apprend. »

C'est le contrôle sur les exemples d'apprentissage qui donne à cette approche l'attribut *actif* et l'oppose à l'apprentissage *passif* qui n'a aucun rôle dans la sélection des données. Ce contrôle peut être effectué à travers des requêtes émanant de l'apprenant sur un exemple d'apprentissage (e.g. demander sa classe) : c'est l'apprentissage par requêtes ou *query learning*, ou encore par l'organisation des expériences d'apprentissage (*optimal experimental design*).

Au départ, l'apprentissage actif a été introduit pour les problèmes d'apprentissage où le coût d'étiquetage des exemples d'apprentissage est coûteux (en temps, en ressources, etc.). Ensuite, il a été appliqué pour améliorer la qualité de l'apprentissage [Cohn *et al.* 1994, Ilhan & Fatih Amasyal 2014].

Une revue étendue des travaux et approches de l'apprentissage actif a été proposée par Settles [Settles 2009]. Il recense trois scénarios de l'apprentissage actif à savoir :

1. Synthèse de requêtes : L'apprenant construit ses requêtes *de novo* selon les besoins d'apprentissage.
2. Échantillonnage à base de flux (Stream-based Sampling) : Les exemples d'apprentissage sont délivrés à l'apprenant de façon séquentielle et c'est à lui de décider de demander l'étiquette ou non.
3. Échantillonnage à base de pool (Pool-based sampling) : Un large pool d'exemples d'apprentissage (base d'apprentissage) est disponible. L'apprenant sélectionne un échantillon d'exemples d'apprentissage pour calculer leurs étiquettes.

2.2.7 Problème de sur-apprentissage

L'objectif de l'apprentissage supervisé est de créer des modèles (hypothèses), à partir d'un ensemble d'exemples, qui se généralisent sur des exemples inconnus. Le sur-apprentissage appelé aussi sur-adaptation ou encore sur-ajustement (en anglais overfitting) est un problème spécifique à l'apprentissage supervisé. Dans [Mitchell 1997], il est défini ainsi :

> Étant donné un espace d'hypothèses $\mathcal{H}$, une hypothèse $h \in \mathcal{H}$ est dite sur-adaptée par rapport aux données d'apprentissage s'il existe une hypothèse alternative $h' \in \mathcal{H}$ telle que h a une erreur d'apprentissage inférieure à celle de h' sur les exemples d'apprentissage, mais h' a une erreur plus petite que h sur l'ensemble des instances.

Le sur-apprentissage survient quand le modèle obtenu par l'algorithme d'apprentissage est très performant sur l'ensemble d'apprentissage mais ses performances se dégradent quand il est évalué sur de nouvelles données. Ce modèle a donc de mauvaises capacités de généralisation. Le sur-apprentissage peut être expliqué en décomposant l'erreur de généralisation en *biais* et *variance*. Le *biais* est la tendance du modèle à reproduire la même erreur. Il correspond à une propriété intrinsèque de la méthode d'apprentissage. Il permet de réduire l'espace des hypothèses $\mathcal{H}$ et est donc indispensable pour pouvoir apprendre. La *variance* est la sensibilité aux données d'apprentissage. Le sous-apprentissage ou sur-généralisation est l'antonyme du sur-apprentissage qui caractérise un manque d'adaptation aux données d'apprentissage.

La figure 2.1 montre ces deux phénomène sur un problème de régression linéaire. Dans les deux cas, le modèle n'est pas généralisable. Un bon modèle est

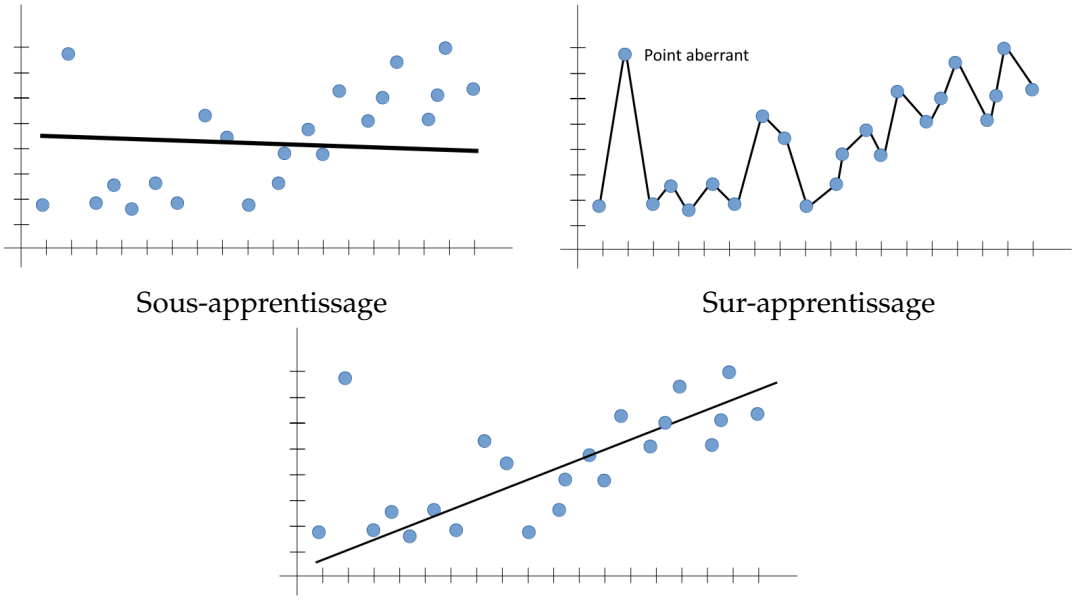


FIGURE 2.1 – Sur-apprentissage et sous-apprentissage.

un juste milieu.

Parmi les causes du sur-apprentissage nous pouvons citer :

- La complexité du modèle : utiliser un modèle sophistiqué pour un problème simple. Par exemple l'utilisation d'un modèle quadratique pour des données linéaires dans le cas de la régression linéaire.
- Les données : quand les données sont bruitées, le modèle s'adapte aux données erronées et aux points aberrants dans la base d'apprentissage. De plus, un nombre insuffisant d'exemples d'apprentissage ne permet pas de trouver un modèle généralisable.
- La taille de l'ensemble des hypothèses : la richesse des modèles à examiner par l'algorithme d'apprentissage augmente la probabilité de trouver un modèle trop adapté à la base d'apprentissage et donc le sur-apprentissage.

Pour éviter le sur-apprentissage, plusieurs techniques peuvent être appliquées qui visent les causes citées ci-dessus. L'apprentissage ensembliste, le partitionnement de données et l'enrichissement de données sont des exemples de techniques génériques. D'autres sont spécifiques à l'algorithme d'apprentissage considéré comme la régularisation pour les réseaux de neurones [Akaho 1992] et l'élagage pour les arbres de décision [Bramer 2013]. Le sur-apprentissage avec la GP a été étudié dans [Zegklitz & Posík 2015, Dabhi & Chaudhary 2012]. Il peut être contourné par des techniques comme l'échantillonnage [Paris *et al.* 2003, Gonçalves *et al.* 2012, Gonçalves & Silva 2013] ou le *backwarding* [Robilliard & Fonlupt 2001].

2.3 Adéquation de GP pour la classification supervisée

Nous traçons, dans cette section, les principaux attributs de GP qui ont suscité son utilisation pour l'apprentissage automatique et la classification supervisée en particulier. Nous commençons par une définition formelle du problème de classification pour passer aux différentes approches d'apprentissage par GP.

2.3.1 Problème de classification

Pour modéliser un problème de classification supervisée, nous considérons l'ensemble d'apprentissage $\mathcal{S}$ de taille n :

$$\mathcal{S} = \{(\overrightarrow{X_i}, Y_i), \ldots, (\overrightarrow{X_n}, Y_n)\}$$

où chaque $\overrightarrow{X_i} = (X_{i1}, \ldots, X_{im})$ représente un exemple d'apprentissage avec m attributs et $Y_i \in \{1, \ldots, c\}$ est la variable de la classe réelle de cet exemple. Ce problème consiste à chercher une fonction $h(X)$ évaluée pour chaque exemple à $\hat{Y}_i = h(\overrightarrow{X_i})$ qui soit la plus proche de la classe réelle Y_i. $h(\overrightarrow{X})$ est appelé un classifieur : une fonction appartenant à l'espace des hypothèses $\mathcal{H}$.

Le cas le plus simple est quand le nombre de classes est égal à 2 (spam/non spam, attaque/normale, falsifié/authentique, etc.). Ce problème est appelé classification binaire. Pour un nombre de classes supérieur (3 ou plus) c'est la classification multiclasses. La majorité des algorithmes d'apprentissage produisent une seule sortie ce qui convient aux problèmes de classification binaire. Deux stratégies ont été définies pour résoudre un problème de classification multiclasses avec une seule sortie :

- One-vs-all : Elle consiste à créer autant de classifieurs que le nombre de classes. Chaque classifieur traite l'appartenance ou non d'un exemple à une classe bien déterminée. Cette stratégie considère les autres classes comme une unique classe.
- One-vs-one : Pour revenir au cas binaire, cette stratégie crée un classifieur pour chaque paire de classes.

Pour évaluer la performance d'un classifieur, une matrice de confusion (voir le tableau 2.1) est calculée. Elle permet de dériver plusieurs autres mesures telles que la précision, le taux de détection, etc. (voir celles définies dans le chapitre 3).

2.3.2 Approches d'apprentissage automatique par GP

Pour trouver les origines de l'application de la GP comme un algorithme d'apprentissage, il suffit d'examiner les premiers travaux associant apprentissage automatique et algorithmes évolutionnaires. En effet, les travaux autour des algorithmes génétiques, qui ont la communauté la plus large parmi les EA, ont dégagé deux principales approches [DeJong 1988] :

TABLE 2.1 – Matrice de confusion.

		Prédiction	
		Oui	*Non*
Réel	*Oui*	True Positives (TP)	False Positives (FP)
	Non	False Negatives (FN)	True Negatives (TN)

- L'approche de Michigan proposée par Holland [Holland 1986] qui utilise un sous-ensemble d'individus de la population comme résultat d'apprentissage où chaque individu code une règle particulière. L'évolution des règles (individus) est effectuée par l'interaction avec l'environnement. Elle est appliquée pour créer un système de classeurs opérant selon le modèle de l'apprentissage par renforcement.
- L'approche Pittsburgh [Smith 1983] utilise le « meilleur » individu de la population qui code plusieurs règles complexes. L'apprentissage supervisé, qui fournit une mesure précise de la qualité des individus, est donc son domaine d'application. La complexité des individus exige une représentation plus expressive. La GP possède un biais de représentation plus adapté que les autres EA. C'est dans cette approche que s'insère les travaux de cette thèse.

Par ailleurs, dans [Cornuéjols & Miclet 2010], les auteurs considèrent l'évolution simulée (pour dire les EA) comme une méthode d'apprentissage par exploration. Ils montrent l'avantage d'utiliser cette famille d'algorithmes pour l'apprentissage qui combine entre méthodes d'optimisation (comme le gradient) et celles de recherche par exploration ou guidée. En effet, les EA permettent de faire une recherche dans l'espace des hypothèses $\mathcal{H}$ en utilisant une population d'hypothèses qui s'améliorent de façon stochastique. Cette recherche est guidée par les opérateurs génétiques et ce en ajoutant un espace « génotypique » $\mathcal{G}$ associé à celui des hypothèses. Ce nouvel espace contribue à l'accomplissement des missions qui incombent à l'espace des hypothèses à savoir : réaliser un biais adéquat, avoir un langage d'expression des hypothèses intelligible et posséder une topologie favorisant une exploration efficace. La mesure de performance s'effectue dans l'espace $\mathcal{H}$ puis la variation est réalisée dans l'espace $\mathcal{G}$. Avec GP, l'expressivité de la représentation a fait que l'espace $\mathcal{G}$ est confondu avec $\mathcal{H}$.

2.3.3 Application de la GP pour un problème de classification

Afin d'illustrer les différentes étapes de la résolution d'un problème d'apprentissage automatique par GP, nous considérons un problème de classification qui consiste à détecter les spams (courrier indésirable). La base d'apprentissage est créée par Hewlett Packard Labs [UCI 1999b]. Elle comporte 4601 exemples d'apprentissage décrits par 57 attributs réels représentant des mesures statistiques

des différents emails (fréquences de certains mots, longueur des séquences en majuscules, ...). Le dernier attribut est celui de la classe réelle (spam = 1 et 0 sinon). L'objectif dans ce problème est de prédire la classe de chaque email. Le pré-traitement des données comme la mise à l'échelle ou le calcul des valeurs manquantes ou encore la sélection des attributs ne sont pas traités dans cet exemple.

Les étapes d'application de GP pour trouver une solution à ce problème sont :

1. **Choix de l'ensemble des fonctions :** avec les problèmes de classification, c'est souvent les fonctions arithmétiques et logiques qui sont utilisées pour la création des individus (expressions). L'ensemble choisi est :

$$\mathcal{N} = \{-, +, *, /, \text{AND}, \text{OR}, \text{NOT}, \text{IF-THEN-ELSE}, <, =\}$$

2. **Choix de l'ensemble des terminaux :** Il se compose de 57 variables représentant les attributs de chaque exemple d'apprentissage de IN0 à IN56 des constante booléenne True et False et une constante réelle aléatoire (C).

$$\mathcal{T} = \{IN0 \ldots IN56, True, False, C\}$$

3. **Choix de la fonction de fitness :** L'objectif étant de maximiser le nombre de bonnes prédictions, la valeur de fitness sera le nombre de « hits ». Un individu est considéré meilleur qu'un autre si sa fitness est supérieure à la sienne.

4. **Évaluation :** Pour un individu qui représente un classifieur, l'évaluation se fait en deux étapes. Dans la première étape, une valeur est calculée selon son phénotype qui est une expression formée par les éléments des ensembles $\mathcal{N}$ et $\mathcal{T}$ en remplaçant, à chaque fois, les terminaux IN0 à IN56 par les valeurs d'un exemple de la base d'apprentissage. Ceci est répété pour tous les exemples de la base d'apprentissage. La deuxième consiste à interpréter la valeur calculée en une classe. Par exemple, si cette valeur est supérieure à 0 alors la classe prédite est « spam ». Il est possible aussi d'utiliser une transformation supplémentaire pour obtenir une probabilité avec une fonction sigmoïde utilisée avec les réseaux de neurones :

$$f(x) = \frac{1}{1 + e^{-x}}$$

Dans ce cas, si la valeur est supérieure à 0.5 le mail est classé comme un spam. La classe prédite est comparée à celle fournie dans la base d'apprentissage. En cas de concordance, la valeur de fitness est alors incrémentée.

5. **Choix de la condition d'arrêt :** Les conditions les plus utilisées sont soit fixer une valeur de fitness à atteindre ou un nombre de générations (qui est ajusté selon les résultats obtenus). Pour notre cas nous avons choisi un nombre de générations égal à 30.

6. **Autres paramètre de GP :** Ces paramètres sont aussi importants que les autres étapes. Ils permettent essentiellement de contrôler l'exploration et l'exploitation de l'espace de recherche. Parmi ces paramètres nous pouvons citer la taille de la population, les probabilités et types pour le croisement et la mutation, la méthode de sélection, la méthode d'initialisation de la population ...

7. **Lancer le run :** Il ne reste plus que déclencher le processus évolutionnaire qui sera suivi par une mesure de performance à la fin du run sur un ensemble de test composé de nouveaux exemples non inclus dans l'ensemble d'apprentissage. Si cet ensemble n'est pas fourni, comme dans notre cas, la base d'apprentissage est scindée en deux ensembles avec des proportions respectives de 80% (3686 exemples) et 20% (915 exemples) pour l'ensemble d'apprentissage et celui de test. Le meilleur classifieur obtenu par un run de 10 générations est représenté par la figure 2.2. Avec le même paramétrage et pour 30 générations, un autre classifieur est représenté par la figure 2.3.

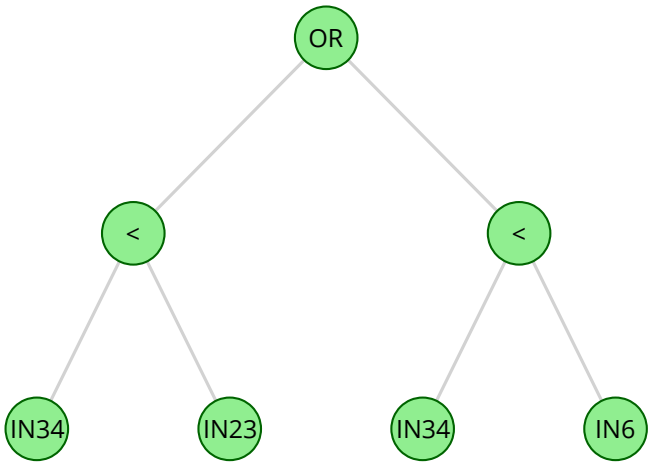


FIGURE 2.2 – Classifieur obtenu après 10 générations. (fitness sur la base d'apprentissage = 3066, fitness sur la base de test = 744)

Un exemple de calcul de la fitness pour le classifieur décrit dans la figure 2.2 avec un exemple d'apprentissage pris de la base Spambase. Il s'agit d'évaluer le résultat de l'expression $OR(IN34 < IN23, IN34 < IN6)$ en utilisant les valeurs

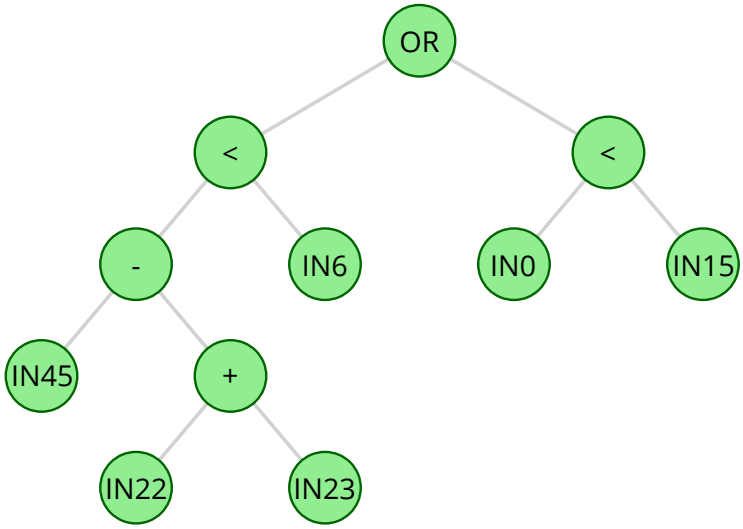


FIGURE 2.3 – Classifieur obtenu après 30 itérations. (fitness sur la base d'apprentissage = 3187, fitness sur la base de test = 791)

des attributs IN6, IN23 et IN34. Les valeurs des 57 attributs sont reportées dans l'extrait suivant :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0.21	0.28	0.5	0.0	0.14	0.28	0.21	0.07	0.0	0.94
10	0.21	0.79	0.65	0.21	0.14	0.14	0.07	0.28	3.47	0.0
20	1.59	0.0	0.43	0.43	0.0	0.0	0.0	0.0	0.0	0.0
30	0.0	0.0	0.0	0.0	0.0	0.0	0.07	0.0	0.0	0.0
40	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.132
50	0.0	0.372	0.18	0.048	5.114	101.0	1028.0	1.0		

Selon les valeurs extraites, nous avons : IN6=0.21, IN23=0.43 et IN34=0.0 (les valeurs soulignées dans l'extrait). Par la suite l'expression devient : $OR(0.0 < 0.43, 0.0 < 0.21)$ qui sera évalué à True ou 1.0. Ce qui revient à une prédiction de la classe spam. Cette prédiction est égale à la classe réelle (valeur doublement soulignée). Il s'agit d'une bonne classification qui sera traduite par l'incrémentation de la fitness de ce classifieur. Ce procédé est répété avec tous les exemples de la base d'apprentissage pour trouver la fitness de cet individu pour la génération en cours.

2.4 Apprentissage par GP à partir du Big Data

2.4.1 Défis de l'apprentissage à partir du Big Data

Nous vivons l'ère du « Big Data », dans laquelle le volume de données numérisées a atteint un niveau très important avec des prévisions d'augmentation pendant les prochaines années pour monter de $33ZO$ [2] en 2013 à environ $175ZO$ en l'année 2025 selon rapport de l'IDC [3] sponsorisé par Seagate [Reinsel *et al.* 2018]. Cette abondance de données brutes constitue un gisement de connaissances pour le domaine de l'apprentissage supervisé avec la panoplie d'outils d'extraction de connaissances qu'il englobe comme les réseaux de neurones artificiels, les arbres de décision, les machines à vecteur de support ...

Avec ce contexte, de nouveaux défis pour la fouille de données et l'apprentissage automatique sont apparus. De plus, d'anciens défis, dont les techniques de résolution ne sont pas adaptées au Big Data, refont surface. Plusieurs publications ont couvert ce sujet indépendamment de l'approche d'apprentissage automatique dans [L'Heureux *et al.* 2017, Zhou *et al.* 2017, Bhatnagar 2019] et particulièrement pour les algorithmes évolutionnaires dans [Bacardit & Llorà 2013].

Ces différents défis sont les liés aux attributs *Volume, Variété, Vélocité et Véracité, Valeur, etc.* du Big Data. Nous résumons ci-après quelques défis posés par les 3V (Volume, Variété et Vélocité).

2.4.1.1 Défis liés au Volume

C'est la taille des données utilisées dans le processus d'apprentissage. La taille concerne les dimensions horizontale (nombre d'attributs ou colonnes) et verticale (nombre d'exemples ou enregistrements). Parmi les défis les plus importants engendrés par la croissance du volume des sources Big Data nous citons :

- **Le coût d'apprentissage :** les ressources requises par un algorithme d'apprentissage en terme de temps processeur et espace mémoire augmentent avec le nombre d'exemples d'apprentissage. Cette thèse tente à apporter une réponse à ce défi concernant le temps d'apprentissage. Les solutions envisagées sont présentées dans la section 2.4.2.
- **La redondance de données :** avec les différentes sources et le grand nombre d'exemples d'apprentissage, la probabilité d'avoir des doublons est très élevée. La redondance a un effet néfaste sur la qualité de l'apprentissage.
- **La corrélation illusoire ou trompeuse :** avec une base d'apprentissage volumineuse, la corrélation faible qui existe entre deux événements devient plus importante induisant ainsi des erreurs d'apprentissage (faux positifs dans les problèmes de classification).

2. ZO : Zéta Octet = $10^{21} Octets$
3. International Data Corporation : https://www.idc.com/

- **Le déséquilibre de classes :** c'est un problème classique pour les problèmes de classification. Les techniques classiques doivent être adaptées pour le Big Data.
- **La dimensionnalité :** tandis que l'ingénierie d'attributs crée de nouvelles colonnes, la sélection d'attributs en réduit le nombre. La première vise l'amélioration de l'apprentissage et le seconde cherche les attributs les plus significatifs. La performance des anciennes techniques est annihilée par le problème de corrélation illusoire.

2.4.1.2 Défis liés à la Variété

La variété du Big Data caractérise la représentation, la sémantique et les sources des données. Cette variété est source de :

- **L'hétérogénéité des données :** utiliser plusieurs sources introduit une hétérogénéité syntaxique (types de données), sémantique (interprétation des données) ou encore statistique (propriétés statistiques).
- **Les données bruitées :** les valeurs erronées comme les points aberrants ou les valeurs manquantes constituent un problème bien connu pour l'apprentissage automatique. Il est encore plus complexe dans le cas Big Data.

2.4.1.3 Défis liés à la Vélocité

La vitesse des données générées et les nouvelles exigences quant à la vitesse de leur analyse posent plusieurs défis à l'apprentissage automatique dont :

- **La disponibilité des données :** avec la diffusion en continu (Streaming) des données, l'ensemble d'apprentissage n'est plus fourni en totalité à l'algorithme d'apprentissage. Ce dernier doit être capable de mettre à jour ses modèle à l'arrivée de nouveaux exemples d'apprentissage.
- **Le traitement temps réel/Streaming :** de nouvelles applications requièrent des réponses instantanées comme la détection de fraudes ou système de surveillance à base de réseau de capteurs.

2.4.2 Approches pour remédier au coût d'apprentissage

Nous nous intéressons, dans cette thèse, au problème du coût d'apprentissage lié à l'attribut *Volume* du Big Data. Manifestement, la croissance de la taille des données augmente le temps requis par les techniques d'apprentissage.

La programmation génétique GP, ayant montré ses preuves dans le contexte de l'apprentissage supervisé et plus particulièrement les problèmes de classification constitue un outil approprié pour ce contexte. La GP fait évoluer une population de classifieurs (des programmes génétiques) par le biais d'opérateurs génétiques (croisement, mutation, ...) au cours d'un nombre bien déterminé de générations. C'est la fonction de fitness qui détermine la qualité d'un classifieur et par conséquent la pérennité de son « patrimoine génétique » au moment de la phase de

sélection. Pour effectuer cette évaluation, chaque classifieur est exécuté en utilisant les données de chaque exemple de l'ensemble d'apprentissage. Ceci requiert un temps de calcul additionnel très important en particulier en présence de très larges bases de données. Ce temps est proportionnel au produit du nombre d'exemples d'apprentissage dans l'ensemble d'apprentissage, la taille de la population et le nombre de générations accomplies durant le processus évolutionnaire. En outre, l'exploitation de la totalité de l'ensemble d'apprentissage peut s'avérer impossible quand les données ne peuvent être chargées en mémoire ce qui est souvent le cas avec les bases qualifiées de Big Data.

Les approches préconisées pour contrecarrer les défis posés par le Big Data pour l'apprentissage sont assez diverses. Elles ont été regroupées dans [L'Heureux *et al.* 2017]. Une autre revue est élaborée dans [Bacardit & Llorà 2013] qui recense les méthodes et techniques pour accélérer les algorithmes évolutionnaires lors de leur application pour des tâches d'apprentissage.

Nous présentons, dans les paragraphes suivants, ces différentes approches en donnant des exemples de techniques pour pallier au problème de temps d'apprentissage engendré par le volume de données selon la classification faite dans [L'Heureux *et al.* 2017]. Nous concluons par énoncer les axes explorés dans cette thèse.

2.4.3 Mise à l'échelle par la manipulation des données, traitements et algorithmes

Ces manipulations sont effectuées au niveau des phases du pipeline d'analyse de données (figure 2.4).

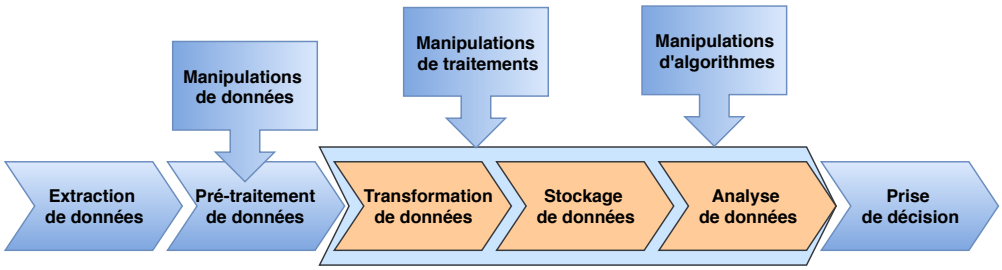


FIGURE 2.4 – Manipulations et phases d'analyse de données [L'Heureux *et al.* 2017].

– Manipulations de données : Elles sont appliquées dans la phase de pré-traitement des données avant de les administrer au processus d'apprentissage. Leur but est la réduction de la taille de la base d'apprentissage par exemple par des méthodes de réduction de dimension ou d'échantillonnage. L'échantillonnage, dans ce cas, est indépendant du processus d'apprentissage.

- Manipulations de traitements : Elles englobent les approches de parallélisation exploitant la nature parallèle de certains algorithmes comme la GP et les EA. La parallélisation peut être réalisée selon deux schémas :
  - ✓ Mise à l'échelle verticale : elle se base sur l'augmentation des ressources d'un nœud unique. Par exemple, l'accélération matérielle basée sur les processeurs graphiques appelés General Purpose Graphics Processing Units (GPGPU) et la parallélisation ont été utilisées dans plusieurs travaux [Harding & Banzhaf 2011, Langdon 2011, Maitre 2013].
  - ✓ Mise à l'échelle horizontale : elle représente les systèmes distribués où les traitements sont déployés sur un cluster de nœuds. C'est la forme de distribution la plus utilisée avec les problèmes Big Data. Plusieurs frameworks ont été mis en place pour deux types de parallélisation : par lot ou orienté flux. Les plus utilisés sont Hadoop MapReduce [Dean & Ghemawat 2004] et Apache Spark [Zaharia *et al.* 2012]. Dans le chapitre 5, nous explorons cette approche pour exécuter GP sur un cluster Spark.
- Manipulations d'algorithmes : dans cette catégorie, des modifications à un algorithme d'apprentissage sont apportées afin d'optimiser son temps d'exécution, d'améliorer sa qualité d'apprentissage, ... Ces modifications peuvent aller jusqu'à intégrer un nouveau paradigme. Nous avons mis en œuvre cette technique en adaptant une implémentation existante de GP pour être exécutée sous Spark (voir chapitre 5).

2.4.4 Mise à l'échelle par le paradigme d'apprentissage

Nous avons présenté dans la section 2.2 plusieurs approches et paradigmes d'apprentissage. Chaque paradigme est plus approprié pour certains types de problèmes. Parmi ceux qui permettent de réduire le coût d'apprentissage nous pouvons citer les ensembles d'apprentissage [Dietterich 2000, Tang *et al.* 2016] et l'apprentissage actif (voir section 2.2.6). L'apprentissage actif par échantillonnage de la base d'apprentissage est étudié dans ce travail.

2.4.5 Approches étudiées

Nous nous sommes fixé l'objectif de résoudre le problème du temps de calcul de GP quand elle est appliquée à un problème d'apprentissage à partir de très larges bases de données. Étant donné que le coût d'apprentissage par GP est dû principalement à la phase d'évaluation, deux voies sont considérées, dans cette thèse, pour alléger ce coût : l'accélération de ces évaluations ou la réduction de leur nombre.

Dans [Bacardit & Llorà 2013], une classification des approches à considérer lors de l'utilisation des EA pour l'apprentissage à partir de larges bases de données est donnée, elle dénombre quatre familles :

1. Les solutions logicielles : toute solution qui n'exige aucun matériel spécifique. Cette famille inclut essentiellement les techniques d'échantillonnage, l'exploitation de la régularité des données, l'extension des opérateurs génétiques et la substitution de fitness par des alternatives moins coûteuses.
2. Les techniques d'accélération matérielle : elles correspondent aux techniques de mise à l'échelle verticale du paragraphe précédent.
3. Les modèles de parallélisation classiques : c'est un sous-ensemble des méthodes de mise à l'échelle horizontale où la communication entre les nœuds est limitée.
4. Les modèles de parallélisation pour données massives (data-intensive) : ce sont les nouveaux modèles de parallélisation introduites dans MapReduce et Spark.

Dans cette thèse deux approches sont explorées. La première est une **solution logicielle** qui opère une mise à l'échelle par le **paradigme de l'apprentissage actif** en introduisant des méthodes d'**échantillonnage** de l'ensemble d'apprentissage afin de réduire le nombre d'évaluations. Tandis que la seconde approche consiste à la **parallélisation** de l'évaluation dans le framework **Spark** afin d'en réduire le temps (mise à l'échelle horizontale et manipulation des traitements).

2.5 Conclusion

Après une introduction de l'apprentissage automatique, nous avons montré comment GP représente une méthode convenable particulièrement pour le problème de classification supervisée selon l'approche Pittsburgh. Par ailleurs, le contexte Big Data pose des défis à cette application dont le plus sévère est celui du coût d'apprentissage lié au grand volume de données. Nous avons opté pour l'exploration de deux approches pour remédier à ce problème. La première voie fait l'objet des chapitres 3 et 4. Le chapitre 5 est consacré à la deuxième approche.

CHAPITRE 3

Échantillonnage pour la GP

Sommaire

3.1 Introduction 65
3.2 Échantillonnage dans la boucle GP 66
3.3 Taxonomie 67
3.3.1 Fréquence d'échantillonnage 68
3.3.2 Stratégie d'échantillonnage 68
3.3.3 Quantité d'échantillonnage 68
3.3.4 Classification des approches d'échantillonnage 69
3.4 Les approches de l'échantillonnage 70
3.4.1 Échantillonnage statique 70
3.4.2 Échantillonnage actif 72
3.4.3 Discussion 84
3.5 Étude comparative 86
3.5.1 Librairie évolutionnaire ECJ 86
3.5.2 Base d'apprentissage 86
3.5.3 Configuration GP 88
3.5.4 Implémentation des algorithmes d'échantillonnage 88
3.6 Résultats 91
3.7 Discussion 93
3.8 Conclusion 95

3.1 Introduction

L'échantillonnage de l'ensemble d'apprentissage a été initialement introduit pour améliorer le processus d'apprentissage et éviter le sur-apprentissage (overfit-

ting) [Iba 1999, Liu & Khoshgoftaar 2004, Paris *et al.* 2003]. Plus tard, il a été adopté comme une stratégie pour la GP dans le but de faire face aux problèmes ayant une large base de données. Une large variété de méthodes d'échantillonnage a été explorée avec de nombreux problèmes de classification par la GP et a obtenu de meilleurs résultats par rapport à l'utilisation de la totalité de l'ensemble d'apprentissage. Cependant, les résultats obtenus à partir des différentes expérimentations ne peuvent pas être comparés car ces dernières sont basées sur des formes différentes de GP (GP linéaire, GP standard, GP cartésiennes, ...), des configurations différentes des paramètres de GP et des données différentes. Avec l'explosion Big Data, les larges bases de données sont devenues abondantes. Cette quantité permet aux classifieurs d'apprendre à partir de données très variées (Volume et Variété) et ainsi améliorer leur précision. Néanmoins, leur taille de plus en plus importante aggrave le problème du temps de calcul pour la GP.

Ce chapitre est consacré à l'étude des méthodes d'échantillonnage dites actives dans la cadre de l'apprentissage supervisé avec la GP. Nous commençons par donner une revue étendue des algorithmes d'échantillonnage tel que décrits par leurs auteurs. Nous introduisons une taxonomie pour les catégoriser et discuter leur aptitude à gérer les bases de données massives. Puis, nous implémentons un nombre important de ces algorithmes pour une étude expérimentale et comparative. Nous avons utilisé, pour cette étude, la base KDD pour la détection d'intrusion [UCI 1999a], contenant 494021 exemples d'apprentissage répartis sur 4 classes d'attaques et une classe normale.

3.2 Échantillonnage dans la boucle GP

Dans la boucle évolutionnaire (figure 1.1), c'est l'évaluation de la population qui est à l'origine du problème de coût de calcul. Ceci est plus accentué avec les problèmes d'apprentissage supervisé. En effet, dans cette étape, tous les individus (programmes) sont exécutés un nombre d'itérations égal à la taille de la base d'apprentissage. La figure 3.1 décortique l'évaluation d'un individu sur un exemple d'apprentissage. Son phénotype est $(if(IN3 > 0.3, IN3, IN0 + IN1) < 0.6)$ qui est représenté par l'arbre d'expression dans la même figure. Selon l'expression, les valeurs sont lues à partir de chaque exemple d'apprentissage (illustrée par une ligne du tableau). Le résultat est traduit en une prédiction de classe dans le cas d'une classification puis comparé à la classe réelle fournie par la base d'apprentissage. La valeur de fitness est calculée après avoir testé tout l'ensemble d'apprentissage selon la fonction de fitness adoptée (par exemple le taux de bonnes classifications).

Le temps d'évaluation est fonction de : la complexité de l'individu (son expression), la taille de la population et la taille de la base d'apprentissage.

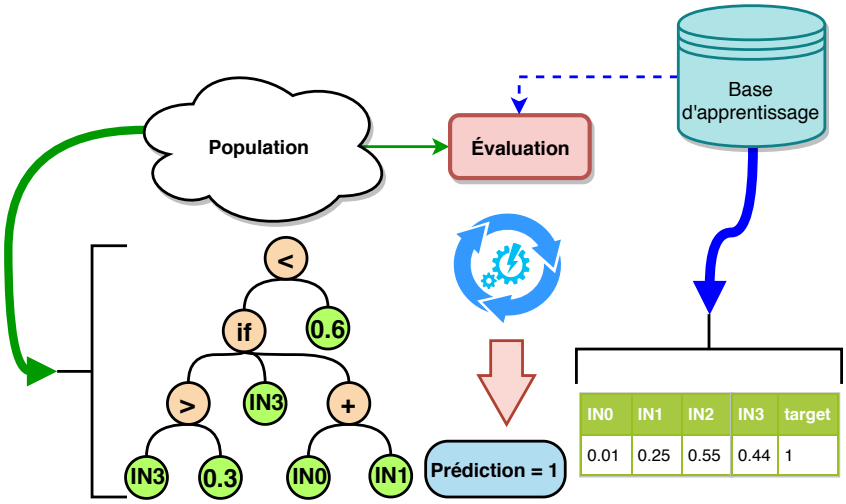


FIGURE 3.1 – Évaluation dans un problème de classification.

Avec les problèmes à très large bases d'apprentissage, le volume de données est très important par rapport aux deux autres facteurs. L'échantillonnage intervient dans cette phase en réduisant le nombre d'exemples d'apprentissage à tester pour évaluer la fitness d'un individu. Nous proposons, dans la section suivante (section 3.3), une taxonomie de l'échantillonnage de la base d'apprentissage avant d'examiner en détails certains algorithmes utilisés avec GP (section 3.4).

3.3 Taxonomie

Soit $\mathcal{B}$ la base d'apprentissage d'origine contenant tous les exemples pour le processus d'apprentissage. Formellement, une méthode d'échantillonnage est la sélection d'un sous-ensemble d'exemples d'apprentissage à partir de $\mathcal{B}$ pour créer un échantillon $\mathcal{S}$ où $\mathcal{S} \subset \mathcal{B}$ et $|\mathcal{S}| \ll |\mathcal{B}|$.

Pour concevoir une technique d'échantillonnage, trois décisions conceptuelles sont nécessaires :

(1) *fréquence d'échantillonnage* : combien de fois le sous-ensemble d'apprentissage est renouvelé durant le processus d'apprentissage (pour la phase d'évaluation) ;

(2) *stratégie d'échantillonnage* : comment sélectionner les exemples d'apprentissage pour construire l'échantillon $\mathcal{S}$ à partir de la base de données $\mathcal{B}$;

(3) *quantité d'échantillonnage* : combien d'échantillons sont requis pour la phase d'évaluation.

3.3.1 Fréquence d'échantillonnage

Pour l'apprentissage avec GP, il est possible d'utiliser un unique échantillon $\mathcal{S}$ pour évaluer tous les individus pendant la totalité du run GP. C'est la fréquence d'échantillonnage *par-run* (*run-wise*) [Freitas 2002]. Cette approche d'échantillonnage est appelée **échantillonnage statique** discuté plus loin. Quand l'algorithme d'apprentissage utilise plusieurs échantillons différents, créés à des instants éparses tout au long d'un run, c'est la fréquence η-*générations* (η-*generations-wise*) qui est appliquée. Dans ce cas, toutes les η générations, un nouveau sous-ensemble $\mathcal{S}$ est extrait et utilisé lors de l'évaluation, où $\eta \geq 1$. Si $\eta = 1$, la population est évaluée sur un échantillon différent chaque génération c'est un échantillonnage *par-génération* (*generation-wise*). Les méthodes de cette catégorie sont aussi connues comme des technique d'**échantillonnage actif** ou **dynamique** (section 3.4.2).

3.3.2 Stratégie d'échantillonnage

Pour sélectionner un sous-ensemble d'apprentissage $\mathcal{S}$ de $\mathcal{B}$, plusieurs approches ont été proposées pour l'échantillonnage statique ou actif. Concernant l'échantillonnage statique, la base d'apprentissage est partitionnée avant le processus d'apprentissage, en se basant essentiellement sur des critères ou des caractéristiques propres aux données étudiées. Cette stratégie n'est pas considérée dans notre cadre d'étude ; quelques techniques sont toutefois exposées dans la section 3.4.1. Quant à l'échantillonnage actif (section 3.4.2), nous définissons 5 principales approches : échantillonnage aléatoire ou stochastique, échantillonnage pondéré, échantillonnage basé sur une topologie des données, échantillonnage balancé et échantillonnage incrémental. Une autre approche consiste à combiner quelques techniques appartenant aux précédentes approches.

Le point commun entre les techniques de l'échantillonnage actif est que le sous ensemble d'apprentissage est changé de manière dynamique en parallèle avec le processus d'apprentissage, et ce en s'appuyant sur différents paramètres reflétant l'état de l'apprentissage (figure 3.2). Cependant, quelques stratégies ne sont pas bien adaptées pour le cas des bases d'apprentissage assez volumineuses, tel que certaines techniques d'échantillonnage entrelacé (voir paragraphe *Autres techniques d'échantillonnage* dans 3.4.2.1). Ces techniques ne sont pas incluses dans notre étude expérimentale.

3.3.3 Quantité d'échantillonnage

La troisième décision pour la mise au point d'une technique d'échantillonnage est le nombre d'échantillons à fournir à GP à chaque fois que la procédure d'échantillonnage est appelée. Avec l'apprentissage par un EA, trois scénarios sont à considérer :

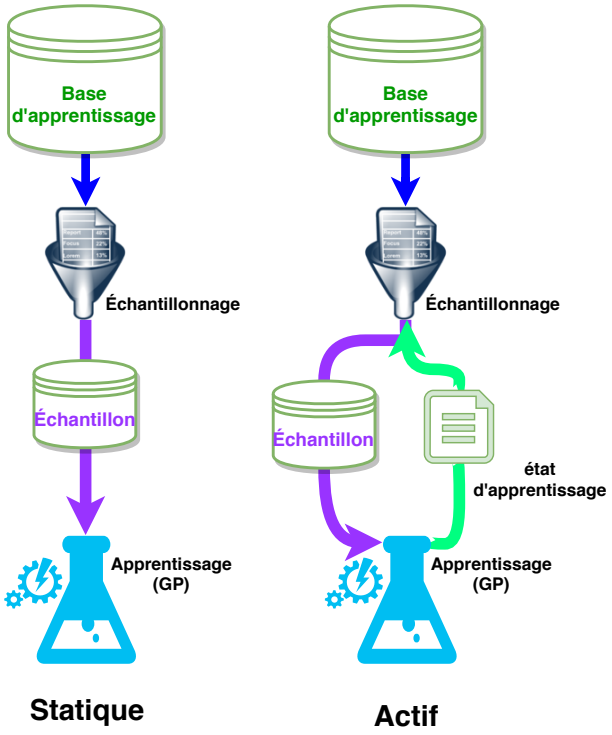


FIGURE 3.2 – Échantillonnage statique et actif.

- *par-individu* (un échantillon par individu (*individual-wise*)) : dans ce cas, un échantillon $\mathcal{S}_j$ est sélectionné comme base d'apprentissage pour chaque individu de la population et chaque classifieur est évalué indépendamment ;
- *par-population* (un échantillon par population (*population-wise*)) : un échantillon unique est créé pour être utilisé avec tous les individus de la population ;
- *par-sous-population* (un échantillon par sous-population (*subpopulation-wise*)) : si l'EA sous-jacent fait évoluer plusieurs sous-populations avec un mécanisme co-évolutionnaire, il est possible de créer un échantillon pour chaque sous-population.

3.3.4 Classification des approches d'échantillonnage

En examinant les trois choix conceptuels décrits au-dessus ainsi que les différentes techniques qui en découlent, certaines semblent plus aptes à traiter des problèmes à données massives en conservant la qualité d'apprentissage. Avant de décider des paramètres pour chaque stratégie, il est important d'étudier l'utilité et l'efficacité de chaque alternative par rapport au problème considéré et ses objectifs.

Par exemple, les techniques qui sont *par-run*/*par-population* font évoluer la population en l'évaluant avec un échantillon fixe et peuvent conduire vers le sur-apprentissage. Ces techniques sont, généralement, inadaptées pour les larges bases

de données. De plus, comme les méthodes *par-génération/par-individu* évaluent les individus par rapport aux différents échantillons créés à chaque génération, le coût de l'évaluation risque d'augmenter de manière drastique. Par conséquent, ces méthodes aussi ne conviennent pas aux problème à grande échelle. Pour résumer, il est recommandé d'éviter toute stratégie susceptible d'augmenter le coût de calcul de l'apprentissage. Les méthodes à privilégier sont donc celles qui réduisent le nombre d'évaluations à effectuer à travers le contrôle de la fréquence et/ou la quantité d'échantillonnage.

La possibilité de combiner plusieurs méthodes d'échantillonnage a conduit à l'introduction de deux principales catégories :

- Méthodes à 1-niveau : méthodes utilisant une unique stratégie de sélection basée sur un critère dynamique, comme la sélection aléatoire, pondérée, ...
- Méthodes multi-niveaux ou hiérarchiques : méthodes combinant plusieurs techniques de la première catégorie de manière hiérarchique. C'est-à-dire que le résultat d'une méthode est l'entrée de celle qui la suit.

Nous pouvons regrouper les différentes méthodes d'échantillonnage sous la classification proposée dans la figure 3.3. Ces méthodes sont détaillées dans la section 3.4.

3.4 Les approches de l'échantillonnage

Dans cette section, nous donnons une description de différentes méthodes d'échantillonnage appartenant aux deux catégories statique et active. Ces algorithmes ont été utilisés avec différentes variantes de GP.

3.4.1 Échantillonnage statique

Avec l'échantillonnage statique, chaque individu de la population obtient le sous-ensemble ou l'échantillon au début du processus d'apprentissage et celui-ci sera préservé sans aucune modification tout au long de l'évolution. Toutes les méthodes de cette catégorie ont une fréquence d'échantillonnage *par-run* et sont *par-population* ou *par-sous-population*. Ce type d'échantillonnage construit un échantillon d'une taille prédéfinie. Les exemples d'apprentissage retenus dans cet échantillon sont sélectionnés indépendamment de l'évolution de l'apprentissage. Certaines méthodes utilisent un sous-ensemble unique pour tous les runs, d'autres peuvent affecter un sous-ensemble différent pour chaque run.

Ci-après, nous donnons une brève description de quelques techniques de cette catégorie.

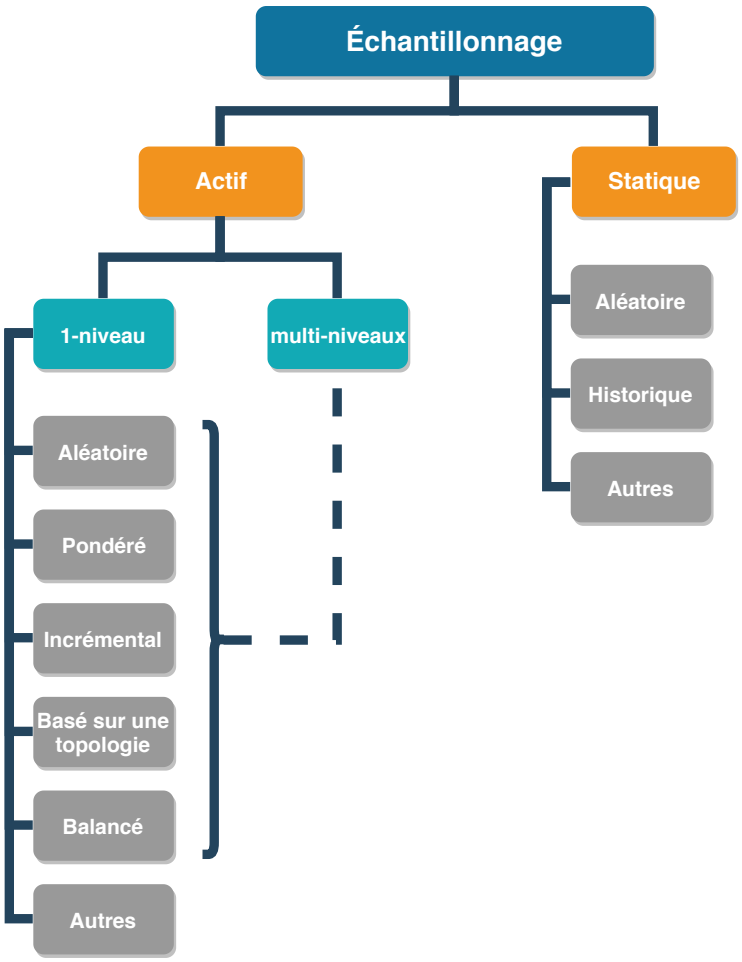


FIGURE 3.3 – Taxonomie des approches d'échantillonnage pour GP.

Échantillonnage aléatoire statique : La technique la plus simple est le *Holdout* qui consiste à subdiviser la base d'apprentissage en deux ensembles l'un pour l'apprentissage et l'autre pour le test. Cette technique a été utilisée dans les premières implémentations de GP comme outil d'apprentissage automatique, elle est encore utilisée pour les problèmes avec une petite base de données.

Le *Static Random Sampling* est une variation du *Holdout* dans laquelle le *Holdout* est répété k fois. L'ensemble de test peut être aussi un sous-ensemble aléatoire ou composé des exemples d'apprentissage non sélectionnés pour la phase d'apprentissage. Chaque échantillon est utilisé pour plusieurs runs. Le processus d'apprentissage est suivi d'une procédure de sélection du modèle. Elle est basée sur des tests de signification statistique appliqués aux meilleurs modèles.

Échantillonnage historique : Pour améliorer cette approche pour la GP, Gathercole et Ross ont suggéré l'échantillonnage historique (*Historical Subset Selection*) **HSS** [Gathercole & Ross 1994]. Il regroupe les exemples d'apprentissage provo-

quant une erreur de classification par le meilleur individu à chaque génération. Ces exemples d'apprentissage étant considérés comme des cas difficiles. Le sous-ensemble contient aussi des cas faciles (correctement prédits) pris à partir des toutes premières générations. L'échantillon final est obtenu après plusieurs runs effectués sur la totalité du l'ensemble d'apprentissage.

Boosting/bagging : Bagging, (ou bootstrap aggregating) est à l'origine proposé par Breiman pour les problèmes de classification [Breiman 1996]. Cette technique provenant de l'apprentissage automatique a été appliquée à la GP dans [Iba 1999, Paris *et al.* 2001] plutôt pour améliorer la qualité du résultat de la GP que pour accélérer le temps du processus évolutionnaire. Elle consiste à faire évoluer plusieurs sous-populations puisant leurs données dans des ensembles d'apprentissage différents (échantillonnage de type par-sous-population). Ces derniers sont générés par sélection à partir de la base initiale avec remise. Chaque exemple d'apprentissage a un poids qui reflète sa difficulté afin que ceux qui sont les plus difficiles apparaissent plus fréquemment vu que la taille cible est égale à la taille de la base initiale. Les méthodes proposées appelées BagGP et BoostGP ont inspiré l'échantillonnage pondéré discuté dans la section 3.4.2.

3.4.2 Échantillonnage actif

Les méthodes d'échantillonnage actif ou dynamique sont aussi appelées les méthodes de sélection active de données. Ce nom trouve ses racines dans l'*Apprentissage Actif* (voir la section 2.2.6).

Dans cette catégorie, les techniques d'échantillonnage sont étroitement liées à l'évolution du processus d'apprentissage. Plus généralement, l'échantillonnage dépend d'un ou plusieurs aspects bien déterminés du processus d'apprentissage comme les exemples difficiles, le nombre de générations opérées, la fitness . . . Selon ces aspects, les principales méthodes d'échantillonnage sont subdivisées en sous-classes : méthodes aléatoires, méthodes pondérées, hiérarchiques . . . La suite de cette section présente les principales méthodes appartenant à ces sous-classes.

3.4.2.1 Échantillonnage 1-niveau

Échantillonnage aléatoire : La sélection des exemples d'apprentissage, avec ce type d'échantillonnage, est basée sur une probabilité uniforme à partir de l'ensemble d'apprentissage et selon le schéma *par-génération*. Cette sélection stochastique permet de réduire le biais dans l'ensemble d'apprentissage. L'algorithme *Random Subset Selection* **RSS** [Gathercole & Ross 1994] fait partie de cette famille d'échantillonnage (cf. Algorithme 2). Dans RSS, la probabilité de choisir un exemple d'apprentissage i lors d'une génération g est égal à $P_i(g)$ tel que :

$$\forall i : 1 \leq i \leq |\mathcal{B}|, \quad P_i(g) = \frac{|\mathcal{S}|}{|\mathcal{B}|}. \tag{3.1}$$

où $|\mathcal{B}|$ est la taille de tout l'ensemble d'apprentissage et $|\mathcal{S}|$ la taille cible pour le sous-ensemble d'apprentissage. En utilisant cette probabilité, le sous-ensemble échantillonné aura une taille qui fluctue autour de $|\mathcal{S}|$.

Algorithme 2 RSS

1: Créer le sous-ensemble $\mathcal{S}(0)$ par sélection des exemples de $\mathcal{B}$ avec une probabilité uniforme $= \frac{|\mathcal{S}|}{|\mathcal{B}|}$
2: $g \leftarrow 0$
3: **tant que** génération $g < g_{max}$ **faire**
4: Évaluer les individus avec le sous-ensemble $\mathcal{S}(g)$
5: Appliquer les opérateurs génétiques
6: Générer aléatoirement le sous-ensemble $\mathcal{S}(g+1)$ avec la même probabilité dans 1
7: **fin tant que**

Fixed Random Selection (**FRS**) [Zhang & Joung 1999] est un autre exemple de méthode aléatoire très similaire à RSS avec un nombre fixe d'exemples d'apprentissage qui sont sélectionnés à chaque génération avec une probabilité constante :

$$\forall i : 1 \leq i \leq |\mathcal{B}|, \quad P_i(g) = \frac{1}{|\mathcal{B}|}. \tag{3.2}$$

Stochastic Sampling **SS** [Nordin & Banzhaf 1997] est aussi une méthode qui utilise la même sélection probabiliste pour créer des sous-ensembles d'apprentissage et ce pour chaque individu par génération (type *par-individu, par-génération*).

L'unique paramètre de cette catégorie d'échantillonnage est la taille du sous-ensemble, que ce soit fixe ou flexible (comme le cas du RSS). Ce paramètre est ajusté comme tous les paramètres de la GP (taille de la population, probabilité de croisement, ...) de façon expérimentale à travers plusieurs runs. D'une part, choisir une très petite taille mène la GP au sur-apprentissage et par conséquent les meilleurs individus obtenus ne peuvent pas être généralisés. D'autre part, un très grand sous-ensemble demande plus de temps de calcul pour l'évaluation des individus de la population.

Échantillonnage pondéré : La sélection des exemples d'apprentissage à retenir dans le sous-ensemble est basée sur un poids calculé pour chaque exemple ce qui est différent d'une probabilité uniforme comme dans le cas des méthodes aléatoires. Ce poids mesure à quel point cet exemple mérite d'être considéré lors du processus d'apprentissage et peut aider à améliorer la qualité de la population. Ce principe est repris de la technique de boosting de l'apprentissage automatique, initialement utilisée pour améliorer la précision des apprenants faibles.

Le tout premier algorithme dans cette catégorie est *Dynamic Subset Selection* **DSS** [Gathercole & Ross 1994, Gathercole & Ross 1997, Gathercole 1998]. Cet algorithme a été élaboré pour assurer la consistance de l'ensemble d'apprentissage

tout en allégeant sa taille : on garde uniquement les exemples difficiles et ceux non sélectionnés pendant un certain nombre de générations pour faire partie de l'ensemble d'apprentissage. Chaque exemple de la base d'apprentissage se voit attribuer un degré de difficulté $D_i(g)$ et une ancienneté ou âge $A_i(g)$ commençant par 0 lors de la première génération et qui sont actualisés à chaque nouvelle génération. La difficulté est ainsi incrémentée à chaque erreur de classification commise par un classifieur et elle remise à 0 si cet exemple d'apprentissage a été sélectionné par l'algorithme d'échantillonnage. L'âge est égal au nombre de générations passées depuis la dernière sélection de l'exemple d'apprentissage, alors il est incrémenté si l'exemple n'a pas été choisi et remis à 0 sinon.

Le poids résultant W du $i^{ème}$ exemple est calculé avec la somme suivante :

$$\forall i : 1 \leq i \leq |\mathcal{B}|, \quad W_i(g) = D_i(g)^d + A_i(g)^a. \tag{3.3}$$

$$D_i(0) = A_i(0) = 0. \tag{3.4}$$

où d est l'exposant de difficulté, a est l'exposant de l'âge que Gathercole fixe de façon à ce que les exemples d'apprentissage âgés de 15 générations ont le même poids que les exemples les plus difficiles.

La probabilité de sélection est biaisée par la difficulté et l'âge :

$$\forall i : 1 \leq i \leq |\mathcal{B}|, \quad P_i(g) = \frac{W_i(g) * |\mathcal{S}|}{\sum_{j=1}^{|\mathcal{B}|} W_j(g)}. \tag{3.5}$$

DSS nécessite l'ajustement de trois paramètres : l'exposant de difficulté, l'exposant de l'âge et la taille cible. L'algorithme 3 décrit les différentes étapes de DSS.

Gathercole [Gathercole & Ross 1994] a testé plusieurs tailles de sous-ensembles ainsi que plusieurs tailles de populations avec le problème de classification de la thyroïde[1]. Ce problème a une base d'apprentissage de taille totale de 3772. Les expériences montrent que quand la taille du sous-ensemble approche ou est inférieur à 200, les performances chutent remarquablement et au-dessus de 400 aucune amélioration n'est détectée. De plus, avec une population de 10000 individus DSS réalise de meilleurs résultats que les réseaux de neurones. Enfin, DSS permet de réduire les évaluations de 20% sans perdre en qualité.

Échantillonnage incrémental : Avec cette catégorie, le processus d'apprentissage commence par un ensemble d'apprentissage contenant quelques exemples (un paramètre à fixer) et accumule progressivement des exemples supplémentaires à chaque génération. Ce processus est répété, le plus souvent, jusqu'à utiliser l'intégralité de la base d'apprentissage. Elle suit le type *par-population* ou *par-individu*.

C'est la cas de Incremental Random Selection (**IRS**) [Zhang & Cho 1999] qui agrandit l'échantillon initial à chaque génération par l'ajout d'un même nombre d'exemples d'apprentissage sélectionnés de manière aléatoire.

1. Voir UCI Machine Learning Repository `http://archive.ics.uci.edu/ml/`

Algorithme 3 DSS

Paramètres : a (exposant de l'âge), d (exposant de la difficulté), $T = |\mathcal{S}|$ (taille cible)

Initialiser pour tout exemple i de $\mathcal{B}$ le degré de difficulté $D_i(0)$ et l'âge $A_i(0)$
$g \leftarrow 0$
tant que génération $g < g_{max}$ **faire**
  Initialiser un sous-ensemble vide $S(g)$
  pour tout exemple d'apprentissage i dans $\mathcal{B}$ **faire**
    si g=0 **alors**
      Calculer $P_i(g)$ avec l'équation (3.1)
    sinon
      Calculer $P_i(g)$ avec l'équation (3.5)
    fin si
    Ajouter l'exemple i à $calS(g)$ avec la probabilité $P_i(g)$
    si i est sélectionné **alors**
      $A_i(g+1) = 0$
    sinon
      $A_i(g+1) = A_i(g) + 1$
    fin si
  fin pour
  Évaluer la population sur le sous-ensemble $\mathcal{S}(g)$
  mettre à jour les degrés de difficultés des exemples de la base d'apprentissage $D_i(g+1)$
  Appliquer les opérateurs génétiques
fin tant que

Zhang et al. [Zhang & Cho 1999, Zhang & Joung 1999] proposent d'effectuer ce qu'ils appellent croisement uniforme de données (uniform data crossover) simultanément avec le croisement des individus lors de l'évolution des programmes génétiques. Cela signifie, qu'au moment d'appliquer l'opérateur génétique de croisement pour créer de nouveaux programmes ou individus, leurs sous-ensembles sont aussi croisés pour obtenir de nouveaux sous-ensembles dérivés induisant un héritage de données à travers les générations. Ce mécanisme permet de sauvegarder les connaissances acquises par les parents. Cette méthode appelée *Incremental Data Inheritance* **IDI**, peut être simplifiée dans les étapes illustrées par l'algorithme 4.

Dans ce qui suit un exemple pratique illustre les différentes phases de cet algorithme. Supposons que la base d'apprentissage initiale est constituée des lettres alphabétiques et les valeurs des différents paramètres utilisés dans IDI : $n_0 = 5$, $\lambda = 3$ et $\rho = 0.5$. Le principe du croisement de données est repris dans l'exemple schématisé par la figure 3.4.

Pour la génération 0, les sous-ensembles alloués à chaque individu sont construits après sélection aléatoires de 5 lettres. Soit i et j deux individus ayant

Algorithme 4 IDI

Paramètres : n_0 (taille de l'échantillon initial), λ (incrément), ρ (portion à importer de la base d'apprentissage d'origine), g_{max} (nombre de générations maximal)

1: Affecter un sous-ensemble $D_i(0)$ à chaque individu i avec une taille initiale n_0 par sélection aléatoire depuis la base d'apprentissage
2: **tant que** génération $g < g_{max}$ **faire**
3: Évaluer les individus avec leurs sous-ensembles $D_i(g)$ de taille n_g
4: Croiser les parents et leurs sous-ensembles respectifs $D_i(g)$ et $D_j(g)$ pour générer les sous-ensembles des enfants $D_i(g+1)$ et $D_j(g+1)$:
5: Fusionner les sous-ensembles dans $D_{i+j}(g) = D_i(g) \cup D_j(g)$
6: Calculer *facteur de diversité* : $d_i = \frac{|D_{i+j}(g)|}{|D_i(g)|} - 1, \quad 0 \leq d_i \leq 1$
7: Calculer *taux d'import* : $r_i = \rho * (1 - d_i), \quad 0 \leq \rho \leq 1$
8: Générer $D_i(g+1)$ et $D_j(g+1)$ de taille $n_{g+1} = n_g + \lambda$ utilisant $((1 - r_i) * n_g)$ exemples de $D_{i+j}(g)$ et $(r_i * n_g + \lambda)$ nouveaux exemples de la base initiale par sélection aléatoire
9: **fin tant que**

été sélectionnés par la GP comme parents qui seront croisés pour créer deux enfants pour la génération 1. Afin de construire deux sous-ensembles pour les enfants de i et j, IDI commence par calculer l'union des sous-ensembles des parents notée $D_{i+j}(0)$ et qui possède dans ce cas 8 lettres et par conséquent $r_i = 0.5 * (1 - (\frac{8}{5} - 1)) = 0.2$. La taille des nouveaux sous-ensembles doit être égale à $n_0 + \lambda = 5 + 3 = 8$ lettres. Ces 8 lettres sont choisies aléatoirement avec les proportions suivantes :

- $(1 - r_i) * n_0 = 0.8 * 5 = 4$ lettres à partir $D_{i+j}(0)$
- $r_i * n_0 + \lambda = 0.2 * 5 + 3 = 4$ à partir de la base initiale (l'alphabet).

Le paramètre λ est souvent ajusté de telle sorte que lors de la dernière génération, tous les individus utiliseront la totalité de la base d'apprentissage.

Pour appliquer cet algorithme, trois paramètres doivent être ajustés : n_0 (taille initiale), λ et ρ. En association avec IDI, Zhang utilise une fonction de fitness adaptative (*Adaptative Fitness Function*) qui varie d'un individu à un autre [Zhang & Joung 1999]. Cette méthode d'échantillonnage a été appliquée pour l'évolution des comportements collectifs d'agents robotiques [Zhang & Cho 1999] et comparée avec la GP standard et aussi GP avec IRS. La base d'apprentissage est composée de 100 exemples d'apprentissage. Dans une deuxième étude [Zhang & Joung 1999], IDI a été confronté à l'utilisation de toute la base d'apprentissage et à l'algorithme FRS dans le cadre du problème de prédiction des séries temporelles. Les deux études expérimentales soutiennent l'affirmation que l'évolution de programmes sur des sous-ensembles sélectionnés de manière incrémentale peut réduire significativement le temps de l'évaluation de la fitness sans pour autant perdre la précision de généralisation de ces programmes.

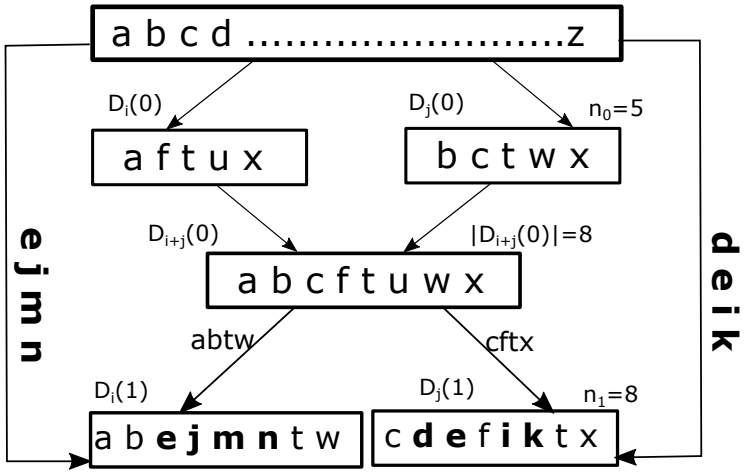


FIGURE 3.4 – Exemple de croisement de données dans IDI.

Échantillonnage basé sur une topologie : Inspirée par l'idée que la structure influence l'efficacité des heuristiques opérant sur elle [Hogg 1996, Walsh 1999], cette méthode consiste à construire une topologie du problème à partir des connaissances acquises par les individus à propos des exemples d'apprentissage.

Lazarczyk [Lasarczyk *et al.* 2004] préconise une méthode d'échantillonnage basée sur la topologie des exemples d'apprentissage appelée *Fitness case Topology Based Sampling* **TBS** dans laquelle la relation entre les exemples d'apprentissage est représentée par un graphe non orienté et valué. Les sommets de ce graphe représentent les exemples d'apprentissage et les arêtes possèdent un poids qui mesure une similarité ou une distance induite par les individus de la population. Par la suite, les exemples ayant une relation forte par rapport à un seuil ne peuvent être sélectionnés dans le même sous-ensemble en supposant qu'ils ont le même degré de difficulté ou encore qu'ils procurent des connaissances équivalentes. Les poids des arêtes commencent à 0 et sont mis à jour après la phase d'évaluation en deux étapes (cf. Algorithme 5).

La seconde étape est exécutée afin de relâcher la relation entre les exemples d'apprentissage détectée dans les précédentes générations. La valeur de λ est déterminée expérimentalement et est généralement comprise entre 0.5 et 0.9.

Cet algorithme utilise un seul sous-ensemble pour tous les individus. Ce sous-ensemble est renouvelé à chaque génération après la mise à jour du graphe de la topologie. La sélection selon TBS est décrite dans Algorithme 6.

Le seuil utilisé dans TBS est calculé durant la sélection dans le but d'avoir une valeur adaptée à la topologie en cours. Le seuil ne doit pas être très élevé ou encore trop bas. Dans le premier cas, il n'y aura pas d'exclusion de candidats dans la phase de sélection et alors TBS sera équivalent à un échantillonnage aléatoire. Dans l'autre cas, le seuil est très sélectif et peu d'exemples seront choisis ce qui ne permet pas d'atteindre la taille cible. Le seuil est calculé avec un algorithme dichotomique (Algorithme 7).

Algorithme 5 Mise à jour du graphe TBS

1: **pour tout** individu **faire**
2: mettre à jour le poids des arêtes :

$$\forall e \in E' \quad W_e = W_e + \frac{2}{|V'|\,(|V'|-1)}. \tag{3.6}$$

V' : l'ensemble des exemples correctement prédits par cet individu.
E' : l'ensemble des arêtes entre tous les exemples de V'.
3: Appliquer un taux de perte λ pour toutes les arêtes :

$$\forall e \in E \quad W_e = \lambda * W_e, \lambda < 1. \tag{3.7}$$

4: **fin pour**

Algorithme 6 Sélection TBS

1: Vider le sous-ensemble sélectionné et initialiser l'ensemble des exemples candidats par toute la base d'apprentissage
2: **tant que** l'ensemble des candidats est non vide et la taille cible non atteinte **faire**
3: Choisir aléatoirement un exemple d'apprentissage de l'ensemble des candidats
4: Supprimer l'exemple choisi et tous les exemples qui y sont connectés avec une arête de poids > seuil
5: **fin tant que**

Les résultats expérimentaux sur des problèmes de classification (le problème des spirales entrelacées [Lang & Witbrock 1988] et le problème de la thyroïde) montrent des améliorations dans la valeur moyenne de fitness au cours des générations par rapport à DSS et SS. Cependant, le coût de calcul supplémentaire de TBS engendré par le calcul du seuil adaptatif et aussi la mise à jour de la topologie ne sont pas mesurés. De plus, cette méthode n'a pas été testée sur une large base d'apprentissage.

Échantillonnage balancé ou encore équilibré [Hunt *et al.* 2010] : c'est une méthode qui vise à améliorer la précision des classifieurs à travers la remédiation au déséquilibre entre les exemples d'apprentissage de la classe majoritaire et ceux de la classe minoritaire dans la base d'apprentissage originale. Certains algorithmes utilisent la taille de la classe minoritaire comme taille de référence, ce qui permet de réduire le nombre d'exemples dans l'échantillon (voir le tableau 3.1). Les méthodes suivantes ont été appliquées avec la GP :

– **Static Balanced Sampling (SBS) :** comme son nom l'indique, c'est un échantillonnage statique qui choisit les exemples d'apprentissage avec une proba-

Algorithme 7 Calcul du seuil TBS

```
Trier les poids des arêtes dans un ordre croissant
seuil ← W_e(i) %Utiliser le seuil trouvé lors de la dernière sélection et plus petit poids à la première sélection)
répéter
    Appliquer la sélection TBS avec le seuil courant
    si taille du sous-ensemble < taille cible alors
        seuil ← W_e(i + pas) %un seuil plus grand
    sinon si taille sous-ensemble > taille cible alors
        seuil ← W_e(i − pas) %un seuil plus petit
    fin si
jusqu'à taille cible atteinte ou nombre d'essais maximal atteint
```

bilité uniforme à partir de chaque classe sans remise jusqu'à obtenir le même nombre d'exemples des deux classes majoritaire et minoritaire ayant la taille souhaitée. L'échantillonnage est renouvelé à chaque génération.

- **Basic Under-sampling (BUSS) :** (sous-échantillonnage de base) il s'agit de prendre tous les exemples d'apprentissage de la classe minoritaire N_{min} dans l'échantillon et puis d'ajouter un nombre équivalent d'exemples d'apprentissage de la classe majoritaire aléatoirement.
- **Basic Over-sampling (BOSS) :** (sur-échantillonnage de base) de manière symétrique au sous-échantillonnage de base, il choisit tous les exemples d'apprentissage de classe majoritaire (N_{maj}). Puis, pour ajouter le même nombre de la classe minoritaire, la sélection aléatoire est effectuée avec remise. Ce procédé engendre la redondance pour les exemples de la classe minoritaire et logiquement l'augmentation de la taille de l'ensemble d'apprentissage.
- **Under-sampling A :** c'est une variante du sous-échantillonnage où plusieurs échantillons (voir le tableau 3.1) sont créés à chaque génération en utilisant le sous-échantillonnage de base. Le nombre d'échantillon $Num_{samples}$ est calculé par $\left\lfloor \frac{N_{maj}}{N_{min}} + \frac{1}{2} \right\rfloor$. Les individus sont évalués avec chacun de ces échantillons et leurs valeurs de fitness sont calculées avec la moyenne des fitness obtenues avec chaque échantillon.
- **Under-sampling B :** il reprend le même traitement effectué que l'algorithme précédent et utilise la fitness minimum au lieu de la moyenne.
- **Over-sampling A :** un nombre égal à $\max(2, \frac{1}{2} \left\lfloor \frac{N_{maj}}{N_{min}} + \frac{1}{2} \right\rfloor)$ échantillons sont créés avec sur-échantillonnage de base. La valeur finale de la fitness est la moyenne obtenue sur tous ces échantillons.
- **Over-sampling B :** a le même traitement que le précédent avec utilisation de la fitness minimum.

TABLE 3.1 – Nombre et tailles des échantillons pour les méthodes balancées.

Méthode	Nombre d'échantillons	Taille
Static Balanced Sampling	1	taille cible [2]
Basic Under-sampling	1	$2N_{min}$
Under-sampling A, Under-sampling B	$\left\lfloor \frac{N_{maj}}{N_{min}} + \frac{1}{2} \right\rfloor$	$2N_{min}$
Basic Over-sampling	1	$2N_{maj}$
Over-sampling A, Over-sampling B	$\max(2, \frac{1}{2} \left\lfloor \frac{N_{maj}}{N_{min}} + \frac{1}{2} \right\rfloor)$	$2 \left\lfloor \frac{N_{maj}}{Num_{samples}} \right\rfloor$

N_{min} : nombre des exemples d'apprentissage de la classe minoritaire.
N_{maj} : nombre des exemples d'apprentissage de la classe majoritaire.
$Num_{samples}$: nombre d'échantillons (colonne 2).
$\lfloor \rfloor$: partie entière.

Autres techniques d'échantillonnage : Ce paragraphe donne des exemples de méthodes d'échantillonnage qui ne sont pas classées dans les catégories précédentes et qui ont été utilisées avec GP.

Rational Allocation of Trials **RAT** [Teller & David 1997] est un algorithme qui alloue les exemples d'apprentissage uniquement pour les individus pour lesquels le coût de l'évaluation d'un exemple supplémentaire est négligeable par rapport à l'utilité attendue de la connaissance apportée par cet exemple d'apprentissage. Bien qu'il s'agisse d'un mécanisme d'accélération qui a fait ses preuves et qui mène à un nombre réduit et varié d'exemples d'apprentissage par individus, il ne peut être considéré comme un algorithme d'échantillonnage indépendant vu qu'il est étroitement lié au processus d'apprentissage et en particulier à la fitness du classifieur lui-même. Le principe de cette méthode est d'utiliser un ensemble minimal d'exemples d'apprentissage. Puis, pour les cas non sélectionnés, opérer au cas par cas selon la probabilité que pourrait avoir l'individu de « gagner des tournois [3] qu'il est en train de perdre ou de perdre des tournois qu'il est en train de gagner ».

Échantillonnage entrelacé Interleaved sampling **IS** [Gonçalves & Silva 2013] consiste à alterner d'une génération à une autre entre deux bases d'apprentissage. La première est la base d'apprentissage complète tandis que la deuxième ne contient qu'un seul exemple d'apprentissage. Selon le moment où il faut basculer de la première à la seconde base, plusieurs variantes de cette approche ont été introduites.

La première variante alterne les bases d'apprentissage à chaque génération.

La deuxième appelée *Interleaved Single Variant* utilise un paramètre qui détermine le nombre de générations pendant lesquelles un seul exemple d'apprentissage est utilisé après une génération qui utilise toute la base d'apprentissage. Par

2. Pour avoir une taille totale de N alors $\frac{N}{2}$ exemples d'apprentissage de chacune des classes majoritaire et minoritaire sont sélectionnés.
3. Gagner un tournoi : être choisi par la méthode de sélection par tournoi

exemple, si ce paramètre vaut 5, nous aurons l'alternance suivante : 1 génération avec toute la base d'apprentissage suivi de 5 générations utilisant un seul exemple d'apprentissage.

La troisième variante est définie de manière symétrique à la deuxième ; elle est appelée *Interleaved All Variant*.

La dernière variante est *Random Interleaved Sampling*. Elle choisit l'ensemble d'apprentissage parmi les 2 utilisés (toute la base ou un seul exemple d'apprentissage) avec une probabilité donnée comme paramètre.

Dans toute ces variantes, la sélection de l'exemple d'apprentissage unique est pris de manière aléatoire à chaque alternance.

Ce type d'échantillonnage a été appliqué pour remédier au problème de sur-apprentissage. Il a été expérimenté contre plusieurs problèmes de régression symbolique et de classification [Martínez *et al.* 2017]. La plus grande base de données utilisée compte 1151 exemples d'apprentissage ce qui reste une taille très inférieure à celle des bases auxquelles nous nous intéressons dans cette thèse.

3.4.2.2 Échantillonnage hiérarchique ou multi-niveaux

Ce type d'échantillonnage est basé sur des niveaux utilisant différentes méthodes d'échantillonnage. C'est un principe qui mime celui du concept de la hiérarchie de mémoire. La figure 3.5 illustre les principales étapes appliquées avec l'échantillonnage hiérarchique pour construire le sous-ensemble d'apprentissage à partir de la base de départ. Le schéma classique est composé de 3 niveaux. Le premier consiste à créer les blocs avec une taille donnée à partir de la base initiale. Ces blocs sont enregistrés sur le disque. Les deux niveaux suivants sont une exécution en pipeline de deux méthodes d'échantillonnage. Il est à noter que celle exécutée au niveau 2 doit être adaptée pour la sélection de blocs et non la sélection d'exemples d'apprentissage.

RSS-DSS et DSS-DSS : Robert Curry et Malcom Heywood ont conçu une extension pour l'algorithme DSS sous la forme d'une hiérarchie de 3 niveaux [Curry & Heywood 2004]. Au niveau 0, la base d'apprentissage est partitionnée en blocs suffisamment petit pour être logés en RAM. Puis, au niveau 1, les blocs sont choisis via la méthode RSS ou DSS. Enfin, au niveau 2, le bloc sélectionné est considéré comme une base d'apprentissage entière sur laquelle sera appliqué DSS à plusieurs reprises. Selon l'algorithme appliqué au niveau 1, nous obtenons une hiérarchie **RSS-DSS** ou **DSS-DSS**.

L'algorithme général est donné ci-après (Algorithme 8).

Hormis les paramètres de DSS (cf. Section 3.4.2.1), de nouveaux paramètres sont ajoutés par cette méthode à savoir : taille du bloc de niveau 0, nombre d'itération du niveau 1, nombre d'itérations du niveau 2, nombre maximum d'itérations du niveau 2 et nombre de tournois. Tous ces paramètres sont à ajuster et spécifier lors de l'initialisation de l'algorithme à l'exception du nombre d'itérations du niveau 2

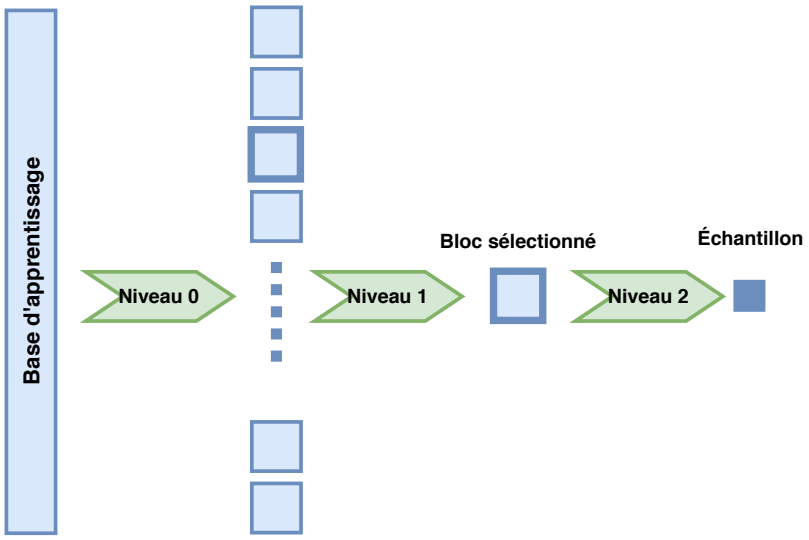


FIGURE 3.5 – Échantillonnage Hiérarchique.

calculé avec la formule :

$$I_b(i) = I_{max} * E_b(i-1). \tag{3.8}$$

$I_b(i)$: nombre d'itérations au niveau 2 sur le bloc b à l'itération i du niveau 1.
I_{max} : nombre maximum d'itérations au niveau 2.
$E_b(i-1)$: taux d'erreurs sur le bloc b par le meilleur individu à l'itération précédente.

Une version modifiée de DSS est utilisée au niveau 2 dans laquelle deux roulettes existent par bloc : une pour contrôler les exemples d'apprentissage relativement à leurs âges alors que dans le cas de la seconde la sélection est faite selon la difficulté. Le choix de la roulette à appliquer dépend d'une probabilité (paramètre supplémentaire).

Pour la hiérarchie DSS-DSS, le poids d'un bloc et sa probabilité de sélection sont définis par l'équation 3.9 afin que DSS puisse être utilisé au niveau 1.

$$\begin{aligned} Block(i)_{weight} &= \frac{\%diff * Block_{diff}(i)}{\sum_j Block_{diff}(j)} + \frac{\%age * Block_{age}(i)}{\sum_j Block_{age}(j)}. \\ P(Block(i)) &= \frac{Block(i)_{weight}}{\sum_j Block(j)_{weight}} \end{aligned} \tag{3.9}$$

Où %diff et %age sont les proportions utilisées par les 2 roulettes ; $Block_{diff}(i)$ et $Block_{age}(i)$ sont respectivement la difficulté et l'âge du bloc i [Curry & Heywood 2004].

Testées sur les bases KDD'99 et Adult dataset [Curry & Heywood 2004], les deux variantes hiérarchiques RSS-DSS et DSS-DSS ont réalisé des résultats compétitifs dans un temps nettement plus court qu'avec RSS ou DSS seuls. Dans ces expériences, DSS-DSS produit de meilleurs résultats que RSS-DSS.

Algorithme 8 Échantillonnage Hiérarchique [Curry & Heywood 2004]

```
1: Partitionner la base en blocs %niveau 0
2: répéter %Itérations de niveau 1
3:    Sélectionner un bloc %algorithme du niveau 1 : RSS ou DSS
4:    nombre d'itérations de niveau 2 ← nombre maximum d'itérations
5:    répéter %Itérations du niveau 2 <= nombre maximum d'itérations
6:       Sélectionner les exemples d'apprentissage %algorithme du niveau 2 : DSS
7:       répéter %Itérations de tournoi
8:          Effectuer sélection par tournoi sur la population
9:          Évaluer les individus sélectionnés sur le sous-ensemble créé en 6
10:         Actualiser l'âge et la difficulté des exemples d'apprentissage
11:         Appliquer les opérateurs génétiques
12:      jusqu'à Fin de tournoi
13:   jusqu'à Fin itération du niveau 2 %calculé en 14
14:   Actualiser le nombre d'itérations de niveau 2 à effectuer lors de la prochaine
      instance du bloc avec équation 3.8
15: jusqu'à Fin niveau 1
```

Balanced Block-DSS (BB-DSS) : Une extension au précédent travail a été proposée dans [Curry *et al.* 2007] avec *Balanced Block DSS* ou DSS avec Bloc équilibré. **BB-DSS** modifie principalement la sélection de bloc au niveau 0 avec l'objectif d'obtenir un bloc équilibré, par rapport à la fréquence des classes, au niveau 1. Le bloc équilibré ne reflète pas la même distribution de l'ensemble de départ mais respecte plutôt un ratio fixe pour chaque classe indépendamment de leurs fréquences initiales. Au niveau 0, toute la base d'apprentissage est triée (selon les classes) puis morcelée en des partitions séparées de chaque classe. Pour créer les blocs utilisés dans le niveau 1, une partition de chaque classe sera utilisée. La taille de la partition d'une classe dépend de la taille du bloc et du ratio de la classe. Pour un bloc de taille égale à 1000 et un problème de 2 classes ayant respectivement les ratios 25% et 75%, les tailles des partitions de classe seront 250 et 750. Pour appliquer DSS sur ces partitions, la difficulté et l'âge de partition sont définis pour calculer le poids de la partition et sa probabilité de sélection [Curry *et al.* 2007]. BB-DSS a été comparé à RSS-DSS et CasGP [4] avec les différentes bases d'apprentissage suivantes : Adult, Census, Shuttle et KDD'99 [5]. Les résultats de ces expériences ont montré que l'algorithme BB-DSS est assez proche de la précision de CasGP en préservant le gain en temps d'exécution de l'algorithme RSS-DSS.

4. Cascaded GP : utilise l'algorithme RSS-DSS pour construire une cascade de classifieurs GP composée d'un nombre bien déterminé de couches.

5. Voir UCI Machine Learning Repository `http://archive.ics.uci.edu/ml/`

3.4.3 Discussion

Nous présentons une synthèse des différentes techniques d'échantillonnage passées en revue dans le tableau 3.2. Les méthodes sont classées selon la taxonomie proposée dans la section 3.3. Cette classification montre que plusieurs configurations n'ont jamais été proposées ni expérimentées. À titre d'exemple, toutes les techniques 1-niveau ont été proposées uniquement avec la fréquence *par-generation*. De plus, la quantité d'échantillonnage *par-sous-population* n'a pas été testée avec les méthodes dynamiques ou actives. Les décisions conceptuelles *par-η-génération*, *par-population* et *par-sous-population* semblent être bien adaptées pour l'apprentissage de très larges bases de données à travers la possibilité de contrôler le nombre d'évaluations à effectuer par génération et le nombre d'échantillons à créer.

TABLE 3.2 – Classification des méthodes d'échantillonnage selon leurs catégories et décisions conceptuelles.
"r" : par-run, "g" : par-génération, "η-g" : par-η-génération
"i" : par-individu, "p" : par-population, "sp" : par-sous-population

Catégorie		Méthode	Fréquence	Quantité	Stratégie
Échantillonnage	**Statique**	Holdout (-)	r	p	aléatoire
		aléatoire (-)	r	p	aléatoire
		bagging [Breiman 1996] (-)	r	p	aléatoire avec remise
		boosting [Breiman 1996] (-)	r	p	pondéré
		BagGP [Iba 1999] (-)	r	sp	aléatoire avec remise
		BoostGP [Iba 1999] (-)	r	sp	pondéré
Échantillonnage actif	**1-niveau**	RSS [Gathercole & Ross 1994] (+)	g	p	aléatoire
		IRS [Zhang & Cho 1999] (-)	g	i	aléatoire
		SS [Nordin & Banzhaf 1997] (-)	g	i	aléatoire
		DSS [Gathercole & Ross 1994] (+)	g	p	pondéré
		IDI [Zhang & Joung 1999] (*)	g	i	incrémental
		TBS [Lasarczyk *et al.* 2004] (+)	g	p	par topologie
		BUSS/BOSS [Hunt *et al.* 2010] (+)	g	p	balancé
		IS [Gonçalves & Silva 2013] (-)	g	p	entrelacé
	Multi-niveaux	RSS-DSS [Curry & Heywood 2004] (+)	η-g/g	p	hybride (aléatoire et pondéré)
		DSS-DSS [Curry & Heywood 2004] (+)	η-g/g	p	pondéré
		BB-DSS [Curry *et al.* 2007](+)	η-g/g	p	hybride (balancé et pondéré)

Toutes les techniques du tableau 3.2 ont fait l'objet d'études expérimentales dans la littérature. Elles ont donné des résultats satisfaisants par rapports aux objectifs définis dans les études respectives : améliorer le taux de succès, perfectionner la généralisation d'un modèle, prévoir le sur-apprentissage, etc. Cependant, il n'y a pas d'études pour comparer ces techniques autour du même objectif et dans un même environnement expérimental.

Quelques travaux (comme [Curry & Heywood 2004, Lasarczyk *et al.* 2004, Hunt *et al.* 2010]) ont proposé une comparaison d'un nombre réduit de méthodes. Toutefois, chaque étude utilise une variante de GP différente avec des paramètres différents et des problèmes différents. En outre, un nombre très limité de méthodes a été testé avec des base de données de larges dimensions.

Pour ces raisons, nous proposons dans la suite de ce chapitre, une étude comparative d'une sélection de méthodes avec GP comme algorithme d'apprentissage pour un problème de classification à large base d'apprentissage. La sélection est basée sur l'aptitude a priori de ces méthodes à gérer des bases volumineuses selon des critères détaillés dans ce qui suit.

Les techniques d'échantillonnage utilisées avec GP en présence de données massives ont deux objectifs : accélérer GP en réduisant les exemples d'apprentissage à évaluer pour le calcul de la fitness des individus et garantir un minimum de généralisation des solutions obtenues. Afin d'atteindre ces objectifs, au moins trois sous-objectifs sont à considérer :

- permettre à la population de voir le plus possible d'exemples d'apprentissage durant l'évolution pour éviter qu'un type particulier de connaissances ne soit ignoré à cause de l'échantillonnage,
- réduire le temps d'apprentissage,
- donner suffisamment de temps à GP pour apprendre à partir des données disponibles.

Étant donné ces sous-objectifs, nous avons évalué l'aptitude et la pertinence de chaque méthode d'échantillonnage. Nous avons aussi considéré les trois décisions conceptuelle ainsi que leur complexité d'implémentation. Cette évaluation est traduite par l'ajout des symboles moins (-) ou plus (+) après le nom de la méthode dans le tableau 3.2. Les techniques marquées avec (-) sont jugées non adaptées (sans modification) pour l'apprentissage à grande échelle. Par exemple, chaque technique basée sur la fréquence *par-run* (toutes les méthodes statiques) ou la quantité *par-individu* (tel que IRS et SS) ne sont pas implémentées dans cette étude. En effet, une stratégie *par-run* utilise un seul échantillon pendant le run. Avec une base d'apprentissage volumineuse, le premier sous-objectif devient alors irréalisable. Quant à la stratégie *p*ar-individu, le nombre d'opérations d'échantillonnage à faire à chaque génération est susceptible de générer un temps de calcul non négligeable ce qui est en contradiction avec le deuxième sous-objectif. Une exception est faite pour IDI, marquée avec (*), qui sera implémentée avec des simplifications détaillées dans la section suivante. En effet, IDI peut contrôler la taille de l'échantillon à travers la taille de l'échantillon initial et du nombre d'exemples d'apprentissage à ajouter à chaque génération.

3.5 Étude comparative

L'objectif de cette section est d'effectuer une étude expérimentale d'un nombre représentatif de méthodes d'échantillonnage à partir de celles décrites dans ce chapitre. Nous avons mené cette étude dans un cadre expérimental homogène pour toutes les méthodes et avec un problème de classification ayant une large base d'apprentissage.

Nous ajoutons la méthode Full Subset Selection (FSS) qui utilise la totalité de la base d'apprentissage en vue de montrer les améliorations introduite par l'échantillonnage dynamique. Par ailleurs, cinq techniques actives de la catégorie 1-niveau sont implémentées en respectant l'algorithme proposé par leurs auteurs respectifs : RSS, DSS, IDI, TBS et BUSS. Nous implémentons aussi une variante de RSS (BRSS), pour générer des échantillons aléatoires mais respectant les proportions originales de chaque classe dans la base d'apprentissage, et une variante de BUSS (BUSS2) qui est une interprétation différente basée sur la considération de 2 classes au lieu de 5 (voir 3.5.4). RSS-DSS est la technique hiérarchique qui a été implémentée.

De plus, ces expérimentations utilisent le même algorithme d'apprentissage basé sur la CGP avec des paramètres de valeurs comparables pour les méthodes implémentées. Dans ce qui suit, nous donnons les détails à propos de la librairie GP adoptée, le choix des paramètres GP, une description de la base d'apprentissage et quelques précisions concernant les implémentations des différents algorithmes. Les résultats obtenus sont ensuite exposés et discutés.

3.5.1 Librairie évolutionnaire ECJ

Parmi plusieurs librairies offrant une implémentation des algorithmes évolutionnaires, nous avons choisi la librairie ECJ de Sean Luke [Luke 2017] pour implémenter et tester les algorithmes d'échantillonnage. En effet, ECJ fournit une implémentation des différents algorithmes évolutionnaire comme GP, Stratégie d'évolution, algorithmes génétiques ... De plus, c'est une librairie à code source ouvert écrite en Java, elle offre la gamme la plus complète de représentations pour GP (Koza, linéaire, grammaticale, ...). Cette librairie constitue une API très flexible fortement paramétrable à travers des fichiers de paramètres bien décrits dans le manuel en ligne : « ECJ owner's manual ».

ECJ est un projet actif à l'université George Mason dans le laboratoire de calculs évolutionnaires « Evolutionary Computation Laboratory » offrant des mises à jour fréquentes. Elle bénéficie de plusieurs contributions comme celle utilisée dans nos expériences qui est une implémentation de la GP cartésienne (cf. 1.5.1.2) développée par David Oranchak [CGP 2009].

3.5.2 Base d'apprentissage

Nous avons choisi une large base d'apprentissage, qui a été au départ créée pour la compétition organisée à la conférence « the Fifth International Conference

on Knowledge Discovery and Data Mining KDD-99 » et concerne le problème de détection d'intrusions [UCI 1999a]. La base originale contient 5 millions d'exemples d'apprentissage contenant les informations à propos des connexions d'un trafic simulé. Chaque exemple est décrit avec 41 attributs et possède un attribut indiquant sa classe. Les exemples sont répartis en 5 classes : une classe pour les connexions normales (Normal) et 4 classes qui regroupent les attaques selon leurs types (Dos : Denial of service, Probe, R2L : Remote to User et U2R : User to Root). Un autre ensemble (corrected set), est représenté de manière identique et utilisé comme base de test. Le processus d'apprentissage est effectué sur la base dérivée de la base originale et appelée 10% KDD-99 pour apprendre à classifier les connexions en normales ou attaques (indépendamment de la classe d'attaque). Le meilleur individu après chaque run est évalué sur la base de test. Ces deux ensembles sont décrits dans le tableau 3.3.

TABLE 3.3 – Bases d'apprentissage du KDD-99.

Classe	**Nombre d'exemples d'apprentissage**	
	10% Ensemble d'apprentissage	*Ensemble de Test*
Normal	97278	60593
Dos	391458	229853
probe	4107	4166
R2L	1126	16347
U2R	52	70
Nombre d'attaques total	396743	250436
Nombre d'exemples total	494021	311029

Chaque exemple d'apprentissage est donc décrit par 41 attributs (cf. tableau A.1 dans l'annexe A). 9 attributs sont discrets et le reste des attributs est continu. Dans ce travail, nous nous focalisons sur les techniques d'apprentissage et nous ne traitons pas le pré-traitement de données comme l'ingénierie des attributs. C'est une étape cruciale dans l'apprentissage automatique qui affecte la qualité des résultats mais nous avons opté pour ce choix afin d'isoler l'effet de l'échantillonnage de tout autre technique d'amélioration des performances. Pour cela, seulement quelques pré-traitements nécessaires des données ont été réalisés :

- Transformation des attributs à valeurs nominales avec la méthode « Label Encoder »,
- Mise à l'échelle ou normalisation avec MinMax :

$$X_n = \frac{X - X_{min}}{X_{max} - X_{min}},$$

- Binarisation des classes d'attaques : le problème est converti en un problème de classification binaire avec la classe « Normal » et la classe « Attaque». Les attaques (Dos, Probe, R2L and U2R) sont fusionnées en une seule classe.

3.5.3 Configuration GP

Les expériences sont réalisées en se basant sur la GP cartésienne (CGP) qui nécessite de nombreux paramètres pour contrôler le processus évolutionnaire de GP. Les deux paragraphes suivants récapitulent la configuration utilisée pour ces paramètres.

Ensembles de terminaux et fonctions

L'ensemble des terminaux inclut les variables d'entrée groupant les 41 attributs de KDD-99 (voir annexe A) avec 8 constantes générées aléatoirement.

Tandis que l'ensemble des fonctions contient les opérateurs arithmétiques de base, de comparaison et logiques totalisant 17 fonctions.

TABLE 3.4 – Ensembles de terminaux et fonctions.

Ensemble de fonctions (nœuds)	
Opérateurs arithmétiques :	$+, -, *, \%$
Opérateurs de comparaison :	$<, >, <=, >=, =$
Opérateurs logiques :	AND, OR, NOT, NOR, NAND
Autre :	NEGATE, IF (IF THEN ELSE), IFLEZE(IF $<=0$ THEN ELSE)
Ensemble de terminaux	
Attributs KDD-99	41
Constantes	8 dans $[-2, 2[$

Paramètres CGP

L'implémentation de CGP requiert différents paramètres déterminant son efficacité. La pratique usuelle pour l'ajustement de GP est de mener des séries d'essais pour faire des choix de paramètres pour l'ultime exécution de GP. Cette procédure ne rentre pas dans les objectifs de cette thèse et n'est pas entièrement examinée. Les paramètres ainsi que leurs valeurs retenues (après quelques expérimentations) sont résumés dans le tableau 3.5.

3.5.4 Implémentation des algorithmes d'échantillonnage

Les valeurs des paramètres supplémentaires introduits par chacune des méthodes sont fixées selon les recommandations de leurs auteurs respectifs. Certaines méthodes ont été simplifiées ou modifiées afin de les garder dans le même contexte de comparaison avec les autres méthodes. Le tableau 3.6 montre la configuration de chaque méthode.

BRSS C'est une légère modification de RSS pour obtenir des échantillons équilibrés (Balanced RSS), dans laquelle l'échantillon généré par RSS reflète les fré-

TABLE 3.5 – Paramètres CGP.

Paramètre	Valeur
Taille de la population	512 pour RSS, DSS, BUSS BRSS et RSS-DSS 128 pour IDI, TBS et GP standard
Nombre de sous-populations	1
Nombre de générations	500 pour RSS, DSS, BUSS et BRSS 100 pour IDI, TBS selon itérations RSS pour RSS-DSS
Nœuds CGP	300
Entrées	49
Sorties	1 (2 classes)
Taille Tournoi	4
Probabilité de croisement	0.9
Probabilité de mutation	0.04
Fitness	Minimiser les erreurs de classification

quences des classes dans la base d'apprentissage initiale. Le nombre d'exemples d'apprentissage de chaque classe est calculé selon la taille cible.

RSS-DSS La version implémentée de RSS-DSS n'utilise pas la même méthode pour le calcul de la fitness que celle proposée par les auteurs dans [Curry *et al.* 2007]. Nous avons utilisé la même fonction de fitness que pour les autres méthodes d'échantillonnage testées. L'effet de cette modification est discuté dans la section 3.6. Deux configurations ont été testées comme le montre le tableau 3.6 (notées RSS-DSS et RSS-DSS2) dans lesquelles la taille cible, les itérations RSS et DSS sont changées simultanément.

IDI Vu que nous utilisons un incrément et un nombre de générations fixes, l'algorithme aboutit à un ensemble plus petit que tout l'ensemble d'apprentissage lors de la dernière génération.

TBS C'est la première fois que cet algorithme est utilisé avec une base de cette taille. Ceci nous a conduit à modifier sa méthode de calcul du seuil comme décrit dans Algorithme 9. Cette modification consiste à utiliser initialement le plus petit poids d'une arête (supérieur à 0) comme seuil de départ. Ceci nous permet d'obtenir un sous-ensemble ayant la taille la plus petite selon le principe de TBS. Ensuite, tant que la taille du sous-ensemble est inférieure à la taille cible, nous refaisons la sélection en se basant sur le prochain seuil de la liste triée dans l'ordre croissant.

TABLE 3.6 – Paramètres spécifiques des algorithmes d'échantillonnage.

Algorithme	**Paramètre**	**Valeur**
FSS	-	-
RSS	Taille cible	5000
BRSS	Taille cible	5000
	Méthode d'équilibrage	selon les proportions des classes dans la base d'origine
DSS	Taille cible	5000
	Exposant de difficulté	1
	Exposant de l'âge	3.5
RSS-DSS/ RSS-DSS2	Taille cible (taille du bloc de niveau 2)	2500 puis 100
	Taille du bloc de niveau 0	5000
	Itérations RSS	200 puis 500
	Max itérations DSS	20 puis 50
	Individus évalués par itération DSS	100 (20% de la taille de la population)
	Exposant de difficulté	1
	Exposant de l'âge	3.5
	Roulette de difficulté	70%
	Roulette d'âge	30%
TBS	Taille cible	1000
	Taux de perte	0.7
	Itérations	voir le paragraphe ci-après
IDI	Taille initiale	1000
	Incrément	10
	Facteur de diversité	0.3
BUSS/BUSS2	-	-

Algorithme 9 Calcul du seuil modifié

seuil ← le plus petit poids d'une arête *%pour avoir le plus petit sous-ensemble possible*
répéter
  Appliquer la sélection TBS avec le seuil courant sur les exemples candidats
  Supprimer les exemples sélectionnés de l'ensemble des candidats
  si taille cible non atteinte **alors**
    seuil ← le poids suivant
    Ajouter les exemples précédemment exclus aux candidats
  fin si
jusqu'à taille cible atteinte

BUSS Le base du KDD-99 est répartie en 5 classes : une classe « normale » et 4 classes d'attaque. La méthode d'échantillonnage utilise « U2R » (52 exemples) comme classe minoritaire puis sélectionne le même nombre d'exemples à partir des 3 classes d'attaques restantes pour obtenir un total de 208 exemples d'attaques. Dans une première expérimentation (notée BUSS), le même nombre d'exemples de la classe minoritaire est pris des exemples normaux et l'échantillon final a une taille de 260. Dans la deuxième expérimentation (notée BUSS2), le même nombre de toutes les classes d'attaques (208) est tiré de la classe normale ce qui donne un échantillon d'une taille plus grande que le précédent et qui a une taille de 416 (toutes les attaques sont ici considérées comme une même classe).

3.6 Résultats

Les différentes expériences ont été réalisées sur une workstation équipée d'un Intel i7-4810MQ (2.8GHZ) dotée de 8 GO de RAM tournant sous Windows 8.1 64-bit. CGP a été utilisée pour apprendre à distinguer les intrusions des connexions normales (problème binaire).

Mesures de performance

À la fin de chaque run, le meilleur individu selon la fonction de fitness est testé sur la totalité de l'ensemble d'apprentissage et puis sur l'ensemble de test. Les résultats sont enregistrés sous la forme d'une matrice de confusion à partir de laquelle le taux de succès (accuracy), le taux de vrais positifs (True Positive Rate : TPR) et le taux de faux positifs (False Positive Rate : FPR) sont calculés.

$$accuracy = \frac{Vrais\ Positifs + Vrais\ Négatifs}{Taille\ totale}. \tag{3.10}$$

$$TPR = \frac{Vrais\ Positifs}{Vrais\ Positifs + Faux\ Négatifs}. \tag{3.11}$$

$$FPR = \frac{Faux\ Positifs}{Faux\ Positifs + Vrais\ Négatifs}. \tag{3.12}$$

Le temps d'apprentissage est la période qui s'étale entre la première et la dernière génération en secondes (Voir tableau 3.5 pour le nombre de générations).

Dans le but de comparer les performances des méthodes implémentées, six mesures sont sauvegardées tout au long des 21 runs effectués pour chaque technique d'échantillonnage : accuracy, TPR et FPR pour le meilleur individu sur les 21 runs et les mesures moyennes sur ces runs. Les meilleures mesures sont reportées dans la figure 3.7 tandis que les mesures moyennes dans la figure 3.6. Par ailleurs, pour comparer ces méthodes par rapport à leurs coûts de calcul, le temps consommé pour achever chaque run et celui pour trouver le meilleur individu sont aussi enregistrés. Le tableau 3.7 montre leurs valeurs moyennes pour toutes les méthodes.

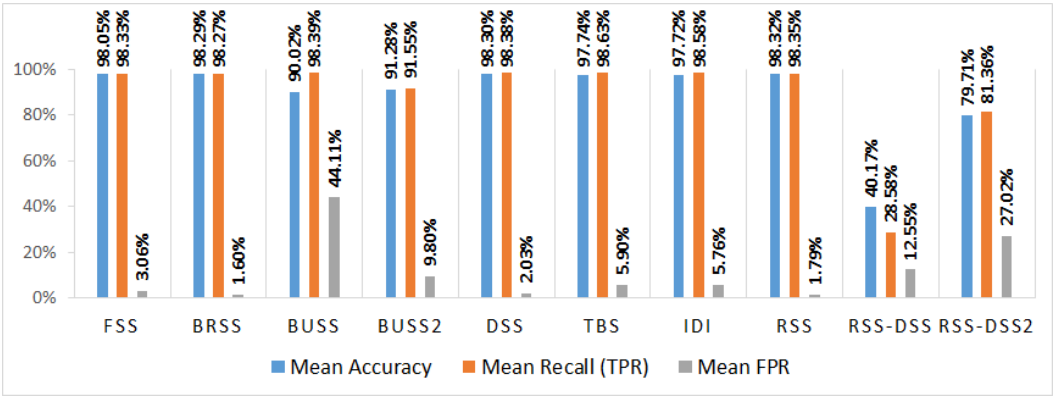


FIGURE 3.6 – Mesures moyennes sur l'ensemble d'apprentissage.

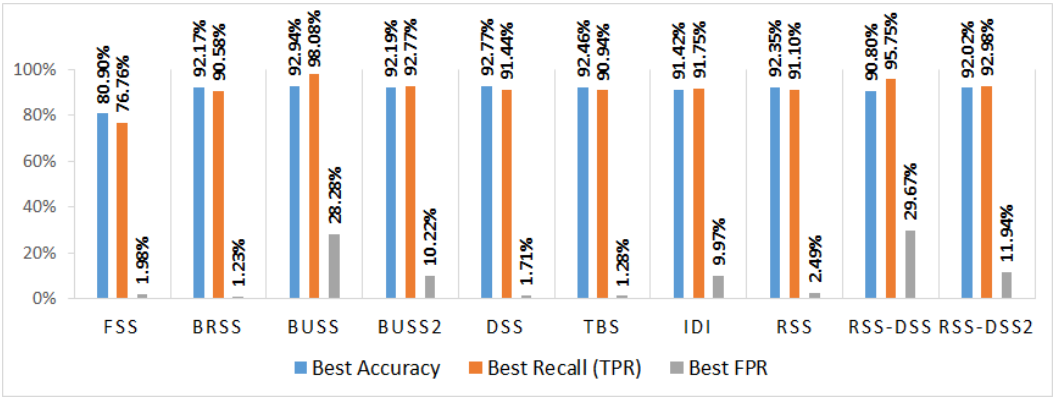


FIGURE 3.7 – Mesures du meilleur individu sur la base de test.

Les résultats obtenus montrent que le fait d'introduire une méthode d'échantillonnage permet d'améliorer les performances lors de l'apprentissage à partir d'une large base d'apprentissage. Toutes les méthodes implémentées ont de meilleurs résultats par rapport à l'utilisation de la totalité de l'ensemble d'apprentissage (FSS), et ce en se basant sur les deux mesures de performance à savoir accuracy et TPR (Figure 3.7).

À partir de la figure 3.6, nous pouvons dénoter que hormis la version de base de l'algorithme RSS-DSS, toutes les méthodes d'échantillonnage étudiées ont donné des résultats satisfaisants sur l'ensemble d'apprentissage. Comme le montre la figure 3.7, accuracy et le taux de vrais positifs des meilleures solutions obtenues par toutes les méthodes sont assez comparables. Néanmoins, cette observation n'est pas valable pour les mesures du taux de faux positifs. En effet, selon accuracy et TPR des meilleurs classifieurs, la méthode BUSS se trouve en tête. Cependant, elle a la $2^{ème}$ plus mauvaise valeur de FPR (après RSS-DSS).

Un temps CPU supplémentaire considérable pour TBS et IDI est facilement détecté depuis le tableau 3.7 malgré l'usage de population de taille plus réduite et d'un nombre de générations inférieur aux autres méthodes. Cela est dû essentiellement à la mise à jour du graphe dans TBS et au croisement des sous-ensembles d'apprentissage pour IDI.

BUSS qui utilise un algorithme simple réalise la meilleure performance quant au temps d'apprentissage. Toutefois, ces résultats ne peuvent pas être généralisés et sont tributaires de la distribution des classes au sein de l'ensemble d'apprentissage original.

TABLE 3.7 – Mesures du temps (en secondes).

Méthode	Temps d'apprentissage moyen	Temps moyen de découverte de la meilleure solution
FSS	7362.272	2854.074
BRSS	1590.819	580.311
BUSS	75.612	20.135
BUSS2	102.428	43.054
DSS	1996.892	1358.257
TBS	11174.507	1494.977
IDI	11162.065	2002.886
RSS	1493.788	530.038
RSS-DSS	1000.366	43.999
RSS-DSS2	364.369	0.124

3.7 Discussion

L'implémentation intégrale de ces techniques nous a permis d'avoir une vision profonde des algorithmes qui y sont inhérents et leur complexité. Cette complexité nécessite un effort considérable selon une perspective de codage, cela est aussi lié à la librairie et l'environnement utilisés. Plusieurs facteurs contribuent à cette complexité dont les plus importants sont :

– le nombre d'étapes et phases incluses dans la méthode d'échantillonnage,

- le nombre de paramètres spécifiques introduits par l'algorithme pour être ajoutés aux paramètres de base de la programmation génétique,
- le degré d'incrustation de l'échantillonnage dans la boucle évolutionnaire et comment les différentes phases de l'algorithme sont invoquées. Par exemple, dans IDI, la phase de croisement des sous-ensembles d'apprentissage doit être réalisée au même moment que l'opération de croisement évolutionnaire,
- les données et les structures de données utilisées pour sauvegarder les informations requises par la méthode d'échantillonnage.

L'échantillonnage aléatoire et celui balancé (RSS, BUSS et BRSS) sont les approches les plus faciles à implémenter. En effet, ces méthodes n'ont pas de paramètres spécifiques et leurs étapes sont exécutés au début de la boucle évolutionnaire en dehors de toute opération évolutionnaire.

Parmi les méthodes implémentées, DSS peut être considérée à mi-chemin entre les méthodes simples et complexes. Elle nécessite seulement deux paramètres spécifiques et génère un échantillon par génération. Comme piste à explorer dans le but d'améliorer cette méthode, nous pouvons citer l'étude de l'effet d'ajouter un paramètre permettant de réguler la fréquence de renouvellement des échantillons (qui est maintenant fixe et égal à 1). Ceci permettrait de garder le même sous-ensemble d'apprentissage pendant un nombre bien déterminé de générations et par conséquent accorderait à la population plus de temps pour apprendre de ces exemples d'apprentissage.

Le reste des méthodes est plus difficile à implémenter. Elles utilisent un nombre plus grand de paramètres spécifiques (cf. TBS, IDI et RSS-DSS dans le tableau 3.6) dont les valeurs requièrent une procédure d'ajustement pour être calculées. Cette procédure ne fait pas partie des axes de recherche de cette thèse.

TBS comprend 2 étapes coûteuses en temps CPU : la mise à jour TBS (Algorithme 5) et le calcul du seuil (Algorithme 7) ce qui lui interdit de traiter de larges bases d'apprentissage comme le montre la section 3.6. Nous avons traité ce problème avec Hierarchical Data Topology Based Selection dans [Hmida *et al.* 2016a]. Ce travail fait l'objet du chapitre 4.

Dans IDI, le besoin d'enregistrer pour chaque individu son sous-ensemble d'apprentissage surcharge le processus d'apprentissage avec un volume de données important. En outre, l'opération de croisement de données (voir Algorithme 4) est injectée dans le croisement évolutionnaire. De ce fait, le temps de calcul additionnel causé par cette technique d'échantillonnage est multiplié par la taille de la population dans la mesure où un échantillon différent est généré pour chaque individu. Par la suite, une amélioration intuitive consiste à limiter le nombre d'opérations de croisement de données exécutées par génération. Dans ce cas, uniquement un sous-ensemble des enfants auront des sous-ensembles obtenus par croisement de données, les autres garderont le sous-ensemble d'un seul parent.

Enfin, l'échantillonnage hiérarchique à deux niveaux possède le plus grand nombre de paramètres qui proviennent des méthodes utilisées à chaque niveau. Selon les résultats expérimentaux dans Section 3.6, RSS-DSS réussit à traiter une

large base d'apprentissage cependant ses indicateurs de performance doivent être améliorés avec des techniques ad hoc comme la fonction de fitness basée sur le cumul de bonne prédictions durant les itérations par tournoi qui a été utilisée dans [Curry & Heywood 2004].

3.8 Conclusion

Le principal objectif de cette étude est de dresser une revue des méthodes d'échantillonnage actives utilisées avec la Programmation Génétique et d'étudier la capacité d'un ensemble de ces méthodes à traiter les grandes bases d'apprentissage. Ces techniques ont été implémentées avec le même environnement puis appliquées à un problème de classification qui est celui de détection d'intrusions avec la base KDD'99.

Trois conclusions peuvent être déduites de cette étude expérimentale et ses résultats. D'abord, toute technique active d'échantillonnage est capable de produire des classifieurs avec une meilleure généralisation par comparaison à la GP standard qui utilise toute la base d'apprentissage. Néanmoins, l'applicabilité de certaines méthodes est limitée à cause du leurs temps de calcul assez élevés (comme IDI et TBS) ou parce qu'elles requièrent des conditions spécifiques pour être efficace (tel que RSS-DSS qui utilise une fonction de fitness spécifique). C'est la seconde conclusion. Tertio, certaines techniques simples comme BUSS ou encore RSS ont un temps de calcul très inférieur et peuvent atteindre des niveaux de performances compétitifs par rapport aux techniques élaborées.

Afin de résoudre le problème de temps d'exécution des méthodes IDI et TBS, les combiner avec d'autres techniques d'échantillonnage de façon hiérarchique est une voie prometteuse, nous l'explorons dans le chapitre 4. La fonction de fitness, comme tous les paramètres de GP, affecte la qualité des résultats et ceci peut être étudié sur les méthodes implémentées dans ce chapitre. Enfin, une importante alternative à examiner est celle de l'injection des méthodes actives d'échantillonnage dans un environnement de stockage et traitement distribués et plus particulièrement Hadoop [6] qui a un écosystème assez riche pour manier des problèmes Big Data. Cette alternative est étudiée dans le chapitre 5.

6. http://hadoop.apache.org

CHAPITRE 4

Extension des méthodes d'échantillonnage pour les données massives avec CGP

Sommaire

4.1 Introduction 98
4.2 Échantillonnage hiérarchique avec TBS 98
4.2.1 Contexte 98
4.2.2 RSS-TBS 99
4.2.3 BUSS-RSS-TBS 99
4.2.4 Étude expérimentale 100
4.2.5 Discussion 103
4.3 Échantillonnage hiérarchique pour classifier les bosons de Higgs . 104
4.3.1 Contexte 104
4.3.2 La base Higgs et travaux antérieurs 105
4.3.3 Extension de CGP avec l'échantillonnage actif 106
4.3.4 Paramètres expérimentaux 108
4.3.5 Résultats et discussion 109
4.4 Échantillonnage adaptatif 110
4.4.1 Contexte et motivations 110
4.4.2 Contrôler la fréquence d'échantillonnage 111
4.4.3 Étude expérimentale 115
4.4.4 Résultats 116
4.4.5 Discussion 120
4.5 Conclusion 121

4.1 Introduction

Dans ce chapitre nous allons présenter les résultats de nos travaux concernant, en premier lieu, l'adaptation de TBS avec l'échantillonnage hiérarchique appliqué à la base KDD et publié dans [Hmida *et al.* 2016a]. Puis, nous exposons les résultats de l'expérimentation de RSS-DSS sur le problème de classification des bosons de Higgs ayant pour objectif de montrer que ce type d'échantillonnage permet de traiter des problèmes Big Data. Enfin, en partant de la discussion des résultats du chapitre 3, nous introduisons un nouveau type d'échantillonnage que nous avons appelée adaptatif. Cet échantillonnage est implémenté et testé à travers l'ajout d'un paramètre ajustant la fréquence de renouvellement de l'échantillon pour DSS.

4.2 Échantillonnage hiérarchique avec TBS

4.2.1 Contexte

L'étude expérimentale dans le chapitre 3, a montré que l'échantillonnage est un moyen efficace pour contourner le coût rebutant de l'apprentissage par GP. Toutefois, selon les 3 décisions conceptuelles (fréquence, stratégie et quantité), un algorithme peut être plus au moins adapté à ce type de problèmes. Cette étude a montré aussi qu'une méthode d'échantillonnage, considérée comme couteuse quand elle est appliquée seule à un grand volume de données, peut devenir performante dans le même contexte si elle est intégrée dans un processus d'échantillonnage hiérarchique (cas de DSS). Ce type d'échantillonnage peut être une solution efficace pour la prise en compte des grands volumes de données. Nous souhaitons généraliser ce concept avec d'autres méthodes comme celle basée sur la topologie des données : TBS.

TBS [Lasarczyk *et al.* 2004] fait partie des algorithmes d'échantillonnage actif. Sa stratégie d'échantillonnage consiste à sélectionner les exemples d'apprentissage qui sont différents par rapport à un seuil défini sur un graphe de similarité construit durant le processus d'apprentissage. TBS a été appliqué pour résoudre des problèmes d'apprentissage automatique de petite taille [Lang & Witbrock 1988, Lasarczyk *et al.* 2004]. Son apport principal est le contrôle de la diversité des données d'apprentissage, ce qui améliore la qualité des classifieurs. Cependant, il génère un coût supplémentaire causé par la mise à jour des différents degrés de similarité entre les exemples d'apprentissage suite à l'évaluation des individus à chaque génération. Ce coût devient très important lorsque la base d'apprentissage est de grande taille.

Notre objectif consiste à allier TBS avec une autre méthode d'échantillonnage de façon hiérarchique dans le but de réduire le coût de mise à jour du graphe de similitude et par la suite permettre son utilisation avec les bases de données massives.

À ce jour, les applications de TBS associé à GP pour l'apprentissage automatique se limitent aux travaux de Lasarczyk et al. [Lasarczyk *et al.* 2004] où les bases d'apprentissage ne dépassent pas quelques milliers d'enregistrements. L'utilisation d'un graphe non orienté complet avec des bases de grande taille est à l'origine de son coût excessif. L'idée de base de notre proposition s'inspire de l'échantillonnage multi-niveaux pour combiner TBS avec d'autres méthodes qui ont pour but de fournir à TBS des sous-ensembles de taille réduite afin d'avoir un coût de mise à jour raisonnable. Les méthodes d'échantillonnage considérées sont RSS qui fait partie des méthodes aléatoires et BUSS appartenant aux méthodes balancées. TBS est alors appliqué sur différents échantillons sélectionnés de manière aléatoire à partir d'un nombre bien déterminé de blocs. Les blocs sont créés proportionnellement à la distribution des classes de la base d'apprentissage (méthode **RSS-TBS**) ou de manière balancée selon BUSS (méthode **BUSS-RSS-TBS**). Ces méthodes sont décrites dans les paragraphes suivants.

4.2.2 RSS-TBS

Au niveau 0, la base d'apprentissage initiale est partitionnée en blocs de taille fixe qui sont enregistrés sur des fichiers séparés sur le disque. Ces blocs sont construits de façon à préserver la distribution des classes dans la base d'apprentissage (Algorithme 10).

Algorithme 10 Construction des blocs de niveau 0

Paramètres : taille des blocs de niveau 0

1: Calculer les proportions des classes
2: Calculer le nombre d'exemples de chaque classe par bloc (selon sa taille)
3: **tant que** la base est non vide **faire**
4: Sélection aléatoire des exemples d'apprentissage de chaque classe selon 2
5: Écriture du bloc sur disque
6: **fin tant que**

Ensuite, au niveau 1, RSS est appliqué pour la sélection d'un bloc aléatoire. Au dernier niveau, le bloc précédemment sélectionné est échantillonné selon l'algorithme TBS (voir figure 4.1).

L'algorithme général est présenté dans algorithme 11.

À l'étape 13, le nombre des itérations TBS est calculé selon l'équation 3.8 dans laquelle le taux d'erreur est celui du meilleur individu actuel du run en cours.

Le calcul du seuil pour TBS suit le même algorithme simplifié utilisé dans l'étude comparative (algorithme 9).

4.2.3 BUSS-RSS-TBS

Cette méthode introduit, par rapport à RSS-TBS, l'application d'un autre algorithme lors de la construction des blocs de niveau 0. En effet, c'est l'algorithme

Algorithme 11 RSS-TBS

Paramètres : taille des blocs de niveau 0, taille des blocs de niveau 1

```
1: Partitionner la base en blocs %niveau 0
2: répéter %Itération du niveau 1
3:    Sélection aléatoire de bloc avec RSS
4:    répéter %Itérations de niveau 2 <= nombre maximum d'itérations
5:       Sélection du sous-ensemble %algorithme de niveau 2 : TBS
6:       répéter %Itérations de tournoi
7:          Effectuer la sélection par tournoi
8:          Entraîner les individus sélectionnés sur le sous-ensemble
9:       jusqu'à Fin de tournoi
10:      Mise à jour du graphe de topologie
11:      Appliquer les opérateurs génétiques
12:   jusqu'à itérations du niveau 2 = calculé en 13
13:   Actualiser le nombre d'itérations de niveau 2 à effectuer lors de la prochaine
      instance du bloc
14: jusqu'à Fin niveau 1
```

BUSS qui est appliqué dans ce niveau. Une première conséquence directe est que la taille du bloc n'est plus un paramètre mais calculé selon BUSS. Nous pensons que l'application de BUSS au niveau 0 permettrait, non seulement de gérer le volume des données, mais aussi une meilleure prise en compte de la variété des données, deux difficultés bien présentes avec les données massives. Le déséquilibre entre les classes aurait ainsi moins d'effets sur la qualité des classifieurs obtenus.

Avec la base KDD-99 utilisée dans l'étude expérimentale, les blocs peuvent avoir une taille de 260 ou 416 selon le principe utilisé dans 3.5.4. Nous notons ces 2 variantes respectivement BUSS-RSS-TBS et BUSS-RSS-TBS2.

La figure 4.1 synthétise les différentes variantes introduites dans cette section.

4.2.4 Étude expérimentale

Pour les méthodes RSS et TBS nous reprenons les mêmes paramètres et résultats du chapitre 3 afin de les comparer aux nouvelles méthodes (Tableau 3.4).

4.2.4.1 Base d'apprentissage

Nous utilisons la même base que celle du chapitre 3 dont la composition est donnée dans le tableau 3.3. Les attributs sont décrits dans l'annexe A. Ceci nous permet de comparer les résultats de TBS avec ceux de RSS-TBS et BUSS-RSS-TBS.

4.2.4.2 Paramètres CGP

Pour rester toujours dans le même environnement, CGP est utilisé pour l'apprentissage en gardant la même configuration de l'étude comparative. Le ta-

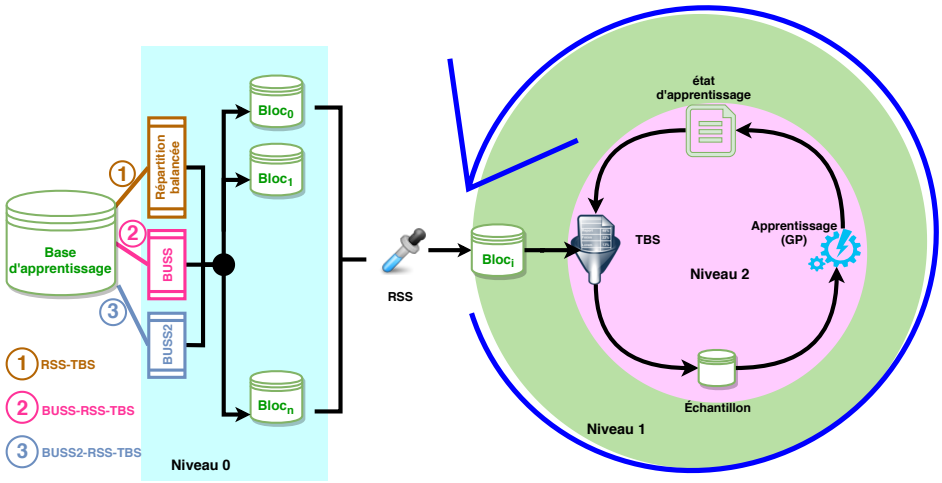


FIGURE 4.1 – Échantillonnage hiérarchique avec TBS.

bleau 3.4 contient l'ensemble des terminaux et fonctions utilisés. Tandis que les paramètres de CGP sont actualisées dans le tableau 4.1.

TABLE 4.1 – Paramètres CGP.

Paramètre	**Valeur**
Taille de la population	512 pour RSS, RSS-TBS et BUSS-RSS-TBS 128 pour TBS
Nombre de sous-populations	1
Nombre de générations	500 pour RSS 100 pour TBS selon itérations RSS pour RSS-TBS et BUSS-RSS-TBS
Nœuds CGP	300
Entrées Sorties	49 1 (2 classes)
Taille Tournoi	4
Probabilité de croisement Probabilité de mutation	0.9 0.04
Fitness	Minimiser les erreurs de classification

4.2.4.3 Paramètres des méthodes d'échantillonnage

Les valeurs des paramètres additionnels de chaque algorithme d'échantillonnage sont détaillés dans le tableau 4.2.

TABLE 4.2 – Paramètres des méthodes d'échantillonnage.

Méthode	Paramètre	Valeur
RSS	Taille cible	5000
TBS	Taille cible	1000
	Taux de perte	0.7
RSS-TBS	Taille cible (taille du bloc de niveau 2)	200
	Taille du bloc de niveau 0	1000
	Itérations RSS	50
	Max itérations TBS	20
	Individus évalués par itération TBS	100 (20% de la taille de la population)
BUSS-RSS-TBS	Taille cible (taille du bloc de niveau 2)	50 puis 200
	Taille du bloc de niveau 0	260 puis 416 (calculés)
	Itérations RSS	40 puis 50
	Max itérations TBS	50
	Individus évalués par itération TBS	100 (20% de la taille de la population)

4.2.4.4 Résultats

Le test des méthodes implémentées à travers des expériences a permis de collecter 6 mesures pour chacune d'elle : accuracy, TPR et FPR du meilleur individu et moyennes sur 21 runs et ce pour la base de test. Les mesures relatives au meilleur classifieur sont illustrées par la figure 4.3 tandis que les performances moyennes des meilleurs individus sont présentées dans la figure 4.2.

De plus, pour suivre l'évolution du coût de ces algorithmes, le temps moyen par run ainsi que le temps moyen pour trouver le meilleur classifieur sont reportés dans le tableau 4.3.

Les indicateurs enregistrés par la figure 4.3 du meilleur individu en termes du taux de succès de classification (accuracy) montrent que RSS-TBS et BUSS-RSS-TBS (avec ses 2 variantes) dépassent légèrement les performances des méthodes non hiérarchiques RSS et TBS. En outre, la valeur de TPR a nettement augmenté pour les trois méthodes hiérarchiques. Mais nous décelons une hausse remarquable de la valeur de FPR. Aussi, chaque fois que la deuxième configuration de BUSS est utilisée (dans BUSS2 et BUSS-RSS-TBS2), elle est accompagnée d'une diminution de la valeur de FPR.

La figure 4.2, qui présente les moyennes des différentes mesures, affiche des résultats moins performants pour les méthodes multi-niveaux pour les trois mesures (accuracy, TPR et FPR).

En se basant sur les mesures du tableau 4.3, il est notoire que les trois méthodes

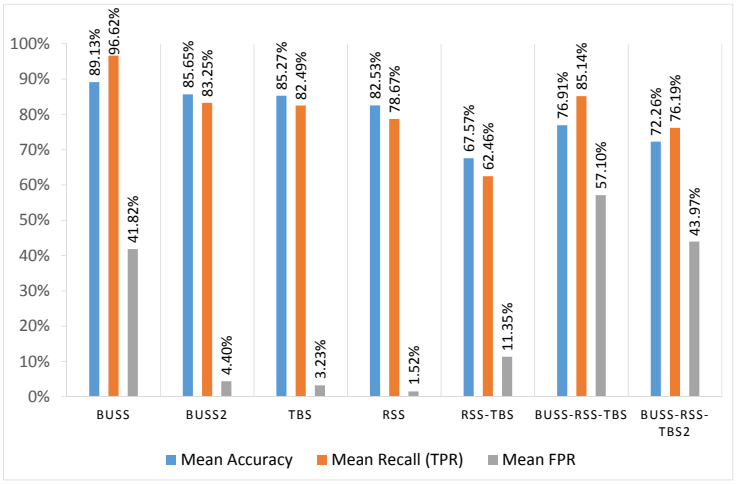


FIGURE 4.2 – Mesures moyennes sur la base de test.

TABLE 4.3 – Mesures du temps (en secondes).

Méthode	**Temps d'apprentissage moyen**	**Temps moyen de découverte de la meilleure solution**
BUSS	75.612	20.135
BUSS2	102.428	43.054
TBS	11174.507	1494.977
RSS	1493.788	530.038
RSS-TBS	714.414	9.524
BUSS-RSS-TBS	157.485	1.712
BUSS-RSS-TBS2	236.029	4.687

hiérarchiques sont plus rapides que TBS tout en faisant évoluer des populations de plus grande taille pour un nombre de générations largement supérieur grâce aux itérations sur deux niveaux.

Quant au temps mis pour trouver la solution, les nouvelles méthodes ont des temps plus courts par rapport au temps moyen d'apprentissage : la solution est découverte au début du run.

4.2.5 Discussion

Dans cette section, nous généralisons le concept d'échantillonnage hiérarchique tout en proposant une solution au problème de temps d'apprentissage de TBS avec les bases de grande taille. Cette solution consiste à l'intégrer dans une cascade de méthodes d'échantillonnage en s'inspirant de RSS-DSS [Curry & Heywood 2004].

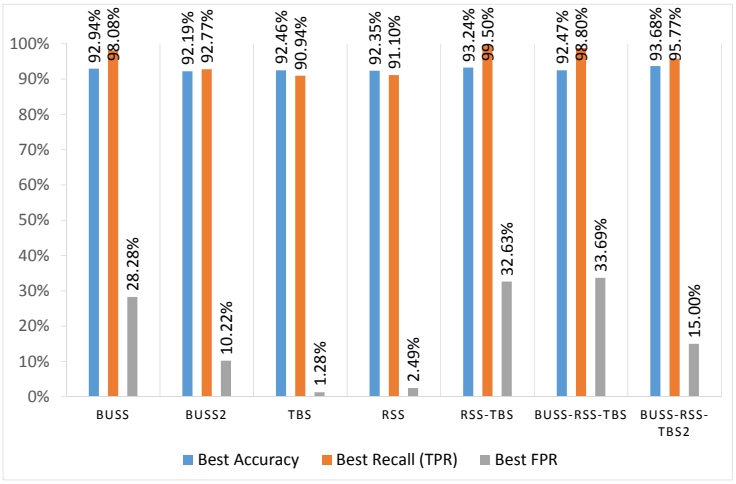


FIGURE 4.3 – Mesures du meilleur individu sur la base de test.

Dans un premier scénario, TBS est combiné avec RSS. Dans le second scénario, BUSS est injecté pour la construction des blocs au niveau 0.

Les résultats expérimentaux montrent un gain en temps d'apprentissage permettant l'application de TBS pour la base KDD-99. Cette amélioration conserve voire même améliore la valeur accuracy par rapport à TBS et RSS pris individuellement. Cependant, une hausse de FPR ainsi qu'une convergence rapide sont enregistrées avec ces méthodes.

Ces points peuvent être adressés par d'autres techniques comme l'utilisation d'une fonction de fitness adaptative. Une seconde amélioration possible est d'optimiser l'utilisation des graphes de topologie. Par exemple, la version que nous avons implémentée crée un nouveau graphe pour chaque bloc sélectionné par RSS et de ce fait, les anciennes mesures de similarité sont perdues. Mettre en place un graphe global pourrait à la fois améliorer les performances et gagner plus de temps.

4.3 Échantillonnage hiérarchique pour classifier les bosons de Higgs

4.3.1 Contexte

Après avoir étudié et testé les différentes méthodes d'échantillonnage actif et leur applicabilité en présence de larges bases, nous avons généralisé le concept d'échantillonnage multi-niveaux. Nous proposons, dans cette section, l'application des résultats de nos études pour la résolution d'un problème de classification

difficile à partir d'une base complexe classée comme Big Data. Nous avons choisi celle du problème de classification des bosons de Higgs [UCI 2014].

En 2014, un groupe de physiciens et « data scientists » de l'ATLAS ont organisé un défi d'apprentissage automatique permettant d'analyser la base Higgs. C'est une base de données de physique expérimentale à haute énergie [Adam-Bourdarios *et al.* 2014a]. La base utilisée pour le défi est une base dérivée. Les données proviennent de l'accélérateur de particules (LHC) permettant de produire des collisions entre protons et/ou antiprotons pour créer des particules exotiques à une très haute énergie. Le problème est que cet accélérateur produit une masse importante de données relatives aux collisions dont la majorité ne donne pas lieu aux particules recherchées. Par exemple, LHC produit environ 100 milliards de collisions par heure, c'est uniquement 300 qui donnent lieu à un boson de Higgs en moyenne [Baldi *et al.* 2014]. De ce fait, une bonne analyse de données est celle qui peut distinguer entre les collisions produisant la particule étudiée (signal) de celles produisant d'autres particules (background). Une description de la base Higgs est donnée dans la section 4.3.2.

Plusieurs techniques d'apprentissage automatique ont été investiguées comme l'apprentissage profond (Deep Neural Networks (DNN)) [Adam-Bourdarios *et al.* 2014a] et les arbres de décision [Chen & He 2014]. Gábor Melis, le vainqueur du « Higgs Boson challenge» grâce à un ensemble de 70 réseaux de neurones, déclare que son plan initial était d'utiliser un algorithme évolutionnaire mais il a dû abandonner à cause du problème de mise à l'échelle de ce dernier.

Les résultats que nous avons obtenus grâce à l'échantillonnage multi-niveaux [Hmida *et al.* 2016a] nous ont inspirés pour montrer que la GP peut être appliquée sur ce problème et proposer une solution qui étend la CGP avec la méthode RSS-DSS pour classifier les événements de la base Higgs. Dans ce choix, nous avons pris en considération le temps d'apprentissage vu la taille de la base considérée. De plus, RSS-DSS est le plus simple des algorithmes multi-niveaux étudiés à mettre en œuvre.

4.3.2 La base Higgs et travaux antérieurs

Un boson Z de Higgs est une particule résultant des collisions du LHC [Baldi *et al.* 2015] qui se désintègre rapidement sous forme d'autres particules plus stables, cet état intermédiaire n'est pas détectable par les détecteurs autour du champ de collision. Il est donc difficile de distinguer entre ces 2 processus qui donnent lieu aux mêmes particules stables sans étudier les états intermédiaires. Il est aussi possible d'étudier la direction et le moment des particules finales. Pour cela, des collisions sont simulées par le simulateur ATLAS [ATLAS collaboration 2014] selon la méthode Monte Carlo pour reproduire les mesures fournies par les détecteurs. Les variables de base sont appelées de bas-niveau. D'autres variables, dites de haut-niveau, sont construites à partir d'une combinaison non linéaire de celles de bas niveau. Elles permettent de mieux discriminer entre boson de Higgs et boson Z. La base obtenue compte 80 millions d'événements de collision, caractérisés par

28 attributs continus : 21 variables de bas niveau décrivant les moments 3D et les énergies des produits de collision et 7 variables de haut niveau (cf. annexe B).

Le problème correspond à un problème de classification binaire ; il s'agit de classifier les événements comme signal (classe positive) ou background.

Pour le défi sur Kaggle [1], une base d'apprentissage de 250K événements, une base de test de 550K événements et 30 attributs sont utilisés (une description détaillée est proposée dans [Adam-Bourdarios *et al.* 2014b]). La plupart des méthodes avancées d'apprentissage automatique ont été appliquées. Les solutions les mieux classées sont en majorité basées sur les ensembles d'arbres de décision notamment XGBoost [2]. La meilleure solution est celle de Gábor Melis [Melis 2014] est basée sur l'apprentissage profond.

Après ce défi, Baldi et al. [Baldi *et al.* 2014] ont publié une base contenant 11 millions d'exemples d'apprentissage et 28 attributs décrits ci-dessous. C'est la base que nous avons utilisée dans cette thèse. Il est à noter qu'au moment de la rédaction de ce chapitre, excepté le travail de Shashidhara et al. [Shashidhara *et al.* 2015], aucune autre étude n'a pris en compte la base entière (seulement des échantillons ne dépassant pas 2 millions évènements).

Le tableau 4.4 résume les différentes caractéristiques de cette base :

Table 4.4 – Base d'apprentissage Higgs.

Nombre total d'événements	11 millions
Nombre d'attributs	28 à valeurs continues (21 de bas niveau 7 de haut niveau)
Pourcentage de signaux	53%
Taille de la base d'apprentissage	10.5 millions
Taille de la base de test	500K

4.3.3 Extension de CGP avec l'échantillonnage actif

Nous avons choisi CGP comme algorithme pour l'apprentissage, celui utilisé pour l'étude comparative dans le chapitre 3. Deux challenges se posent avec la base Higgs : le volume et la complexité. Il est à noter qu'il est impossible d'appliquer CGP sur la base entière. Pour relever les deux challenges, nous appliquons une extension de CGP avec l'échantillonnage actif. Le choix est porté sur l'algorithme RSS-DSS qui est introduit afin de réduire la taille des données à évaluer par CGP. RSS-DSS est le premier algorithme de la classe des algorithmes multi-niveaux qui a montré une aptitude à traiter de larges bases données. Il est aussi le plus simple à mettre en œuvre par rapport aux autres algorithmes de la même classe.

1. Kaggle : https://www.kaggle.com
2. XGBoost : une implémentation du Gradient Boosting Trees.

4.3.3.1 Échantillonnage de la base d'apprentissage

Pour comparer l'efficacité de l'échantillonnage avec la base Higgs, une technique de type actif à 1-niveau et une deuxième de type 2-niveau sont considérées à savoir RSS et RSS-DSS. L'implémentation de RSS reprend celle effectuée dans le chapitre 3. RSS-DSS suit un processus sur 3 niveaux. Au niveau 0, la base originale est subdivisée en blocs de même taille enregistrés sur disque. Ensuite, au niveau 1, un bloc est choisi de façon aléatoire. Enfin, le niveau 2, applique DSS sur le bloc du niveau 1. L'évaluation de la population est effectuée sur tout l'échantillon sans utiliser un mécanisme de tournoi puisque la population n'est pas de grande taille. Aussi, le nombre d'itérations du niveau 2 est fixé (sans calcul) comme un paramètre (algorithme 12) .

Algorithme 12 CGP + RSS-DSS

Parameters
$\mathcal{S}$: base d'apprentissage
T_{l1} : itérations maximum de niveau 1
T_{l2} : itérations maximum de niveau 2
Partitionner $\mathcal{S}$ en blocs *%niveau 0*
répéter *%itérations de niveau 1*
  Sélectionner d'un bloc aléatoire avec RSS *%niveau 1*
  réinitialiser les vecteur d'âge et de difficulté ($A(g)$ et $D(g)$)
  répéter *%itérations de niveau 2*
    Sélection d'instance avec DSS *%niveau 2*
    Évaluation des individus sur l'échantillon DSS
    Mise à jour du vecteur $A(g)$
    Mise à jour du vecteur $D(g)$
    Évaluation du meilleur individu de la génération sur le bloc RSS
    Mise à jour du meilleur individu du run
    Application des opérateurs génétiques
  jusqu'à fin niveau 2
jusqu'à fin niveau 1

4.3.3.2 Mesures de performance

Á la fin de chaque run, le meilleur individu est évalué sur la totalité de la base de test. Les résultats sont enregistrés sous forme de matrice de confusion à partir de laquelle les mesures de accuracy, TPR et FPR sont calculées (voir équations 3.10, 3.11 et 3.12). La fonction objectif est de maximiser le taux de bonnes classifications.

Par ailleurs, le classement des participants au défi sur Kaggle est basé sur la mesure AMS (Approximate Median Significance) (eq. 4.1). Elle est calculée à partir des valeurs FPR et TPR et vise à pénaliser les faux positifs. Une description plus détaillée est donnée dans [Adam-Bourdarios *et al.* 2016]. Plusieurs formules ont

été proposées ; nous utilisons celle dans [Adam-Bourdarios *et al.* 2014b] qui est comme suit :

$$AMS = \frac{TPR}{\sqrt{FPR + (FPR \times \varepsilon)^2}} \text{ where } \varepsilon < 0.1 \tag{4.1}$$

4.3.4 Paramètres expérimentaux

Ensembles de terminaux et fonctions : Par rapport à l'étude comparative dans 3.5, c'est le nombre de terminaux qui a subi un changement pour devenir 28 attributs provenant de la base Higgs (décrits dans la section 4.3.2).

Paramètres CGP et échantillonnage : Quelques séries de runs ont permis d'ajuster les paramètres de CGP qui sont récapitulés dans le tableau 4.5.

Table 4.5 – Paramètres CGP.

Paramètre	Valeur
Taille de la population	128
Nombre de générations	3500
Nœuds CGP	500
CGP levels-back	3
Entrées/Sorties	36/1
Taille tournoi	4
Probabilité de croisement	0.9
Probabilité de mutation	0.04

Les paramètres de l'échantillonnage actif sont exposés dans le tableau 4.6. RSS possède un paramètre unique qui est la taille cible. Il est identique à celui du niveau 2 de l'algorithme RSS-DSS.

RSS-DSS s'appuie sur un nombre supérieur de paramètres dont les valeurs sont ajustées en concordance avec ceux du chapitre 3. Avec des réglages différents pour 3 paramètres : taille de l'échantillon de niveau 2, le nombre des itérations RSS et les itérations DSS.

Table 4.6 – Paramètres d'échantillonnage.

Méthode	Paramètre	Valeur
RSS	Taille échantillon	7000
RSS-DSS	Taille échantillon (niveau 2)	7000
	Taille bloc de niveau 0	50000
	Itérations RSS	50
	Itérations DSS	70
	Exposants de Difficulté/Âge	1/3.5
	Roulette Difficulté/Âge	70%/30%

4.3.5 Résultats et discussion

Les expériences sont conduites dans les mêmes environnements logiciel et matériel utilisés dans l'étude comparative 3.5.

Neuf mesures sont enregistrées sur les 5 runs par configuration : accuracy, TPR, FPR et AMS pour le meilleur individu ainsi que la valeur moyenne entre les meilleurs individus des runs exécutés. Pour calculer AMS, ε (eq. 4.1) est égal à 0.05. Les valeurs correspondantes sont reportées dans le tableau 4.7.

TABLE 4.7 – Mesures de performance sur la base de test.

		Accuracy	**TPR**	**FPR**	**AMS**
RSS	Meilleur	0,637854	0,6803448	0,3139881	1,214
	Moyenne	0,6293352	0,6620886	0,35647769	1,08
RSS-DSS	Meilleur	0,65038	0,686341	0,39	1,1
	Moyenne	0,6459072	0,6426552	0,3504401	1,0855

Le défi de cette étude est de prouver que la GP, à travers une extension, est capable de traiter et d'apprendre à partir de très larges bases de données. Cet objectif est atteint et CGP est appliquée dans le problème de détection des bosons de Higgs utilisant une base de 10.5 millions d'observations dans un temps raisonnable.

En effet, avec la configuration décrite dans le tableau 4.5, un run dure aux alentours de 2 heures et demie. Ce temps est, très inférieur à celui de CGP sans échantillonnage. En effet, nous n'avons pas pu effectuer un run complet de CGP sur la totalité de la base d'apprentissage mais l'évaluation d'un seul individu sur cet ensemble requiert plus de 20 secondes. Par interpolation, le temps d'apprentissage nécessaire d'un run avec une population de 128 individus et pendant 500 génération serait de 355 heures (2 semaines).

De plus, ce temps est meilleur que d'autres techniques d'apprentissage automatique comme la régression logistique ou SVM linéaire (table 4.8).

Quant aux performances d'apprentissage, les deux méthodes d'échantillonnage ont des résultats très proches (tableau 4.7).

Nous n'avons pas de résultats pour CGP ou autre variante de GP pour Higgs pour les utiliser comme référence. Les résultats les plus connus pour la détection des bosons de Higgs sont ceux obtenus lors du défi organisé sur Kaggle. Cependant, nous ne pouvons pas nous y comparer pour les raisons suivantes :

- la base de données utilisée par Kaggle est différente et possède un nombre d'événements très inférieur à celui de la base utilisée dans ce travail,
- l'ensemble des attributs est différent ; il est composé de 30 attributs dont 13 sont dérivés et sélectionnés par les physiciens d'ATLAS,
- le classement est basé sur la mesure AMS calculé avec une formule différente qui prend en considération des poids affecté à chaque ligne de la base de données par des experts,

– les techniques d'amélioration de performances d'apprentissage sont appliquées par les différents participants comme la réduction des attributs, ensemble de classifieurs, bagging, etc.

De plus, Baldi et al. [Baldi *et al.* 2015] ont appliqué les DNN sur un sous-ensemble de la base comportant 2.6 millions d'enregistrements et 100 mille pour la base de test. Ces sous-ensembles n'ont pas été publiés et par la suite la comparaison n'est pas possible avec leurs résultats.

Ce sont les travaux de Shashidhara et al. [Shashidhara *et al.* 2015] qui utilisent la configuration la plus proche de celle sur laquelle nous nous sommes basés. Les résultats exposés concernent l'évaluation de certaines librairies d'apprentissage automatique sur le problème des bosons de Higgs en utilisant la même base d'apprentissage. Les auteurs ont testé la régression logistique et SVM linéaire avec les implémentations fournies par Weka, Scikit-Learn et Spark. Seul Spark a réussi à exécuter ces algorithmes à cause de la taille de la base de données. Les résultats que nous avons obtenus avec CGP étendu par l'échantillonnage RSS-DSS sont meilleurs que ceux de Shashidhara et al. (tableau 4.8) selon la valeur accuracy avec un temps réduit d'un facteur avoisinant 50%.

TABLE 4.8 – Résultats de Shashidhara et al. [Shashidhara *et al.* 2015].

Classifieur	**Temps d'apprentissage**	**Accuracy**
Régression logistique	4 heures	0.6076
SVM	5 heures	0.5290

Grâce à l'échantillonnage multi-niveaux, CGP est devenue applicable sur un problème Big Data ; elle est capable de produire des résultats satisfaisants par rapport au temps et qualité d'apprentissage. Toutefois, des améliorations peuvent être apportées pour obtenir des résultats plus performants.

En premier lieu, l'ajustement des paramètres comme la taille des blocs, la taille de l'échantillon de niveau 2, etc. peut être étudié de manière plus approfondie pour réduire le temps et avoir des modèles plus performants.

De plus, la valeur du score AMS reste insatisfaisante. Notre solution utilise une fonction de fitness qui vise l'amélioration du pourcentage de bonne classification (accuracy). L'étude de l'utilisation d'une autre fonction de fitness qui prend en considération cette mesure pourrait améliorer les performances globales.

4.4 Échantillonnage adaptatif

4.4.1 Contexte et motivations

En examinant la terminologie donnée dans le chapitre 3 ainsi que le tableau 3.2, nous avons constaté que la totalité des méthodes d'échantillonnage dite actives sont proposées avec une fréquence d'échantillonnage générationnelle. En effet, ces

méthodes utilisent un échantillon renouvelé au début de chaque génération. Par comparaison avec une approche qui n'utilise pas l'échantillonnage, les données d'apprentissage sont les mêmes pour tous les individus durant tout le processus d'apprentissage. Nous sommes passés, avec l'échantillonnage actif, d'un ensemble d'apprentissage stable vers un ensemble assez variable.

Il est à noter que le taux de renouvellement des exemples d'apprentissage d'une génération par rapport à celle qui la précède dépend de la stratégie d'échantillonnage. Des algorithmes comme IDI, contrôlent cette diversité par un paramètre qui permet d'injecter progressivement de nouveaux exemples d'apprentissage à l'échantillon précédent. DSS maintient les exemples d'apprentissage les plus difficiles à classifier dans tous les échantillons. Pour RSS, ceci est lié à une probabilité.

Intuitivement, avec la taille réduite des échantillons, une population a besoin de suffisamment de temps (générations) pour tirer profit des exemples d'apprentissage sans pour autant tomber dans le cas de sur-apprentissage. Un renouvellement systématique de l'échantillon ne laisse pas ce temps nécessaire à chaque classifieur pour adapter son génome.

De plus, l'opération de sélection des exemples d'apprentissage réitérée à chaque génération génère un temps supplémentaire (au temps d'apprentissage) qui peut être assez important.

Nous proposons, dans cette section, une extension applicable à tous les algorithmes d'échantillonnage actif. Dans cette extension, le renouvellement de l'échantillonnage n'est plus systématique mais dépend d'un nouveau paramètre. Ce paramètre est assez flexible pour implémenter différentes stratégies pour contrôler le renouvellement de l'échantillon d'apprentissage.

4.4.2 Contrôler la fréquence d'échantillonnage

4.4.2.1 Principe de l'échantillonnage adaptatif

La fréquence d'échantillonnage (f) est un paramètre principal pour toute technique d'échantillonnage actif. Elle définit la fréquence à laquelle le sous-ensemble de d'apprentissage est renouvelé au cours du processus d'apprentissage. Lorsque $f = 1$, l'échantillon d'apprentissage est extrait à chaque génération. La méthode d'échantillonnage est dite *par-génération* (cf. chapitre 3). La plupart des techniques d'échantillonnage appliquées avec GP appartiennent à cette catégorie. Dans cette situation, les individus de la population actuelle ne disposent que d'une génération pour adapter leur génotype à l'environnement actuel caractérisé par l'échantillon d'apprentissage. Il est très difficile, voire impossible pour une population de résoudre tous les exemples dans un ensemble d'apprentissage en une génération.

Par ailleurs, une valeur élevée de f correspond à un nombre réduit d'échantillons à générer par run. Chaque échantillon est traité par la population durant plusieurs générations. Ceci pourrait avoir deux conséquences négatives : la première est le gaspillage du temps de calcul lorsque la population s'est adaptée à

l'échantillon et un risque de sur-apprentissage. La deuxième est le risque, qui accompagne le nombre réduit d'échantillons, de ne pas sélectionner des exemples d'apprentissage ayant le potentiel d'améliorer la qualité des classifieurs de la population.

Nous pensons que la fréquence d'échantillonnage doit être mise à jour en fonction de l'état d'évolution et de la difficulté de l'échantillon courant.

L'idée de l'échantillonnage adaptatif est d'ajouter un paramètre supplémentaire aux algorithmes d'échantillonnage jouant le rôle de modérateur ou régulateur de ré-échantillonnage en contrôlant la fréquence de renouvellement et la création des sous-ensembles d'apprentissage. Par opposition aux méthodes dites actives qui renouvellent l'échantillon avec une fréquence fixe égale à 1, l'échantillonnage adaptatif repose sur une condition devant être vraie pour générer un nouvel échantillon à utiliser dans les générations qui suivent.

La figure 4.4 illustre ce principe. Le moteur d'apprentissage à base de GP interagit avec le processus d'échantillonnage en lui fournissant les données sur l'état de l'apprentissage qui varient selon l'algorithme d'échantillonnage. Par exemple, DSS a besoin de savoir les exemples d'apprentissage mal classés par chaque individu de la population pour mettre à jour sa difficulté. L'algorithme d'échantillonnage actif répond par la génération d'un nouvel échantillon. Avec l'échantillonnage adaptatif, un prédicat doit être vérifié avant de procéder au ré-échantillonnage. Nous supposons que toutes les données requises pour évaluer ce prédicat sont disponibles dans les données fournies par le moteur GP.

Plusieurs prédicats peuvent être définis pour contrôler l'échantillonnage en se basant sur des paramètres comme :

- Nombre de générations,
- Fitness moyenne de la population,
- Taux d'amélioration de la fitness moyenne,
- Taux d'amélioration de la meilleure fitness,
- etc.

Il s'agit de définir des seuils pour ces indicateurs : le ré-échantillonnage est déclenché si la mesure de cet indicateur pour la génération en cours est supérieure ou inférieure à ce seuil.

Un exemple est de fixer à 0.002 le taux d'amélioration de la meilleure fitness. Dans ce cas, la GP continue à utiliser le même échantillon tant que le meilleur individu de la génération en cours fait des progrès par rapport à celui de la génération précédente de plus de 0.002. Sinon, un nouvel échantillon sera généré.

Le seuil lui-même peut avoir une valeur fixe ou variable en s'adaptant au processus d'apprentissage.

Dans la suite de cette section, nous donnons une implémentation de ce principe et les résultats d'une série d'expérimentations afin de montrer les apports de cette approche. Nous proposons de nouvelles techniques pour adapter la fréquence d'échantillonnage. Nous commençons par un bref rappel de la méthode classique

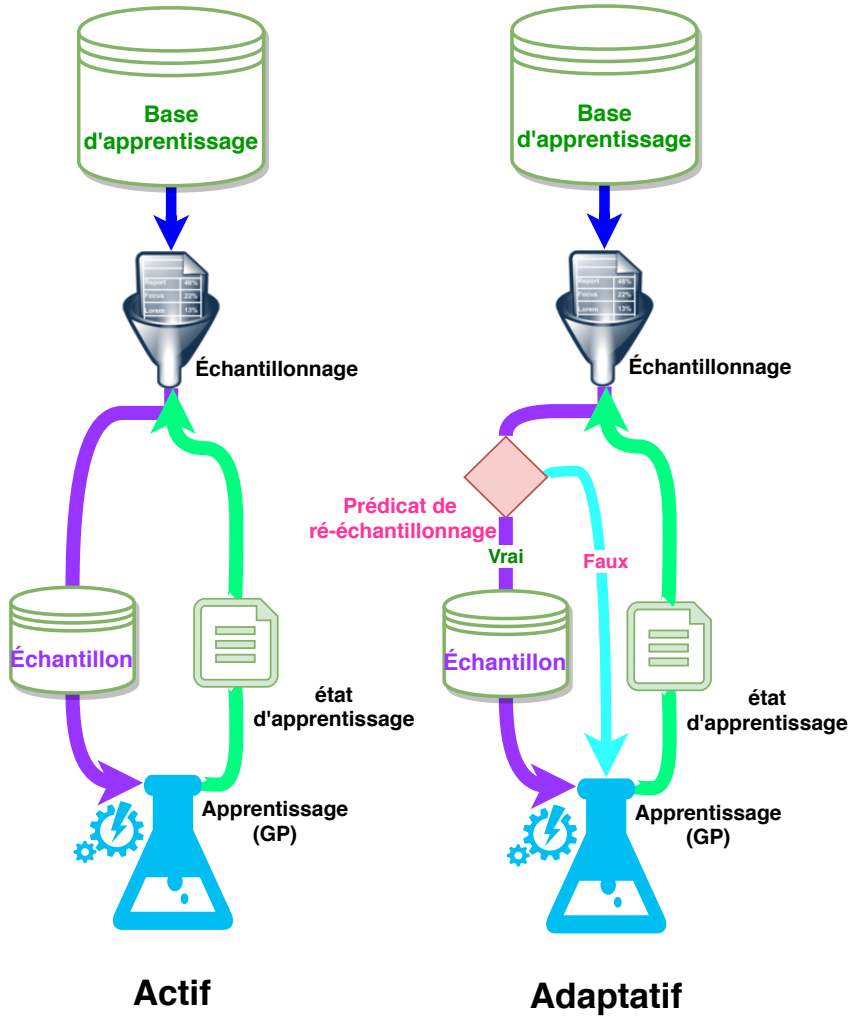


FIGURE 4.4 – Échantillonnage actif et adaptatif.

considérant la fréquence d'échantillonnage en tant que paramètre d'entrée avec une valeur fixe généralement définie sur 1. Ensuite, nous introduisons la fréquence d'échantillonnage déterministe dont la valeur évolue selon un schéma prédéfini (basé sur une fonction mathématique). Nous présentons ensuite la fréquence d'échantillonnage adaptative qui dépend de l'état actuel du processus d'apprentissage. Elle est basée soit sur l'évolution de la fitness moyenne, soit sur le nombre d'exemples résolus pour décider de créer un nouvel échantillon ou de poursuivre l'apprentissage avec l'échantillon précédent.

4.4.2.2 Fréquence d'échantillonnage fixe

La valeur de la fréquence d'échantillonnage f est affectée avant de démarrer le run GP comme tous les autres paramètres de GP. Cette demeure inchangée jusqu'à la dernière génération. Ceci est représenté par l'algorithme 13.

Algorithme 13 Échantillonnage à fréquence fixe.

Paramètres : f : fréquence d'échantillonnage
1: **pour tout** génération $g < g_{max}$ **faire**
2: **si** $g \mod f = 0$ **alors**
3: générer un nouvel échantillon
4: **fin si**
5: **fin pour**

4.4.2.3 Fréquence d'échantillonnage déterministe

Quand la fréquence d'échantillonnage est contrôlée de façon déterministe, f prend différentes valeurs tout au long du run GP. Ces valeurs sont déterminées par une fonction qui donne les mêmes valeurs à chaque run. Ainsi, la fréquence peut être croissante, décroissante ou suit une courbe complexe.

Les étapes décrites ci-dessous permettent un contrôle déterministe de f. La fréquence est croissante, elle commence par des échantillons ayant une courte durée de vie (en nombre de générations) et finit par des échantillons qui sont utilisés pendant un grand nombre de générations. La fréquence est calculée à chaque génération par la fonction $f = (C \times g)^{\alpha}$. Les coefficients réels C et α contrôlent l'allure de la courbe des valeurs de f.

Algorithme 14 Fréquence d'échantillonnage déterministe.

1: **pour tout** génération $g < g_{max}$ **faire**
2: $f = (C \times g)^{\alpha}$ %$C, \alpha \in \mathbb{R}$
3: **si** $g \mod f = 0$ **alors**
4: générer un nouvel échantillon
5: **fin si**
6: **fin pour**

Pour avoir un comportement inverse (i.e. fréquence décroissante), nous pouvons utiliser les mêmes étapes avec une fonction comme : $f = (C \times (g_{max} - g))^{\alpha}$.

4.4.2.4 Fréquence d'échantillonnage adaptative

Cette technique ajuste la fréquence d'échantillonnage pour l'accommoder à l'état actuel du processus d'apprentissage. Par conséquent, la fréquence peut augmenter ou diminuer d'une valeur variable. Nous supposons que les classifieurs moins performants ont besoin de plus de temps pour s'améliorer. Symétriquement, ceux qui sont assez performants sur un échantillon doivent examiner de nouveaux exemples d'apprentissage.

L'échantillonnage adaptatif peut reposer sur différents indicateurs de performance d'apprentissage. Il reçoit ces indicateurs du moteur GP. Ci-après, nous proposons deux exemples de cette technique.

Le premier exemple, utilise un seuil pour la fitness moyenne de la population pour détecter si les individus sont en train de s'améliorer ou non. Dans le cas négatif, un nouvel échantillon est créé en supposant que le précédent est totalement exploité par la population.

Algorithme 15 Fréquence d'échantillonnage adaptative

Paramètres : r : taux de variation de la fitness moyenne = (fitness moyenne(g)-fitness moyenne(g-1))/fitness moyenne(g-1), t : seuil

1: **pour tout** génération $g < g_{max}$ **faire**
2: **si** $r > t$ **alors**
3: générer un nouvel échantillon
4: **fin si**
5: **fin pour**

Le deuxième exemple se base sur la mesure du nombre moyen des individus ayant correctement classifié chaque exemple d'apprentissage de l'échantillon. Si cette valeur a atteint un seuil alors de nouveaux exemples d'apprentissage seront sélectionnés.

Dans les paragraphes suivants, nous donnons les détails sur l'étude expérimentale de l'échantillonnage adaptatif appliqué sur quelques algorithmes d'échantillonnage dynamique. Puis, nous exposons les résultats obtenus et nous analysons l'effet de la fréquence d'échantillonnage et l'échantillonnage adaptatif sur les performances de GP pour la résolution du problème KDD de détection d'intrusions.

4.4.3 Étude expérimentale

4.4.3.1 Paramètres expérimentaux

Dans ces expériences, nous utilisons les mêmes conditions que celles de l'étude comparative du chapitre 3. En effet, GP est appliqué sur le problème de détection d'intrusion KDD'99 dont la base de données est décrite dans le tableau 3.3. Nous nous limitons aux algorithmes BRSS, BUSS (la deuxième implémentation), DSS et RSS qui sont les plus simples à implémenter et à modifier. Aussi, nous utilisons les mêmes paramètres de CGP (voir les tableaux 3.4, 3.5 et 3.6) mais avec un nombre de générations total de 200, une taille de la population égale à 256 et une taille d'échantillon de 5000.

4.4.3.2 Paramètres de l'échantillonnage

Dans la première partie des expériences, nous avons testé six valeurs de la fréquence d'échantillonnage : 1, 10, 20, 30, 40 et 50. Dans la seconde partie, nous avons implémenté 4 différentes techniques pour contrôler la fréquence d'échantillonnage. Deux sont à fréquence déterministe et deux à fréquence adaptative comme suit :

- *Deterministic+* : contrôle déterministe avec la fonction croissante $f = (2 \times g)^{0.5}$,
- *Deterministic-* : contrôle déterministe avec la fonction décroissante $f = (2 \times (g_{max} - g))^{0.5}$,
- *Average Fitness* : contrôle adaptatif basé sur l'amélioration de la fitness moyenne de la population avec un seuil de 0.001,
- *Min Resolved* : contrôle adaptatif basé sur la proportion moyenne de la population représentant les individus ayant résolu les exemples d'apprentissage. Nous avons utilisé un seuil minimum de 0.5.

Quant aux paramètres des algorithmes d'échantillonnage actif, ils sont récapitulés dans le tableau 4.9.

TABLE 4.9 – Paramètres des algorithmes d'échantillonnage.

Méthode	**Paramètre**	**Valeur**
Toutes (sauf BUSS)	Taille cible	5000
BRSS	Méthode d'équilibrage	Distribution des classes de la base d'origine
BUSS	Taille cible	416
DSS	Exposant de difficulté	1
	Exposant de l'âge	3.5

4.4.4 Résultats

Cette étude est organisée en 2 catégories d'expériences. La première vise à étudier l'impact de la variation de la fréquence d'échantillonnage sur les mesures de la performance d'apprentissage.

Pour chaque valeur de la fréquence de ré-échantillonnage et technique de contrôle, 21 runs de chaque algorithme sont exécutés. Nous enregistrons le temps moyen d'apprentissage de chaque configuration et les valeurs accuracy et FPR du meilleur individu sur les bases d'apprentissage et de test.

4.4.4.1 Effet de la fréquence de ré-échantillonnage

La figure 4.5 représente la variation du temps d'apprentissage en fonction de la fréquence de ré-échantillonnage pour les 4 algorithmes étudiés. Les figures 4.6 et 4.7 montrent la variation des 2 mesures de qualité d'apprentissage du meilleur individu sur la bases d'apprentissage et de test à savoir : accuracy et FPR.

L'observation de l'allure des courbes de la figure 4.5 permet de détecter deux comportements distincts lorsque la fréquence d'échantillonnage augmente. Le premier concerne les algorithmes BUSS, BRSS et RSS qui ont enregistré une variation insignifiante qui va dans les sens d'une diminution du temps d'apprentissage

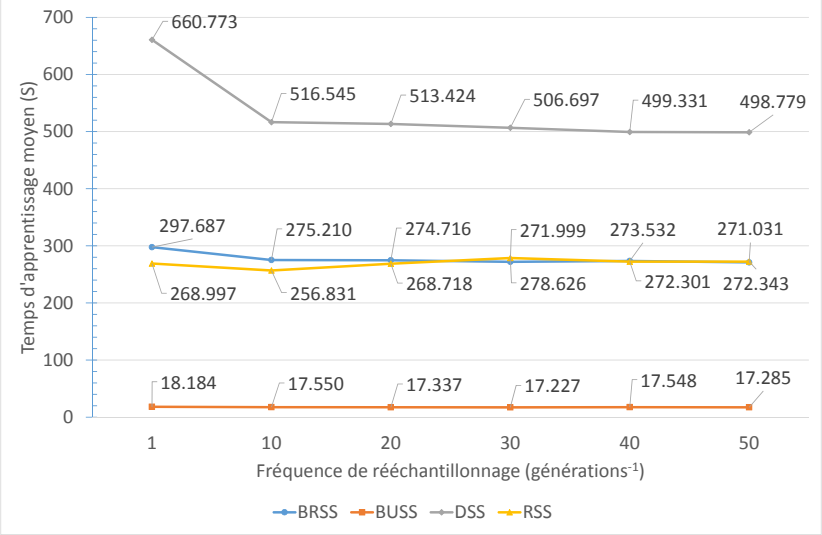


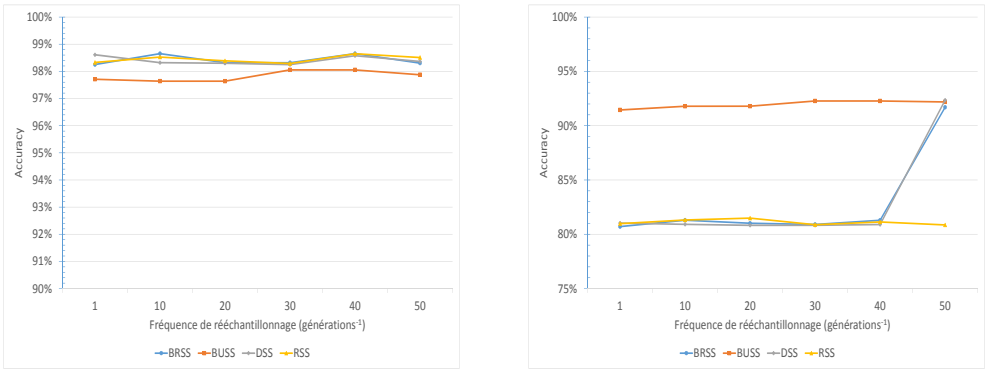
FIGURE 4.5 – Variation du temps d'apprentissage moyen selon la fréquence de ré-échantillonnage.

moyen. Le deuxième comportement est celui de DSS ayant une diminution plus remarquable.

En effet, pour une fréquence égale à 1, 200 échantillonnages sont effectués et pour une fréquence de 50 ce nombre descend à 4 échantillonnages. Le gain de temps dépend du temps de sélection des exemples d'apprentissage par rapport au temps d'une génération. C'est pour cela que le gain en temps n'est pas très important vu que c'est le temps d'évaluation de la population qui est le prédominant dans le temps d'apprentissage pour la GP.

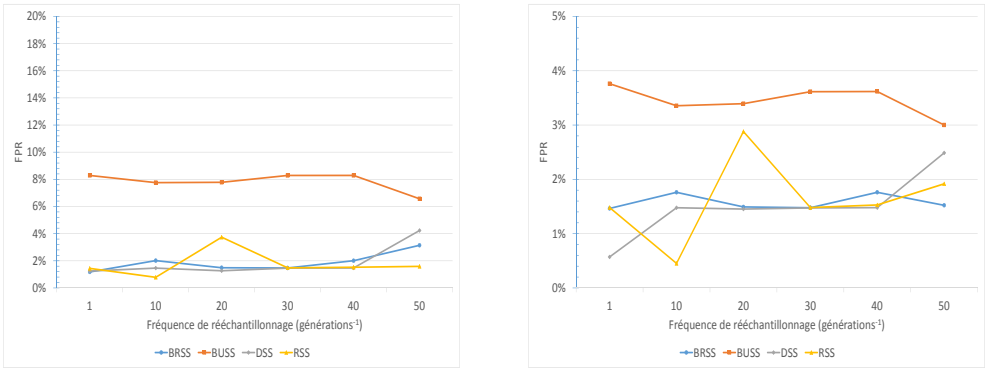
L'algorithme DSS se distingue, par rapport aux autres techniques, par une mise à jour de certains paramètres (âge et difficulté). Mais elle reste sans incidence sur le gain du temps d'apprentissage puisqu'elle est effectuée à chaque génération indépendamment de la fréquence de ré-échantillonnage. Dans DSS, la sélection d'un exemple d'apprentissage nécessite le calcul d'une probabilité à partir des valeurs d'âge et de difficulté de la totalité de la base d'apprentissage. Ceci engendre un temps plus grand que les autres techniques qui est à l'origine de la différence dans le gain de temps.

À partir des deux courbes de la figure 4.6, la variation du taux de bonne classification, contrairement à l'hypothèse de départ, n'a enregistré aucune amélioration pour les 2 bases. Par contre, il y a une augmentation de l'Accuracy au niveau de la fréquence 50, pour DSS et BRSS sur la base de test. Toutefois, ceci est ponctuel et ne peut être généralisé. L'utilisation du prédicat « fréquence de ré-échantillonnage » ne semble donc pas avoir d'effet notable sur l'Accuracy des



(a) Base d'apprentissage (b) Base de test

FIGURE 4.6 – Variation de l'accuracy du meilleur individu selon la fréquence de ré-échantillonnage.



(a) Base d'apprentissage (b) Base de test

FIGURE 4.7 – Variation du FPR du meilleur individu selon la fréquence de ré-échantillonnage

algorithmes considérés dans ces expériences.

La variation la plus remarquable est celle de la FPR (figure 4.7) sans pouvoir déduire une corrélation empirique avec la variation de la fréquence de ré-échantillonnage pour BRSS, DSS et RSS. Uniquement BUSS a vu sa valeur FPR diminuer avec l'augmentation de cette fréquence et ce aussi bien avec la base d'apprentissage qu'avec la base de test.

4.4.4.2 Échantillonnage adaptatif

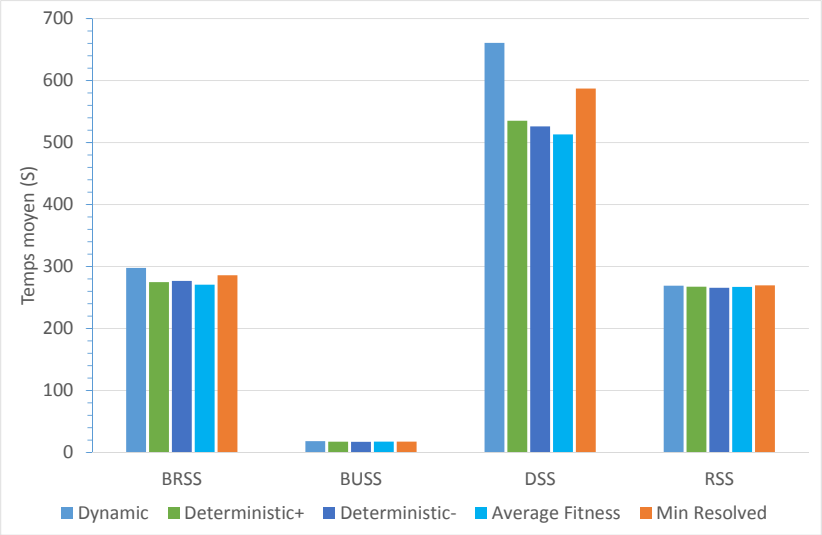
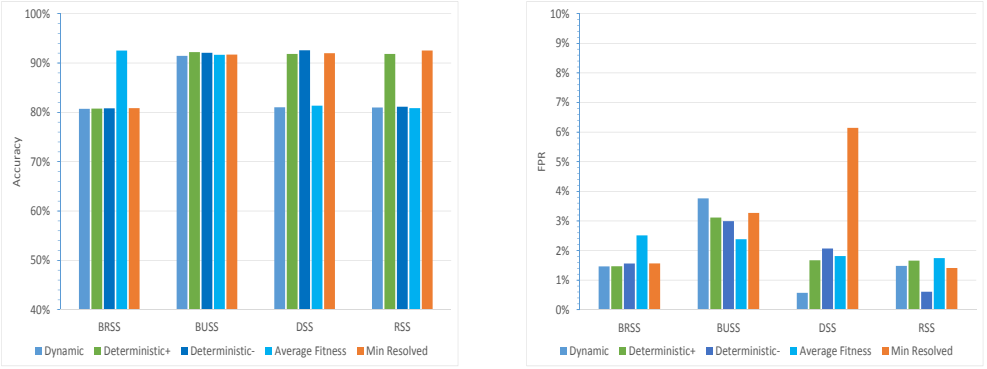


FIGURE 4.8 – Variation de temps d'apprentissage moyen selon la stratégie de contrôle de la fréquence de ré-échantillonnage.

Les figures 4.8 et 4.9 montrent les résultats des 4 algorithmes étendus avec 4 techniques pour le contrôle de la fréquence d'échantillonnage (section 4.4.2) : déterministe basée sur une fonction croissante puis décroissante (notées Deterministic+ et Deterministic-) et adaptative basée sur la fitness moyenne de la population (Average Fitness) ou le nombre moyen des individus ayant résolu les exemples d'apprentissage (Min Resolved). Les résultats obtenus sont comparés à ceux de l'algorithme dynamique original.

Concernant le temps moyen d'apprentissage, les résultats de la figure 4.8 confirment les précédentes observations autour de ce comportement dans la section 4.4.4.1. C'est seulement le temps d'apprentissage de DSS qui est clairement affecté. Ceci est vrai pour les deux techniques déterministes et adaptatives. Cet effet est proportionnel au nombre d'échantillon par run GP.

Quant à la performance d'apprentissage sur la base de test, les résultats ne



(a) Accuracy (b) FPR

FIGURE 4.9 – Variation de l'accuracy et FPR selon la stratégie de contrôle de la fréquence de ré-échantillonnage.

révèlent une réponse généralisable à l'introduction des stratégies de contrôle par les techniques étudiées. Par ailleurs, BUSS montre une légère amélioration de son accuracy avec ces techniques de contrôle. De plus, la valeur de FPR a fait la même variation et dans la même direction (figure 4.9).

Pour chaque méthode d'échantillonnage, il existe au moins une technique de contrôle de fréquence qui lui permet d'améliorer, en grande partie ou légèrement, ses performances d'apprentissage. Par exemple, pour le RSS, les techniques *Deterministic+* et *Min Resolved* ont permis à la valeur accuracy de passer de valeurs autour de 80% à des valeurs supérieures à 90%. De même, les deux méthodes déterministes (*Deterministic+* et *Deterministic-*) et la méthode adaptative *Min Resolved* ont amélioré la performance de DSS en augmentant son accuracy de plus de 10%. Cependant, une détérioration du FPR a été également enregistrée.

Pour BUSS, seule la technique de contrôle adaptative *Mean Fitness* a réussi à améliorer son accuracy. Toutefois, les autres techniques n'ont aucune incidence négative sur ses performances par rapport à la fréquence *par-génération* (Dynamic). En effet, le contrôle de la fréquence d'échantillonnage, s'il ne permet pas une amélioration des résultats, il ne génère pas une dégradation de la performance, sauf dans certains cas pour la mesure FPR.

4.4.5 Discussion

L'idée de l'échantillonnage adaptatif trouve ses origines dans l'hypothèse qui dit qu'une population a besoin d'un temps suffisant (générations) pour s'adapter aux données d'apprentissage. Nous avons formalisé ce principe sous la forme d'un prédicat de ré-échantillonnage qui se base sur une condition pour décider de modifier l'échantillon en cours.

Les expériences sont limitées au test de l'échantillonnage adaptatif par le

contrôle de la fréquence avec des prédicats simples. Les résultats ont montré un effet léger sur le temps d'apprentissage. Cet effet va dans le sens d'une diminution mais avec des degrés différents selon la méthode d'échantillonnage.

Avec l'échantillonnage adaptatif, la diminution du temps d'apprentissage par rapport à l'algorithme dynamique est possible si le processus de sélection des exemples d'apprentissage est un processus coûteux en temps de calcul. C'est le cas de DSS.

Les prédicats utilisés n'ont pas dégradé les indicateurs de performance d'apprentissage. Toutefois, ils n'ont pas eu d'apport généralisable sur la qualité de l'apprentissage. Une étude expérimentale plus étendue utilisant d'autres bases de données permettrait de mieux généraliser les extensions proposées. Par ailleurs, l'exploration d'autres prédicats qui prennent en compte les caractéristiques de la base d'apprentissage et le problème étudié pourrait permettre d'identifier des prédicats plus pertinents pour l'amélioration des classifieurs obtenus par GP.

Plusieurs nouvelles pistes de recherche émergent de cette étude et méritent d'être approfondies. Une première voie consiste à explorer d'autres prédicats qui prennent en compte les caractéristiques de la base d'apprentissage et le problème sous-jacent afin de trouver des prédicats plus pertinents pour les améliorations du classificateur GP. Une deuxième consiste à étendre la portée de l'échantillonnage adaptatif à d'autres propriétés de l'échantillon. Par exemple, un échantillonnage adaptatif peut réduire ou augmenter la taille de l'échantillon au lieu d'en générer un nouveau. Nous pouvons également combiner plusieurs stratégies et algorithmes d'échantillonnage dans une même méthode. Ensuite, en fonction de l'évolution de l'apprentissage, un échantillon est généré en utilisant la stratégie appropriée de manière entrelacée : nous utilisons un algorithme différent à chaque fois que nous devons créer un nouvel échantillon.

4.5 Conclusion

Dans ce chapitre, 3 extensions ont été présentées dont 2 sont basées sur l'échantillonnage hiérarchique et une introduit l'échantillonnage adaptatif.

La première extension est effectuée sur la méthode d'échantillonnage TBS qui a prouvé son efficacité avec les bases d'apprentissage de taille réduite mais ne peut être appliquée à de très larges bases de données. Nous l'avons hiérarchisée selon deux configurations appelées RSS-TBS et BUSS-RSS-TBS qui ont permis d'utiliser TBS avec GP sur la base KDD'99.

La deuxième extension est celle d'introduire l'échantillonnage hiérarchique afin de pouvoir appliquer CGP à un problème que nous pouvons qualifier de « Big Data » qui est celui de la détection des bosons de Higgs. Bien que nous ayons obtenu des résultats comparables à d'anciens travaux pour la même base, ces résultats restent perfectibles. L'ajustement des paramètres, un domaine à part entière pour la GP, reste à examiner pour améliorer encore la qualité des classifieurs. Les deux premières études démontrent que l'échantillonnage multi-niveaux peut

être généralisé et considéré comme une solution confirmée pour l'apprentissage à partir de grandes bases de données.

La troisième extension concerne un nouveau modèle d'échantillonnage applicable à toutes les méthodes d'échantillonnage de type 1-niveau. Il permet de contrôler l'opération de génération d'un nouvel échantillon en s'appuyant sur une ou plusieurs conditions. Le prédicat basé sur un nombre de générations fixe n'a pas apporté d'améliorations très remarquables mais n'a pas non plus détérioré les résultats acquis.

L'ensemble des travaux menés dans ce chapitre ouvre plusieurs voies de recherche intéressantes. Parmi les voies les plus prometteuse, nous pouvons citer l'utilisation des fonctions de fitness adaptatives et l'extension de l'échantillonnage adaptatif pour contrôler les 3 aspects de l'échantillonnage : la fréquence, la quantité et la stratégie.

CHAPITRE 5

GP sous Spark

Sommaire

5.1 Introduction 123
5.2 Environnement distribué Spark 124
5.2.1 MapReduce et Hadoop 124
5.2.2 Spark 128
5.2.3 YARN 129
5.2.4 EA et environnements distribués 130
5.3 DEAP sous Spark 132
5.3.1 Modèle d'implémentation 133
5.3.2 Échantillonnage avec Spark 135
5.4 Étude expérimentale 137
5.4.1 Environnement 137
5.4.2 Paramètres GP 138
5.5 Résultats et discussion 139
5.5.1 Temps d'apprentissage 139
5.5.2 Qualité de l'apprentissage 141
5.6 Conclusion 144

5.1 Introduction

À l'ère du Big Data et des *lacs de données*[1], la disponibilité de données nécessaires à la résolution d'un problème n'est plus un obstacle à l'obtention de

1. data lake : Une collection d'instances de stockage de diverses ressources de données en plus de leurs sources. (Source : Gartner)

résultats de classification de qualité. Un grand volume de données est un frein à l'application de la GP lorsqu'on dispose de ressources limitées en terme de mémoire et de puissance de calcul. Nous avons contourné ce problème, dans les chapitres précédents avec l'approche de réduction de la taille de l'ensemble d'apprentissage par différents algorithmes d'échantillonnage. Nous proposons dans ce chapitre une autre réponse à ce problème : une approche distribuée pour l'utilisation de la GP.

En effet, le Big Data a apporté de nouvelles approches et de nouveaux outils comme Hadoop MapReduce [2] et Apache Spark [3] qui implémentent un nouveau modèle de programmation au-dessus d'une architecture de stockage distribué. Ils sont les outils de facto pour toute application à haut volume de données.

Ce chapitre montre comment nous avons adapté une implémentation existante de GP au contexte de Spark afin de bénéficier de ses performances. Cette solution est appliquée au problème de classification des bosons de Higgs décrit dans le chapitre précédent. Nous étudions, particulièrement, l'effet de la variation de certains paramètres de GP sur le temps et la qualité d'apprentissage. De plus, nous ajoutons une étape d'échantillonnage pour examiner ces effets dans ce processus sous les mêmes conditions. Nous commençons par présenter un aperçu sur Spark et les travaux récents sur l'utilisation de GP dans cet environnement. Puis dans la section 5.3, nous exposons comment nous avons modifié une librairie GP pour être distribuée sous Spark ainsi que l'intégration de l'échantillonnage dans ce contexte. Les détails des expériences et les résultats obtenus sont discutés respectivement dans les sections 5.4 et 5.5.

5.2 Environnement distribué Spark

5.2.1 MapReduce et Hadoop

MapReduce est un modèle de programmation parallèle introduit par Dean et al. dans [Dean & Ghemawat 2004] pour Google. Il a été rendu populaire avec son implémentation Apache Hadoop. L'idée fondamentale qui a donné naissance à ce modèle est de déplacer les traitements vers les données en réduisant le trafic de données entre les différents nœuds.
MapReduce opère au-dessus d'un système de fichiers distribué. Hadoop offre son propre système HDFS (Hadoop Distributed File System) sachant que depuis sa version 2.0 d'autres systèmes de fichiers peuvent être utilisés comme GFS (Google File System). HDFS apporte la gestion du partitionnement et la réplication de données ainsi que l'ordonnancement des tâches et la communication entre les nœuds.

2. https://hadoop.apache.org
3. https://spark.apache.org

MapReduce applique la technique de « diviser pour régner » pour décomposer un traitement en plusieurs tâches exécutées en concurrence sur des machines différentes sur les portions de données qu'ils abritent. Ces tâches sont de deux types Map et Reduce dont l'origine est la programmation fonctionnelle.

Dans la phase **Map**, un traitement est exécuté en parallèle sur les partitions de données (Mappers) et produit un résultat intermédiaire sous la forme de couples (clé, valeur) où à chaque clé est associée une ou plusieurs valeurs. Les valeurs sont déterminées par la fonction du programme utilisateur passée au Map. Les résultats sont ensuite enregistrés sur le disque local de chaque nœud.

Tandis que, dans la phase **Reduce**, les résultats intermédiaires sont chargés et une fonction de réduction est exécutée pour agréger les valeurs se rapportant à la même clé. Chaque tâche de réduction (Reducer) prend en charge une ou plusieurs clés selon la cardinalité de l'ensemble des clés et la granularité de la phase de Reduce.

Il y a une phase intermédiaire qui est invisible appelée **Shuffle and Sort**. Elle consiste à envoyer à travers le réseau les résultats intermédiaires à chaque Reducer. Ils sont alors triés selon la clé et stockés en local. La figure 5.1 [4] montre ce principe appliqué pour compter le nombre de mots d'un texte.

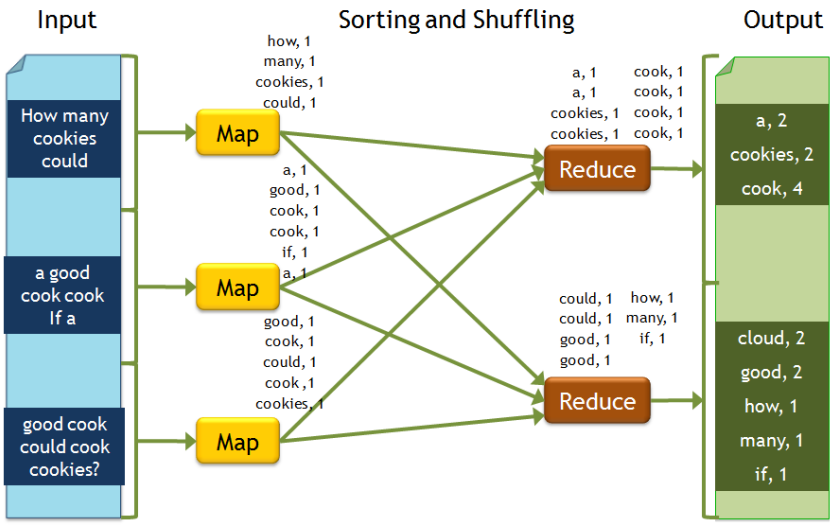


FIGURE 5.1 – Exemple de déroulement des opérations MapReduce pour faire le décompte de mots d'un fichier.

1. À gauche, le fichier source qui est partitionné sous HDFS.
2. **Map** construit les couples en utilisant chaque mot comme la clé à laquelle est associée la valeur 1.

4. Source : http://blog.enablecloud.com/2012/06/what-lies-at-core-of-hadoop.html

3. **Shuffle and Sort** regroupe les couples se rapportant à la même clé dans le même Reducer dans l'ordre des clés.
4. **Reduce** fusionne les couples ayant la même clé en un seul couple en additionnant les valeurs. Les résultats sont enregistrés sur HDFS.

Malgré la simplicité de ce modèle qui a permis la mise à l'échelle de plusieurs problèmes, il n'est pas adapté aux algorithmes itératifs comme la GP qui est pénalisée par le grand nombre d'E/S et le temps de latence réseau. En effet, MapReduce ne tire pas profit de la mémoire disponible et n'utilise pas de mise en cache des résultats intermédiaires, ce qui diminue sa performance globale pour les traitements répétitifs.

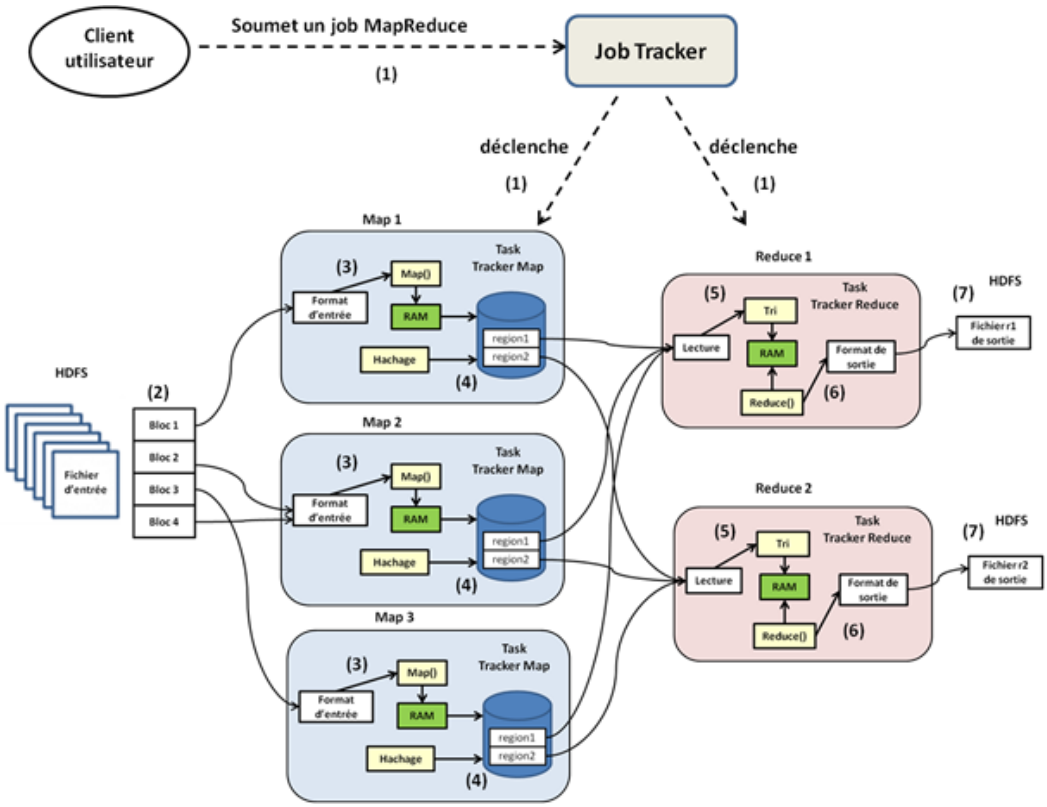


FIGURE 5.2 – Modèle d'exécution de MapReduce sur un cluster [Chokogue 2017].

La figure 5.2, décrit les étapes d'exécution d'un programme MapReduce sur un cluster à savoir :

1. L'utilisateur écrit son programme MapReduce, le configure et l'exécute. Le jobtracker déclenche les tasktrackers nécessaires pour les tâches Map et Reduce.
2. Le système de fichier HDFS se charge du partitionnement du fichier source en blocs et de leur réplication sur les nœuds.

3. Chaque Mapper applique la fonction Map sur les blocs locaux et les transforme en paires (clé, valeur) en mémoire.
4. Les paires obtenues sont sérialisées périodiquement dans un fichier sur le disque local du nœud qui sera partitionné à l'aide d'une fonction de hachage.
5. Chaque Reducer récupère depuis le disque des Mappers les régions qui leurs sont affectées. Puis les trie, selon la clé, en mémoire.
6. La fonction Reduce est appliquée sur les données triées. Les résultats sont sérialisés dans des fichiers séparés dans HDFS.
7. Fin du programme.

Dans le cas d'un programme itératif, l'étape 4 se traduit par plusieurs opérations d'E/S pour la sérialisation des résultats intermédiaires et leur chargement en mémoire. En outre, MapReduce exécute les tâches de manière séquentielle sans aucun mécanisme de cache pour les résultats intermédiaires pour la prochaine itération ni une procédure d'optimisation de l'exécution de l'ensemble des tâches. De plus, la phase Reduce doit attendre toutes les tâches Map pour ensuite effectuer des lectures sur les nœuds distants (étape 5). Ceci génère un trafic réseau important et par conséquent une latence importante.

MapReduce souffre également d'autres inconvénients comme l'absence d'interactivité, la complexité de la création de programmes MapReduce et son incapacité à traiter les flux continus de données (streaming). Pour remédier à ces lacunes, plusieurs solutions alternatives à MapReduce ont été développées. Le choix d'un framework Big Data dépend du cas d'utilisation à traiter. Plusieurs études comparatives [Veiga *et al.* 2016, Alkatheri *et al.* 2019] ont été réalisées (voir tableau 5.1).

TABLE 5.1 – Comparaison de frameworks Big Data [Veiga *et al.* 2016, Alkatheri *et al.* 2019].

Critère	**MapReduce**	**Spark**	**Flink**	**Storm**	**Samza**
Modèle de traitement	Batch	Batch, Micro-batch	Événement, Batch Micro-batch	Événement	Événement
Langages	Java, Scala	Java, Scala, Python, R	Java, Scala, Python	JVM	JVM
Latence	importante	faible	faible	très faible	faible
Date (Comité)	01/2008	02/2014	12/2014	09/2014	01/2015
Contributeurs	203	74	51	40	26
Programmation	Difficile	API riche	API limitée	API riche	API riche
Mise à l'échelle	Horizontale	Horizontale	Horizontale	Horizontale	Horizontale
Flux	Séquentiel	DAG [5]	CDG [6]	DAG	utilise Kafka
Rapidité	+	++++	+++	++	++

5. Directed Acyclic Graph : graphe orienté acyclique
6. Cyclic Dependecy Graph : Graphe de dépendance cyclique

Compte tenu du type de problème de notre étude, nous avons choisi le framework Spark pour sa rapidité avec les données en batch et la facilité d'écriture de programmes Spark grâce à ces primitives de haut niveau avec plusieurs langages (notamment Python). Spark est largement adopté et possède une grande communauté active.

5.2.2 Spark

Apache Spark a été développé à l'université de Berkeley pour contourner essentiellement les limitations de MapReduce en gardant ses atouts (scalabilité, tolérance aux pannes et localité de données). La pierre angulaire de Spark est son RDD : Resilient Distributed Datasets [Zaharia *et al.* 2012]. Un RDD est une structure de données parallèle typée, immuable et qui peut être mise en cache. Ces RDDs supportent 2 types d'opérations : les transformations qui produisent un RDD modifié et les actions générant des résultats qui ne sont pas de type RDD (un entier, un tableau, etc.) dont l'exécution nécessite la totalité des partitions de la RDD. Spark est jusqu'à 100 fois plus rapide que MapReduce avec les algorithmes itératifs grâce à :

1. Calculs in-memory (voir la figure 5.3) : à chaque itération, MapReduce effectue une lecture HDFS pour charger les entrées puis une écriture pour enregistrer les résultats. Spark effectue une seule lecture avant la première itération pour charger les données en mémoire. Les transformations se font en mémoire c'est uniquement le résultat final qui sera écrit sur HDFS.
2. Évaluations paresseuses ou tardives : l'exécution des transformations sur les RDD sont retardées jusqu'à rencontrer une action. Ceci permet à Spark d'optimiser leur exécution en se basant sur son ordonnanceur DAGScheduler.

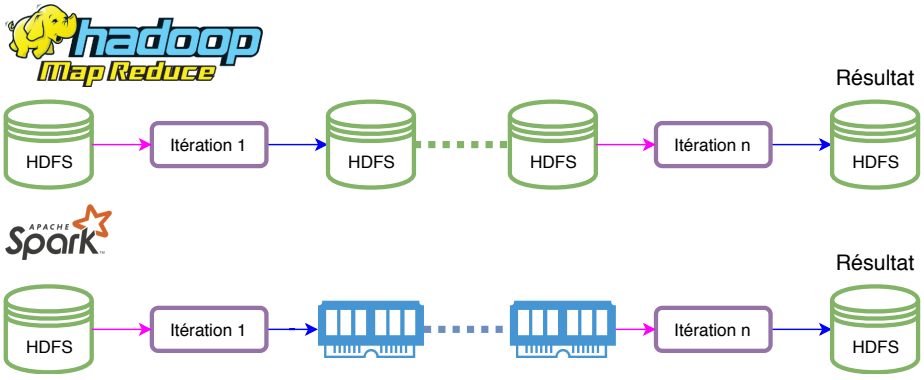


FIGURE 5.3 – MapReduce vs Spark.

Avec sa librairie **MLlib**, Spark offre une variété d'algorithmes d'apprentissage automatique de classification, régression, etc. Toutefois, à notre connaissance, aucun algorithme évolutionnaire n'est implémenté dans cette librairie.

5.2.3 YARN

Étant conçu pour un environnement distribué, Spark est typiquement déployé sur un cluster dont la taille peut varier de quelques machines à des milliers de machines gérées selon le principe maître-esclave (Driver-Worker)(figure 5.4). Pour orchestrer ces machines, Spark s'exécute au-dessus d'un gestionnaire de cluster. En plus du mode Standalone inclus avec Spark, il est compatible avec YARN, Mesos et Kubernetes) [Kienzler 2017].

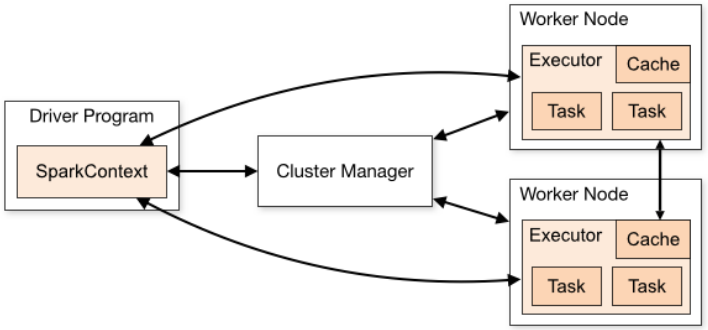


FIGURE 5.4 – Vue d'ensemble d'un cluster Spark.

YARN (Yet Another Resource Negotiator) est le gestionnaire de cluster de Hadoop et, par conséquent, le plus utilisé avec Spark. L'architecture de YARN est comme Spark selon le modèle maître-esclave représenté par les 2 composants : le gestionnaire de ressources (Resource manager) et le gestionnaire de nœud (Node manager). La figure 5.5 montre le scénario d'exécution d'une application Spark avec YARN.

1. Le client soumet une application Spark au gestionnaire de ressources.
2. Le gestionnaire de ressources demande à un gestionnaire de nœud de lancer un container[7] pour AM (Application Master).
3. Lancement du premier container pour AM qui contient Spark Driver.
4. Demande d'allocation de ressources pour l'application.
5. Demande de démarrage des containers pour chaque executor Spark.
6. Chaque gestionnaire de nœud démarre les containers.
7. Communication entre Driver Spark et executors.

7. Processus JVM avec les ressources CPU et mémoire alloués.

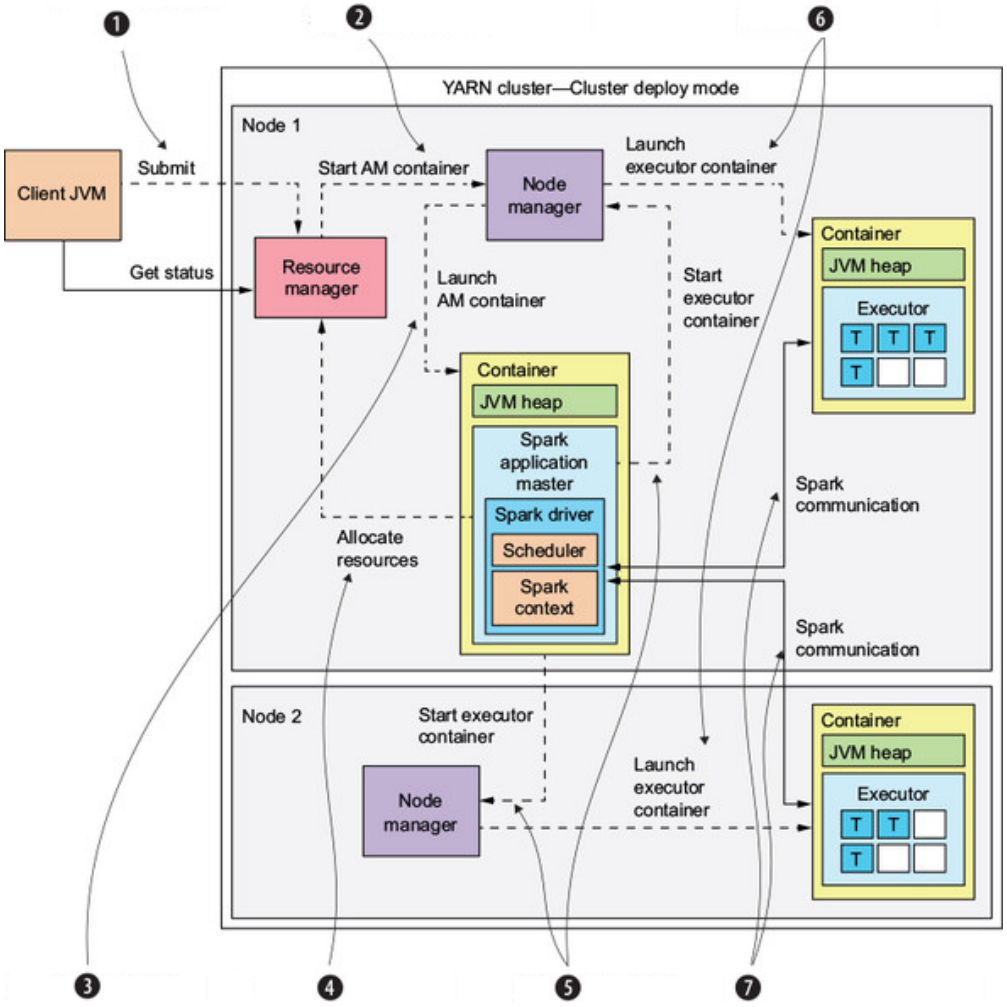


FIGURE 5.5 – Modèle d'exécution de Spark avec YARN [Zecevic & Bonaci 2016] (sur un cluster de 2 nœuds).

5.2.4 EA et environnements distribués

Les nouveaux environnements du paysage Big Data ont suscité l'intérêt des chercheurs et rapidement des applications ont vu le jour. Nous citons ci-dessous quelques travaux récents qui ont utilisé un algorithme évolutionnaire dans un environnement distribué.

Rong-Zhi Qi et al. [Qi *et al.* 2016] et Padaruru et al. [Paduraru *et al.* 2017] : ces travaux appliquent les algorithmes génétiques (GA) pour la génération automatique de jeux de tests de logiciels. Le premier a pour objectifs de couvrir les combinaisons possibles des valeurs de paramètres d'un programme. Les données en entrée sont le nombre de paramètres et les valeurs possibles pour chacun. Le second cherche les schémas d'exécution qui provoque un comportement inhabituel

(bugs par exemples). Les données d'entrée de ce problème sont des codes binaires. Les solutions sont une parallélisation de tout le processus évolutionnaire (évaluation et opérateurs génétiques) avec Spark. N'ayant pas de bases d'apprentissage, les RDDs sont utilisés pour paralléliser les populations et par conséquent les phases d'évaluation et d'application des opérateurs génétiques.

Chávez et al. [Chávez *et al.* 2016] : Il s'agit d'une modification apportée à la bibliothèque ECJ utilisée dans les chapitres 3 et 4 lui permettant d'utiliser MapReduce lors de l'évaluation des individus. Il exploite le mécanisme des points de sauvegarde fourni par ECJ et l'optimise avec la sérialisation des individus et la compression pour réduire le temps des E/S. Le nouvel outil ECJ+Hadoop est testé sur les algorithmes génétiques dans un problème de reconnaissance de visage. La taille des bases d'apprentissages utilisées est autour de 50MO. L'auteur s'est intéressé à l'impact de cette approche sur le temps d'exécution et a montré que MapReduce ne donne satisfaction que si la fonction de fitness est complexe ou la taille de la population est importante (>500000).

Peralta et al. [Peralta *et al.* 2015] : Avec cette solution, le modèle MapReduce est appliqué pour implémenter un GA pour la préparation de grandes bases d'apprentissage (jusqu'à 67 millions d'enregistrements et des attributs de 631 à 2000). C'est une application de sélection d'attributs où chaque mapper crée un vecteur d'attributs sur des sous-ensembles disjoints de la base originale. La phase Reduce agrège l'ensemble des vecteurs obtenus. La classification est faite par des algorithmes non évolutionnaires (classification naïve bayésienne et machines à vecteurs de support SVM).

Funika et Koperek [Funika & Koperek 2016] : Les auteurs ont implémenté un « Service d'évaluation » qui peut être interrogé à travers une API REST. Ce n'est pas une implémentation d'un EA particulier mais plutôt une externalisation de la phase d'évaluation. Ce service peut être utilisé par n'importe quel algorithme qui a besoin d'évaluer une expression sur un ensemble de données. Ils ont testé ce service sur 3 expressions différentes générées par un système à base de GA. La taille des données de test varie entre 1MO et 1024MO.

Al-Madi et Ludwig [Al-Madi & Ludwig 2013] : Dans ce travail, une implémentation complète de GP, basée sur MapReduce, est décrite. Elle consiste à distribuer la population selon le nombre de mappers et sa taille. La population étant enregistrée sur HDFS. Chaque mapper, calcule la fitness des individus qui lui sont parvenus sous forme du couple clé-valeur (identifiant individu, individu). La phase Map émet de nouveaux couple après mise à jour de la valeur de fitness sous la forme (numéro population, individu) permettant ainsi de regrouper les individus de la population dans la phase Reduce. Une fois les individus réunis, les opérations de sélection, croisement et mutation sont appliquées pour créer une

nouvelle population pour l'itération suivante. L'accent est mis sur l'utilisation de grandes populations (jusqu'à 50000 individus) tandis que les 6 bases d'apprentissage sont relatives à des problèmes de classification de petite taille (la plus grande compte 2126 enregistrement et 22 colonnes réduites à 7).

Notre proposition s'inspire de ces travaux, et principalement de ceux de Chávez et al. [Chávez *et al.* 2016] dans la démarche pour modifier un outil existant et de Peralta et al. [Peralta *et al.* 2015] pour la distribution de la base d'apprentissage. Mais nous utilisons Spark avec une très large base d'apprentissage qui est celle des bosons de Higgs. Notre objectif est de montrer l'avantage de la parallélisation de la phase d'évaluation dans la GP Standard. De plus, nous proposons d'intégrer l'échantillonnage aléatoire en amont de cette phase d'évaluation.

5.3 DEAP sous Spark

DEAP (Distributed Evolutionary Algorithms in Python) [Fortin *et al.* 2012] est présenté comme un framework de prototypage rapide et de test qui supporte la parallélisation et le mode multi-processeur. Il implémente plusieurs algorithmes évolutionnaire : GA, ES et GP.
Le choix de ce framework est justifié par les raisons suivantes :

- il est écrit en Python, l'un des 3 langages supporté par Spark (Scala, Java et Python),
- il implémente GP dans sa version standard,
- il est prédisposé pour le calcul parallèle.

Bien que DEAP soit structuré de manière à faciliter la distribution des calculs, il n'est pas compatible avec Spark de manière native. En effet, il n'utilise aucune de ses structures de données parallèles (RDDs, DataFrames ou DataSets).

L'adaptation de DEAP, comme le montre sa documentation[8], est simplifiée avec l'utilisation d'autres bibliothèques de calculs multiprocesseurs. Il suffit de remplacer la méthode *map* de la classe Toolbox par celle qui est parallélisée. Le listing dans la figure 5.6 permet l'utilisation de DEAP au-dessus de SCOOP (Scalable COncurrent Operations in Python) [Hold-Geoffroy *et al.* 2014]. SCOOP est l'une des bibliothèques de calcul distribué permettant la parallélisation sur plusieurs environnements.

```
1 from scoop import futures
2 toolbox.register("map", futures.map)
```

FIGURE 5.6 – DEAP avec SCOOP.

8. https://deap.readthedocs.io/en/master/tutorials/basic/part4.html

5.3.1 Modèle d'implémentation

Après l'examen des travaux décrits dans 5.2.4, deux principaux scénarios pour la parallélisation de tout algorithme évolutionnaire, et en particulier GP, peuvent être dressés :

1. Parallélisation de tout le processus évolutionnaire de la GP : chaque mapper, dans ce cas, est un run indépendant qui utilise les données locales. C'est assez proche du schéma de la co-évolution [Hillis 1990] ou le modèle en *îlots* [Alba & Tomassini 2002] dans la mesure où chaque mapper fait évoluer une population indépendante. À la fin de ce scénario, une agrégation est requise pour obtenir la solution finale. L'agrégation est exécutée par le driver, par exemple, une technique de vote entre les meilleurs individus de chaque population dans un problème de classification ou le choix du meilleur individu après un test sur la totalité de l'ensemble d'apprentissage.

2. Paralléliser la population : ce scénario correspond à faire évoluer une seule population. La parallélisation est réalisée par l'utilisation de structures parallèles et principalement les RDDs. Par conséquent, les individus sont distribués sur les différents nœuds et chacun traite les individus locaux. À cause de la nature immuables des RDDs, à la fin de chaque phase de GP (cf. 1.2.4.4), un nouveau RDD contenant les individus modifiés est créé pour être utilisé lors de la phase suivante.

Pour un problème de classification avec la GP ayant une large base d'apprentissage, il y a deux composantes qui génèrent un coût de calcul additionnel à savoir la GP représentée par la population d'individus et la base d'apprentissage.

L'objectif visé est de permettre l'utilisation de GP sur ce type de problèmes en exploitant des clusters de petite taille. Pour cela, nous limitons la distribution à la phase d'évaluation. Le reste des opérations évolutionnaires est exécuté par le driver. Et pour réduire le besoin en ressources, nous faisons évoluer une seule population.

La première solution à investiguer est de distribuer simultanément la base d'apprentissage et la population. Nous commençons alors par réécrire la phase d'évaluation d'un individu en considérant une fonction de fitness pour maximiser le taux de bonnes prédictions (ou minimiser les erreurs de classification) (figure 5.7 lignes 6–10). Elle permet de calculer la fitness d'un individu en évaluant sa performance sur la base d'apprentissage parallélisée avec *TrainingRDD* (voir la figure 5.7 ligne 4).

Un individu est représenté par une fonction *func*. Chaque mapper, associe 1 à chaque enregistrement de la partition s'il est correctement classé et 0 sinon. Pour obtenir la fitness finale, un Reduce permettant de compter les bonnes prédictions est effectué.

```
from pyspark import SparkContext
sc = SparkContext(appName="DEAPSPARK")
# création de la RDD de la base d'apprentissage
TrainingRDD = sc.textFile("training.csv").cache()
# fonction d'évaluation d'un individu
def evalSpark(individual):
  func = toolbox.compile(expr=individual)
  correctPredictionsRDD =\
    TrainingRDD.map(lambda line:isCorrectPrediction(func,line))
  fitness = correctPreditionsRDD.reduce(lambda v1,v2:v1+v2)
  return fitness,
# enregistrement de la fonction evalSpark avec DEAP comme fonction
# d'évaluation
toolbox.register("evaluate", evalSpark)
```

FIGURE 5.7 – Redéfinition de la fonction *evaluate* avec une base d'apprentissage parallélisée.

Ensuite, nous distribuons la population en utilisant une RDD qui sera utilisée selon le modèle des bibliothèques multiprocesseur comme SCOOP. Ceci est réalisé avec le code dans la figure 5.8.

```
# fonction d'évaluation de la population
def sparkMap(evaluation, population):
  return sc.parallelize(population).map(evaluation)
# enregistrement de sparkMap avec DEAP
toolbox.register("map", sparkMap)
```

FIGURE 5.8 – Réécriture de *map* pour DEAP avec une population parallélisée.

Cette solution est confrontée à une contrainte de Spark qui interdit l'imbrication de RDDs. En effet, pour évaluer un individu de la population parallélisée, nous avons besoin d'accéder à la base d'apprentissage dans une autre RDD. Cette solution est ipso facto infaisable sous Spark.

En se limitant à la parallélisation avec la réécriture de l'évaluation tel que préconisée dans la figure 5.7, le nombre d'opérations Reduce est égal à la taille de la population. Une règle importante pour l'optimisation sous Spark est de diminuer le nombre d'actions par rapport aux transformations dont l'évaluation est tardive et plus optimisée par Spark.

Nous suggérons, dans la figure 5.9, la modification de DEAP pour diminuer le nombre de Reduce à une opération pour toute la population. Il s'agit d'évaluer

tous les individus par chaque mapper sur la partition locale.

À la fin de ce processus, *updateFitness* récupère un vecteur des valeurs de fitness et les affecte aux individus de la population. C'est seulement les programmes de chaque expression d'un classifieur qui sont sérialisées pour être transférés aux différents nœuds. Pour modifier DEAP, c'est la boucle principale de GP qui a été modifiée.

```
1 def evalPopulation(population):
2     popFunctions = [toolbox.compile(ind) for ind in population]
3     fitnessRDD = TrainingRDD.map(lambda line:\
4         [isCorrectPrediction(func,line) for func in popFunctions])
5     fitnessValues =
6         fitnessRDD.reduce(lambda v1,v2:list(map(operator.add,v1,v2)))
7     updateFitness(population, fitnessValues)
```

FIGURE 5.9 – Évaluation parallèle de population avec DEAP sous Spark.

La figure 5.10 donne le diagramme global de l'algorithme d'évaluation modifié.

D'abord, la base d'apprentissage est transformé en RDD (*TrainingRDD*) à partir d'un fichier enregistré sur HDFS et mis en cache (ligne 4 de la figure 5.7).

Ensuite, à chaque génération de GP, les expressions de chaque individus de la population sont sérialisées et transférées vers les mapper pour évaluer la population sur *TrainingRDD* qui est partitionné sur les nœuds du cluster (lignes 2–4 de la figure 5.9).

Enfin, les fitness partielles sont agrégées à l'aide de l'opération Reduce pour construire la liste de valeurs de fitness de la population (lignes 5 et 6 de la figure 5.9).

Après, une nouvelle population sera créée par application de sélection, mutation et croisement au niveau du Driver. La boucle reprend de nouveau avec la nouvelle population tant que le nombre maximum de générations n'est pas atteint.

5.3.2 Échantillonnage avec Spark

L'apprentissage à partir de base données massives comme celle de Higgs accentue le problème de temps de calcul de GP. Pour les clusters de petite taille, exécuter GP pour résoudre de tels problèmes est toujours d'un coût élevé. Par ailleurs, la redondance dans ces bases de données est inévitable. L'échantillonnage est une technique qui offre un moyen pour contourner ces difficultés. Nous avons montré dans les chapitres 3 et 4 que, selon l'algorithme appliqué, l'échantillonnage atténue le sur-apprentissage, améliore la qualité de l'apprentissage et permet d'accroître la taille des populations pour une meilleure diversité et/ou de prolonger l'apprentissage avec un nombre de générations plus important.

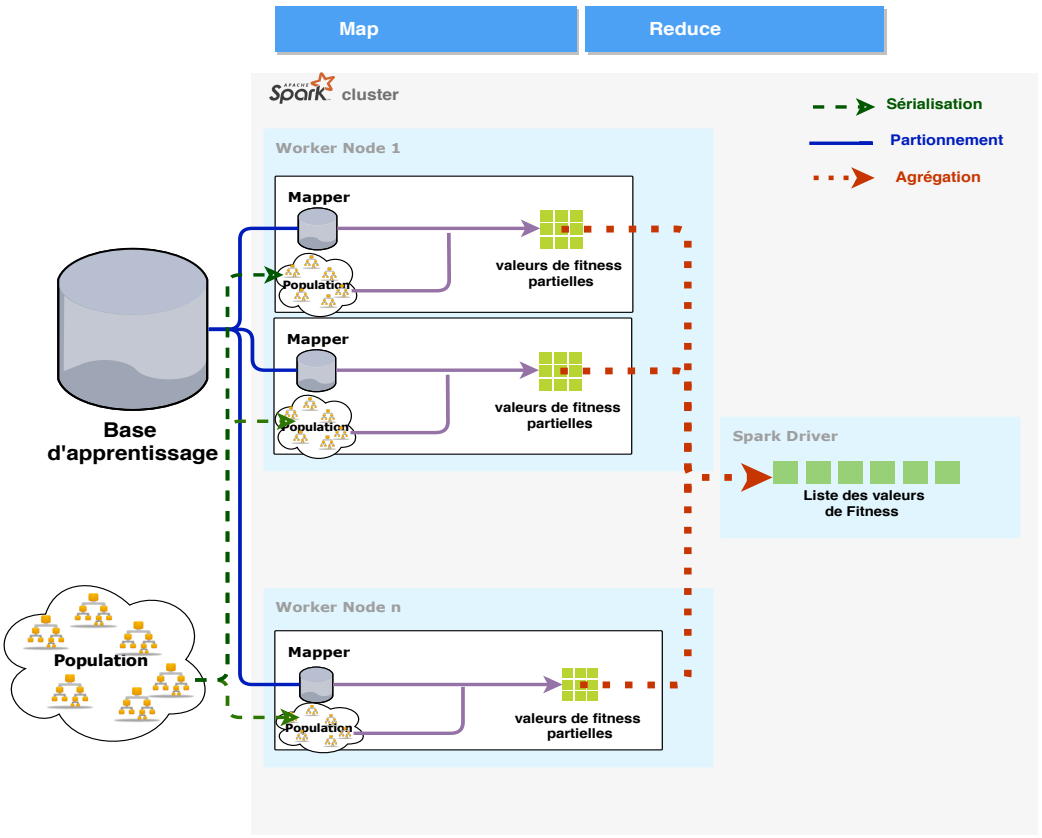


FIGURE 5.10 – Diagramme global de l'évaluation modifiée.

Pour ces raisons, nous avons étudié l'effet de l'échantillonnage dans le modèle décrit dans le paragraphe précédent. Nous avons choisi de commencer par un échantillonnage simple et efficace qui est RSS [Gathercole & Ross 1994].

Les RDDs disposent de deux méthodes *sample* et *takeSample*. Elles sélectionnent des instances des données de manière aléatoire en respectant une proportion donnée et sans remise. La taille de l'échantillon oscille autour de la proportion demandée. C'est le principe de l'algorithme RSS.

Afin d'exploiter le traitement in-memory de Spark, nous avons utilisé *sample* qui est une transformation et donc retourne l'échantillon sous forme de RDD. *takeSample* est une action et ne peut être optimisée par l'ordonnanceur de Spark. L'échantillon est renouvelé par génération avant l'évaluation de la population. La figure 5.11 illustre cette intégration de l'échantillonnage.

Spark fournit une autre méthode qui peut être utilisée pour l'échantillonnage *filter*, elle se base sur une fonction pour décider de sélectionner un enregistrement ou non. Néanmoins, les algorithmes d'échantillonnage qui ont besoin de données supplémentaires (comme DSS qui calcule l'âge et la difficulté de chaque instance) sont plus difficiles à implémenter sous Spark.

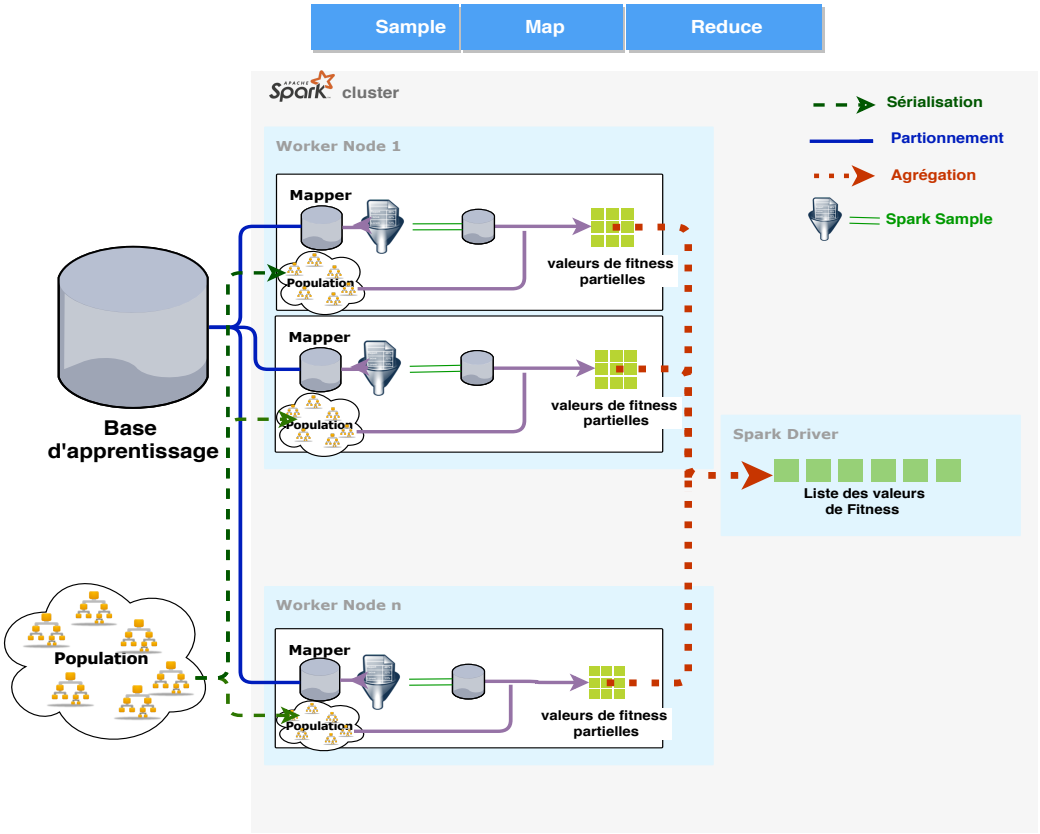


FIGURE 5.11 – Échantillonnage avec Spark.

5.4 Étude expérimentale

Dans ce qui suit, nous exposons le détail des ressources matérielles et logicielles exploitées pour les différentes expérimentations réalisées sous Spark. Nous donnons également les valeurs des paramètres de GP.

5.4.1 Environnement

Environnement logiciel Le détail de la configuration logicielle est comme suit :

- Version Spark : 2.1.0
- Version Hadoop : 2.9.1
- Gestionnaire de ressources : YARN
- Système d'exploitation : SMP Debian 4.9.130
- Version DEAP : 1.2.2

Cluster Spark Nous avons utilisé un cluster de petite taille composé de 4 nœuds. Ces nœuds ont une configuration matérielle identique donnée dans le tableau 5.2.

TABLE 5.2 – Configuration matérielle d'un nœud du cluster.

CPU	Processeur Intel Core (Haswell)
Cores	16
Fréquence CPU	2.397 GHZ
RAM	45GO
Stockage (HDFS)	1TO

La soumission de l'application Spark à travers le script *spark-submit* est faite sous les mêmes directives pour toutes les expériences afin d'annuler l'effet de la variation de ces paramètres sur les résultats. Ces valeurs ont été ajustées dans l'optique d'exploiter de façon optimale les ressources disponibles en appliquant plusieurs recommandations comme celles dans [Karau & Warren 2017].

5.4.2 Paramètres GP

Les paramètres de GP sont listés dans le tableau 5.3. Ils sont définis en accord avec les expériences réalisées lors des chapitres précédents et n'ont pas été l'objet d'une optimisation poussée.

TABLE 5.3 – Paramètres GP.

Ensemble de fonctions	
Opérateurs arithmétiques	$+, -, *, /$
Opérateurs de comparaison	$<, >, =$
Opérateurs logiques	AND, OR, NOT
Autres	IF (IF THEN ELSE)
Ensemble de terminaux	
Attributs de la base Higgs	28
Constantes aléatoires	1 in $[0, 100[$
Constantes booléennes	True, False
Opérations génétiques	
Initialisation	Ramped half and half
Taille de tournoi	4
Taille limite de l'arbre	17
Probabilité de croisement	0.9
Probabilité de mutation	0.04

5.5 Résultats et discussion

L'objectif de la modification de DEAP est de contrecarrer le problème de temps de calcul de GP dans les problèmes d'apprentissage supervisé. Pour cette raison, la première mesure à relever est le temps d'apprentissage. C'est le temps écoulé entre l'initialisation de la première population jusqu'à la dernière génération sans pour autant inclure le temps d'évaluation sur la base de test.

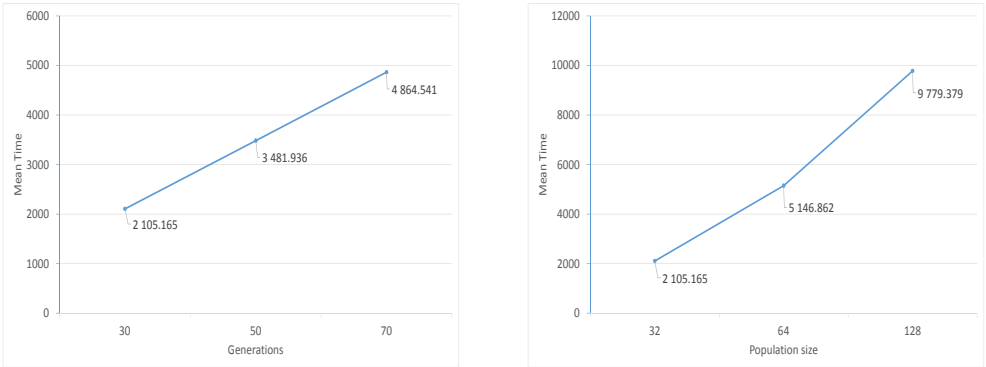
Nonobstant, la réduction du temps ne doit pas se faire au détriment de la qualité d'apprentissage. Pour cela, à la fin de chaque run, le meilleur individu par rapport à sa fitness est évalué sur la base de test de 500000 instances. Les résultats sont sauvegardés sous forme d'une matrice de confusion de laquelle les mesures de TPR (précision) et FPR (rappel) sont dérivées.

Nous avons expérimenté 61 combinaisons différentes obtenues par la variation de 3 paramètres : la taille de la population, le nombre de générations et la taille de l'échantillon. Chaque configuration est testée durant 11 runs distincts. Ensuite, nous calculons le temps d'apprentissage moyen ainsi que les meilleures performances du meilleur individu de ces runs.

Ces expériences ont pour objectif de détecter le gain en temps pour GP et l'effet du changement de ces paramètres sur les mesures considérées.

5.5.1 Temps d'apprentissage

La figure 5.12 montre le temps d'apprentissage moyen des runs avec 3 nombres de générations (30, 50 et 70) pour une population de 32 individus. Puis les temps sont donnés avec 3 tailles de populations (32, 64 et 128) et un nombre de générations égal à 30 avec la totalité de la base d'apprentissage (FSS).



(a) Population = 32 (b) Générations = 30

FIGURE 5.12 – Temps d'apprentissage moyen à partir de la base Higgs sous Spark.

Il est clair que la parallélisation de GP sur Spark, facilite son utilisation pour résoudre les problèmes de classification à données massives comme celui des

boson de Higgs. En effet, sous Spark, avec seulement 4 nœuds, un run GP dure en moyenne 4864.541 secondes dans 70 générations et 32 individus et 9779.379 secondes dans 30 générations et 128 individus (figure 5.12). Une exécution non parallélisée de GP sur un nœud prend plus de 20 secondes pour évaluer un seul individu sur la totalité de la base d'apprentissage et, par interpolation, pourrait dépasser les 44800 secondes pour itérer 70 générations de 32 individus (voir tableau 5.4).

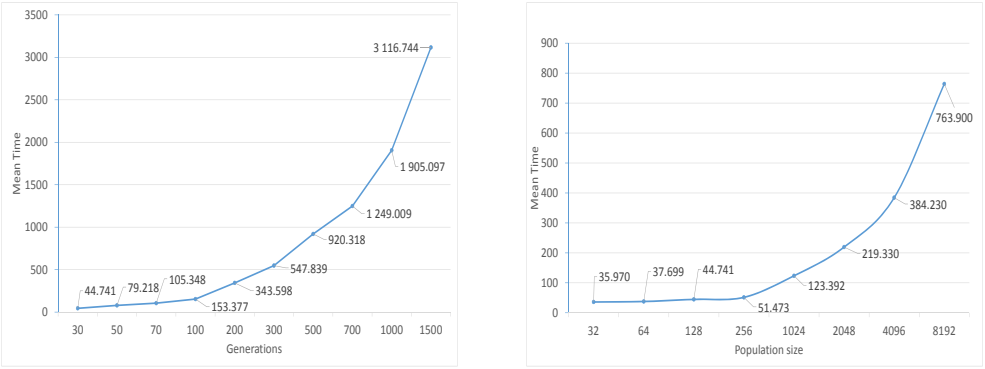
TABLE 5.4 – Temps d'exécution avec Spark en secondes.

	70 générations et 30 individus	**30 générations et 128 individus**
Exécution parallélisée	4864.541	9779.379
Exécution non parallélisée	44800	76800

La parallélisation de l'évaluation a réalisé une accélération avec un facteur supérieur à neuf fois. Ce résultat peut être rehaussé en déployant plus de nœuds sur le cluster. Par ailleurs, le temps d'apprentissage croît de manière linéaire par rapport à la taille de la population et aussi par rapport au nombre de générations sous la même configuration du cluster Spark.

Puis, pour pouvoir évoluer des populations plus grande et réaliser plus de générations sans ajouter de nœuds au cluster, nous avons intégré l'algorithme RSS dans la GP modifiée. En utilisant les mêmes paramètres mais une plage plus grande pour la taille de la population allant de 32 jusqu'à 8192 et le nombre de génération qui varie entre 30 et 1500, nous avons obtenu les résultats exposés dans les figures 5.13 et 5.14. La taille cible de l'échantillon est 10000 instances.

Le temps d'apprentissage moyen pour 128 individus pendant 1500 générations est 3116.744 secondes. Pour 30 générations et 8192 individus ce temps devient 763.9 secondes.



(a) Population = 128 (b) Générations = 30

FIGURE 5.13 – Temps moyen d'apprentissage pour RSS (10000 instances).

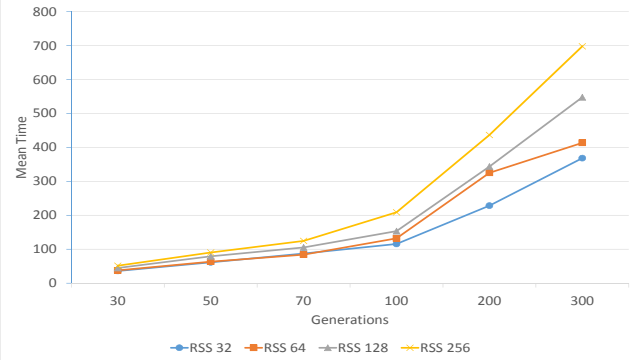


FIGURE 5.14 – Temps moyen d'apprentissage pour RSS (10000 instances) avec variation de la taille de la population et du nombre de générations.

Les deux courbes (figure 5.13) ont la même allure. C'est dû à l'augmentation du nombre d'évaluations à faire par la GP (en augmentant la taille de la population ou en augmentant le nombre de générations). Aussi, pour les valeurs basses des paramètres taille de la population et nombre de générations, la courbe du temps d'apprentissage a une légère pente quasi-plate. Le temps, dans ce cas, est affecté par les délais d'ordonnancement dans Spark. Ces observations sont consolidés avec les résultats de la figure 5.14.

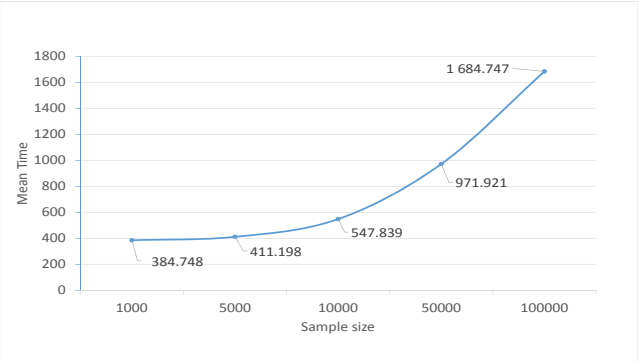
Afin d'étudier l'effet de la taille de l'échantillon (figure 5.15), nous avons testé les valeurs suivantes : 1000, 5000, 10000, 50000 et 100000 instances ce qui représente une proportion allant de 10^{-5} à 10^{-3} de la base d'apprentissage. Pour une population fixée à 128 individus et un nombre de générations à 300, la variation de la taille de l'échantillon a le même effet que le nombre de générations et la taille de la population lors de l'utilisation de faibles valeurs.

Enfin, pour comparer les résultats de GP parallélisée avec ou sans échantillonnage RSS, la figure 5.16 juxtapose leurs temps d'apprentissage moyens. Les graphiques confirment encore la même conclusion que l'étude comparative dans le chapitre 3 quant au gain du temps d'apprentissage suite à l'intégration de l'échantillonnage RSS avec GP.

5.5.2 Qualité de l'apprentissage

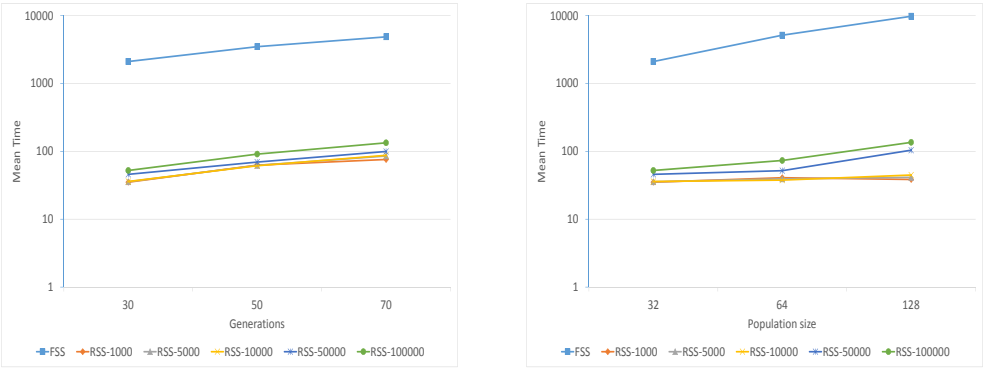
Le premier objectif par rapport à la qualité d'apprentissage est de réaliser des résultats comparables à ceux obtenus par d'autres techniques. Le meilleur résultat publié pour ce problème avec les mêmes conditions est de 60.76% réalisé par régression logistique [Shashidhara *et al.* 2015]. Le second objectif est de garder les mêmes performances d'apprentissage qu'avant l'introduction de l'échantillonnage.

Tout d'abord, nous mesurons la qualité de GP distribuée an utilisant entière-



Population = 128, Générations = 300

FIGURE 5.15 – Effet de la taille de l'échantillon RSS sur le temps.



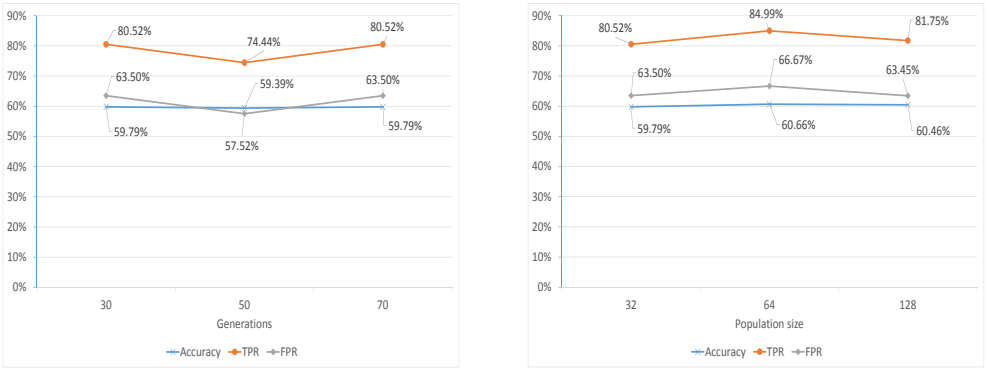
(a) Population = 32 (b) Générations = 30

FIGURE 5.16 – Temps moyen d'apprentissage RSS contre FSS.

ment la base d'apprentissage (figure 5.17).

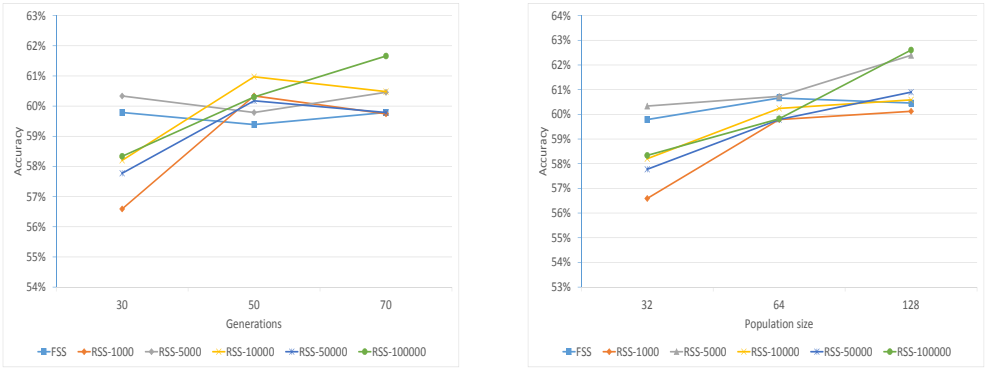
Le meilleur individu parmi ces configurations est obtenu avec une population de 64 individus et 30 générations. Les indicateurs de qualité pour cet individu sont : Accuracy =60.66% , TPR = 84.99% et FPR = 66.67%. C'est légèrement au-dessous de celui obtenu par Shashidhara et al. [Shashidhara *et al.* 2015].

Au vu des résultats de la figure 5.16, GP combinée avec RSS donne de meilleurs temps pour les 5 tailles d'échantillons testés (à savoir 1000, 5000, 10000, 50000 et 100000). Quant à la qualité d'apprentissage, du point de vue taux de détection (accuracy), les expériences utilisant RSS sont moins efficace lorsque la taille de la population ou le nombre de générations sont faibles. L'échantillonnage avec RSS donne de meilleurs résultats dès qu'on utilise des populations ou des générations plus nombreuses. D'une part, dans la figure 5.18(a), c'est à partir de 50 générations que toutes les variantes RSS (notées RSS-Taille échantillon) obtiennent de meilleures



(a) Population = 32 (b) Générations = 30

FIGURE 5.17 – Performance du meilleur individu obtenu par FSS.



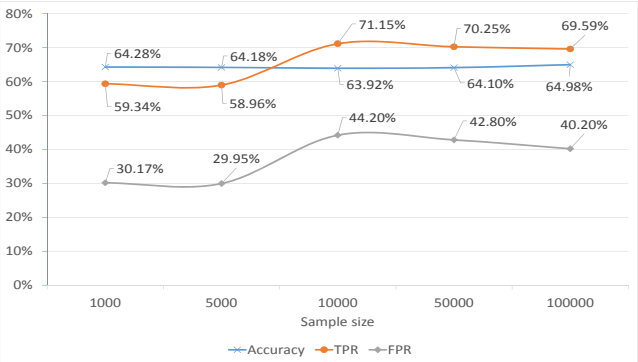
(a) Population = 32 (b) Générations = 30

FIGURE 5.18 – Taux de détection de RSS contre FSS.

valeurs pour la mesure accuracy. D'autre part, la figure 5.18(b), montre qu'avec une population de taille 128, RSS devient au-dessus de GP sans échantillonnage (FSS).

Enfin, l'impact de la variation de la taille de l'échantillon obtenu par RSS sur les 3 mesures de la qualité d'apprentissage est illustré dans la figure 5.19. Bien que le taux détection s'améliore avec l'augmentation de la taille de l'échantillon, cette augmentation demeure assez légère. En plus, les courbes représentant TPR et FPR empruntent la même forme. Ces deux indicateurs sont affectés de manière identique par le changement de la taille de l'échantillon.

Il est important de mentionner qu'aucune technique d'amélioration de la qualité de l'apprentissage tel que l'ingénierie des attributs n'a été déployée dans ces expériences (hormis la normalisation). Toutefois, la meilleure performance selon le taux de détection des bosons est de 66.93% avec RSS (population : 128, générations :



Population = 128, Générations = 300

FIGURE 5.19 – Effet de la taille de l'échantillon RSS sur la qualité de l'apprentissage.

1500, échantillon : 10000) en 3190.02 secondes c'est à dire en moins d'une heure. Ce résultat est meilleur que celui de Shashidhara et al. 60.76% [Shashidhara *et al.* 2015]. Il est aussi meilleur que les résultats obtenus dans le chapitre 4 par RSS (63.79%) et RSS-DSS (65.04%).

5.6 Conclusion

Nous avons présenté notre approche pour remodeler la librairie DEAP en parallélisant l'évaluation de la population sur un cluster Spark. Nous avons obtenu des résultats encourageants qui montrent que Spark est un environnement efficient et convenable à la distribution des itérations de GP.

Puis, nous avons intégré une technique simple d'échantillonnage qui permet simultanément de :

- préserver la qualité de l'apprentissage,
- appliquer GP à un problème de classification à base d'apprentissage de grande taille,
- résoudre ce problème sur un cluster de petite taille,
- réaliser des apprentissages avec de grandes population et un nombre de générations élevé.

À travers l'étude expérimentale, nous avons étudié l'effet de la variation des 3 paramètres : taille de la population, nombre de générations et taille de l'échantillon. Nous avons constaté un impact identique sur la TPR et le FPR ainsi qu'une augmentation logique du temps avec le nombre d'évaluation. L'exécution distribuée de GP réalise un important gain de temps d'apprentissage par rapport à une solution non distribuée. Enfin, l'intégration de RSS à cette solution distribuée a permis de réaliser de meilleurs résultats que ceux obtenus dans le chapitre 4 et ceux de Shashidhara et al. [Shashidhara *et al.* 2015].

Une extension logique de cette étude est d'investiguer l'impact de la configuration de Spark (nombre de nœuds, partitionnement RDD, taille des partitions, etc.). Une telle étude permettrait de trouver la meilleure configuration de Spark pour GP.

Une seconde voie consiste à vérifier la faisabilité de l'application d'autres techniques d'échantillonnage, en particulier celles appartenant à l'échantillonnage actif qui ont prouvé leur contribution avantageuse pour l'apprentissage automatique. L'échantillonnage hiérarchique que nous avons utilisé dans [Hmida *et al.* 2018], est un candidat prometteur pour cette voie.

Conclusion générale

Sommaire

Bilan **147**
Perspectives **148**

Bilan

Le coût de calcul assez élevé qui caractérise les GP met un frein à l'adoption de ces solveurs universels dont le potentiel est pourtant largement reconnu. La génération de larges volumes de données avec le Big Data, bien que favorable à une extraction de connaissances de qualité, accentue cette faiblesse des GP et rend impossible son application pour résoudre des problèmes d'apprentissage supervisé. L'objectif de cette thèse était d'apporter des solutions à ce problème.

Pour arriver à cette fin, nous nous sommes orientés, dans un premier temps, vers la réduction de la base d'apprentissage par le biais d'un algorithme d'échantillonnage. La revue étendue des algorithmes d'échantillonnage conçu à des fins différentes a abouti à la mise en place d'une taxonomie de ces techniques. L'étude comparative expérimentale sur le problème KDD'99 a montré leur apport dans le processus d'apprentissage en produisant des classifieurs avec une meilleure généralisation que la GP standard. Mais certaines méthodes se sont avérées inadaptées pour un grand volume de données. En effet, IDI et TBS maintiennent le problème de temps de calcul de la GP. Par ailleurs, la qualité des classifieurs obtenus avec RSS-DSS est tributaire de la fonction de fitness utilisée.

Ensuite, nous avons conçu et implémenté deux extensions à base d'échantillonnage multi-niveaux (hiérarchique). La première vise à adapter les méthodes pour lesquelles le problème de temps de calcul persiste. La validation est effectuée par la proposition de deux méthodes hiérarchiques RSS-TBS et BUSS-RSS-TBS qui ont pu être appliquées avec des temps raisonnables sur le problème KDD'99 sans déployer des ressources matérielles supplémentaires. La seconde extension consiste à permettre l'application de CGP pour résoudre un problème encore plus large, celui de la détection des bosons de Higgs, impossible avec la version standard. Cette extension a réalisé des performances comparables à des travaux

qui n'utilisent pas GP.

De plus, nous avons proposé le schéma d'une nouvelle classe d'algorithmes d'échantillonnage appelé l'échantillonnage adaptatif. Ce schéma s'applique à n'importe quelle stratégie de sélection d'un algorithme de type 1-niveau en transformant la génération d'échantillon d'une exécution systématique à une exécution contrôlée par un prédicat plus au moins complexe. De plus, nous avons défini trois schémas de contrôle : fixe, déterministe et adaptatif. L'implémentation de ce modèle, avec ses trois schémas, a été faite avec les méthodes RSS, DSS, BUSS et BRSS pour contrôler la fréquence de ré-échantillonnage. Ce principe a permis de réduire le temps d'apprentissage pour l'échantillonnage dont la phase de sélection des exemples d'apprentissage est coûteuse. Toutefois, son application reste sans effet notable positif ou négatif sur la qualité de l'apprentissage.

Finalement, nous avons contribué à l'utilisation de la GP standard avec un problème Big Data en exploitant les nouveaux outils de ce domaine. En effet, nous avons modifié la GP implémentée dans la librairie DEAP en parallélisant l'évaluation de la population sur un cluster Spark. Les résultats obtenus en sollicitant 4 nœuds confirme l'adéquation de Spark pour les traitements itératifs. Par ailleurs, en utilisant les fonctions de Spark, nous avons montré l'utilité de l'échantillonnage dans cet environnement pour les petits clusters.

Perspectives

Les travaux réalisés dans cette thèse ouvrent plusieurs directions de recherches pour approfondir certains travaux entamés ici ou pour explorer de nouvelles perspectives. Nous les décrivons dans les points qui suivent.

- Pour l'ensemble des contributions de cette thèse, l'optimisation des paramètres de GP n'a pas été examinée de façon approfondie. Comme toutes les techniques d'apprentissage automatique, l'ajustement des hyper-paramètres possède une forte influence sur la qualité des résultats obtenus. En outre, l'introduction de l'échantillonnage apporte ses propres paramètres dont le plus commun est la taille cible. Il est important de déterminer une valeur optimale pour ce paramètre ou encore un intervalle dans lequel l'apprentissage est plus performant.
- L'échantillonnage adaptatif a été introduit pour se positionner à mi-chemin entre l'échantillonnage statique et dynamique. La mise en place d'autres déclencheurs (ou prédicats) pour le ré-échantillonnage constitue une voie à développer et peut avoir des retombées positives principalement sur la qualité des classifieurs.
- La parallélisation des évaluations de GP sur un cluster Spark est un premier pas réalisé dans cette thèse à travers l'adaptation de DEAP. L'expérimentation a été effectuée sur un cluster de petite taille ce qui n'a pas permis d'essayer des configurations différentes de Spark avec YARN. L'impact du changement

des paramètres comme le nombre de nœuds, le partitionnement RDD, la taille des partitions, etc. nécessite une étude complémentaire qui pourrait donner la meilleure configuration Spark en fonction de la taille de la base d'apprentissage et la taille de la population GP.

– L'implémentation d'autres techniques d'échantillonnage sur les RDD, notamment l'échantillonnage hiérarchique, constitue une suite logique à ces travaux. Ceci peut être réalisé à travers la décomposition de la base d'apprentissage en plusieurs RDD.

– L'étude des différents modèles de parallélisation des algorithmes évolutionnaires [Alba & Tomassini 2002] pour les actualiser au vu des nouveaux outils et frameworks Big Data.

Bibliographie

[Adam-Bourdarios *et al.* 2014a] Claire Adam-Bourdarios, Glen Cowan, Cecile Germain, Isabelle Guyon, Balázs Kégl et David Rousseau. *Learning to discover : the higgs boson machine learning challenge*, 2014. `http://higgsml.lal.in2p3.fr/documentation`. En ligne : visité le 16 mai 2019. (Cité en page 105.)

[Adam-Bourdarios *et al.* 2014b] Claire Adam-Bourdarios, Glen Cowan, Cecile Germain, Isabelle Guyon, Balázs Kégl et David Rousseau. *The ATLAS Higgs Boson Machine Learning Challenge*. In International Conference on High Energy Physics(ICHEP) Conference, Nuclear Physics B - Proceedings Supplements, Valencia, Spain, Juillet 2014. (Cité en pages 106 et 108.)

[Adam-Bourdarios *et al.* 2016] Claire Adam-Bourdarios, Glen Cowan, Cécile Germain, Isabelle Guyon, Balázs Kégl et David Rousseau. *How Machine Learning won the Higgs Boson Challenge*. In European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2016. (Cité en page 107.)

[Ain *et al.* 2017] Qurrat Ul Ain, Bing Xue, Harith Al-Sahaf et Mengjie Zhang. *Genetic programming for skin cancer detection in dermoscopic images*. In Jose A. Lozano, editeur, 2017 IEEE Congress on Evolutionary Computation (CEC), pages 2420–2427, Donostia, San Sebastian, Spain, 5-8 Juin 2017. IEEE. (Cité en page 2.)

[Akaho 1992] Shotaro Akaho. *Regularization Learning of Neural Networks for Generalization*. In Shuji Doshita, Koichi Furukawa, Klaus P. Jantke et Toyoaki Nishida, editeurs, Algorithmic Learning Theory, Third Workshop, ALT '92, Tokyo, Japan, October 20-22, 1992, Proceedings, volume 743 of *Lecture Notes in Computer Science*, pages 99–110. Springer, 1992. (Cité en page 54.)

[Al-Madi & Ludwig 2013] Nailah Al-Madi et Simone A. Ludwig. *Scaling Genetic Programming for data classification using MapReduce methodology*. In Fifth World Congress on Nature and Biologically Inspired Computing, NaBIC 2013, Fargo, ND, USA, August 12-14, 2013, pages 132–139. IEEE, 2013. (Cité en page 131.)

[Alba & Tomassini 2002] Enrique Alba et Marco Tomassini. *Parallelism and evolutionary algorithms*. IEEE Trans. Evolutionary Computation, vol. 6, no. 5, pages 443–462, 2002. (Cité en pages 133 et 149.)

[Alkatheri *et al.* 2019] Safaa Alkatheri, Samah Abbas et Muazzam Siddiqui. *A Comparative Study of Big Data Frameworks*. International Journal of Computer Science and Information Security, vol. 17, no. 1, pages 66–73, 01 2019. (Cité en page 127.)

[Angeline 1996] Peter J. Angeline. *Two Self-Adaptive Crossover Operators for Genetic Programming*. In Peter J. Angeline et K. E. Kinnear, Jr., editeurs, Advances in Genetic Programming 2, chapitre 5, pages 89–110. MIT Press, Cambridge, MA, USA, 1996. (Cité en pages 29 et 30.)

[ATLAS collaboration 2014] ATLAS collaboration. *Dataset from the ATLAS Higgs Boson Machine Learning Challenge 2014*. `http://opendata.cern.ch/record/328`, 2014. En ligne : visité le 16 mai 2019. (Cité en page 105.)

[Atlas *et al.* 1990] Les E. Atlas, David Cohn et Richard Ladner. *Training Connectionist Networks with Queries and Selective Sampling*. In Advances in Neural Information Processing Systems 2, pages 566–573. Morgan-Kaufmann, 1990. (Cité en page 52.)

[Bacardit & Llorà 2013] Jaume Bacardit et Xavier Llorà. *Large-scale data mining using genetics-based machine learning*. Wiley Interdiscip. Rev. Data Min. Knowl. Discov., vol. 3, no. 1, pages 37–61, 2013. (Cité en pages 60, 62 et 63.)

[Back *et al.* 1997] Thomas Back, David B. Fogel et Zbigniew Michalewicz, editeurs. Handbook of evolutionary computation. IOP Publishing Ltd., Bristol, UK, UK, 1st édition, 1997. (Cité en page 17.)

[Baker 1987] James E. Baker. *Reducing Bias and Inefficienry in the Selection Algorithm*. In John J. Grefenstette, editeur, Proceedings of the 2nd International Conference on Genetic Algorithms, Cambridge, MA, USA, July 1987, pages 14–21. Lawrence Erlbaum Associates, 1987. (Cité en page 22.)

[Bakurov & Ross 2018] Illya Bakurov et Brian J. Ross. *Non-photorealistic Rendering with Cartesian Genetic Programming Using Graphics Processing Units*. In Antonios Liapis, Juan Jesús Romero Cardalda et Anikó Ekárt, editeurs, Computational Intelligence in Music, Sound, Art and Design, pages 34–49, Cham, 2018. Springer International Publishing. (Cité en pages 47 et 48.)

[Baldi *et al.* 2014] Pierre Baldi, Peter Sadowski et Daniel Whiteson. *Searching for exotic particles in high-energy physics with deep learning*. Nature communications, vol. 5, 2014. (Cité en pages 105 et 106.)

[Baldi *et al.* 2015] Pierre Baldi, Peter Sadowski et Daniel Whiteson. *Enhanced Higgs boson to τ+ τ- search with deep learning*. Physical review letters, vol. 114, no. 11, pages 111–801, 2015. (Cité en pages 105 et 110.)

[Banzhaf 1993] Wolfgang Banzhaf. *Genetic Programming for Pedestrians*. In Stephanie Forrest, editeur, Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93, page 628, University of Illinois at Urbana-Champaign, 17-21 Juillet 1993. Morgan Kaufmann. (Cité en page 42.)

[Barr *et al.* 2015] Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean et Justyna Petke. *Automated software transplantation*. In Michal Young et Tao Xie, editeurs, Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12-17, 2015, pages 257–269. ACM, 2015. (Cité en page 2.)

[Barricelli 1954] Nils Aaall Barricelli. *Esempi numerici di processi di evoluzione*. Methods, no. 6, pages 45–68, 1954. (Cité en page 14.)

[Bhatnagar 2019] Roheet Bhatnagar. *Unleashing Machine Learning onto Big Data : Issues, Challenges and Trends*. In Aboul Ella Hassanien, editeur, Machine Learning Paradigms : Theory and Application, volume 801 of *Studies in Computational Intelligence*, pages 271–286. Springer, 2019. (Cité en page 60.)

[Brameier 2004] Markus Brameier. *On Linear Genetic Programming*. thesis, Université de Dortmund, 2004. (Cité en page 44.)

[Bramer 2013] Max Bramer. Avoiding overfitting of decision trees, pages 121–136. Springer London, London, 2013. (Cité en page 54.)

[Breiman 1996] Leo Breiman. *Bagging Predictors*. Machine Learning, vol. 24, no. 2, pages 123–140, 1996. (Cité en pages 72 et 84.)

[Browne 2012] Cameron Browne. *Yavalath : Sample Chapter from Evolutionary Game Design*. ICGA Journal, vol. 35, no. 1, pages 20–27, 2012. (Cité en page 47.)

[Cao *et al.* 2016] Van Loi Cao, Nhien-An Le-Khac, Michael O'Neill, Miguel Nicolau et James McDermott. *Improving Fitness Functions in Genetic Programming for Classification on Unbalanced Credit Card Data*. In Giovanni Squillero et Paolo Burelli, editeurs, Applications of Evolutionary Computation, pages 35–45, Cham, 2016. Springer International Publishing. (Cité en page 2.)

[Carbonell *et al.* 1983] Jaime G. Carbonell, Ryszard S. Michalski et Tom M. Mitchell. *An overview of Machine Learning*. In Ryszard S. Michalski, Jaime G. Carbonell et Tom M. Mitchell, editeurs, Machine Learning, pages 3 – 23. Morgan Kaufmann, San Francisco (CA), 1983. (Cité en page 50.)

[CGP 2009] CGP. *Cartesian GP website : http://www.cartesiangp.co.uk*, 2009. En ligne : visité le 27 mars 2017. (Cité en page 86.)

[Chávez *et al.* 2016] Francisco Chávez, Francisco Fernández, César Benavides, Daniel Lanza, Juan Villegas-Cortez, Leonardo Trujillo, Gustavo Olague et Graciela Román. *ECJ+HADOOP : An Easy Way to Deploy Massive Runs of Evolutionary Algorithms*. In Applications of Evolutionary Computation, EvoApplications 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings, Part II, volume 9598 of *Lecture Notes in Computer Science*, pages 91–106. Springer, 2016. (Cité en pages 131 et 132.)

[Chen & He 2014] Tianqi Chen et Tong He. *Higgs Boson Discovery with Boosted Trees*. In Workshop on High-energy Physics and Machine Learning, HEPML 2014, held at NIPS 2014, Montreal, Quebec, Canada, December 8-13, 2014, volume 42 of *JMLR Workshop and Conference Proceedings*, pages 69–80. JMLR.org, 2014. (Cité en page 105.)

[Chokogue 2017] Juvénal Chokogue. Hadoop devenez opérationnel dans le monde du big data. Expert IT. ENI, Paris, France, 2017. (Cité en page 126.)

[Clemente *et al.* 2018] Eddie Clemente, Marlen Meza-Sánchez, Eusebio Bugarin et Ana Yaveni Aguilar-Bustos. *Adaptive Behaviors in Autonomous Navigation with Collision Avoidance and Bounded Velocity of an Omnidirectional Mobile Robot - A Control Theory with Genetic Programming Approach*. Journal of Intelligent and Robotic Systems, vol. 92, no. 2, pages 359–380, 2018. (Cité en page 2.)

[Cohn *et al.* 1994] David Cohn, Les Atlas et Richard Ladner. *Improving Generalization with Active Learning*. Machine Learning, vol. 15, pages 201–221, 1994. (Cité en pages 52 et 53.)

[Cornuéjols & Miclet 2010] Antoine Cornuéjols et Laurent Miclet. Apprentissage artificiel. Algorithmes (Paris). Eyrolles, 2010. (Cité en pages 51 et 56.)

[Cramer 1985a] Nichael Lynn Cramer. *A Representation for the Adaptive Generation of Simple Sequential Programs*. In John J. Grefenstette, editeur, Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985, pages 183–187. Lawrence Erlbaum Associates, 1985. (Cité en page 16.)

[Cramer 1985b] Nicheal Lynn Cramer. *A representation for the adaptive generation of simple sequential programs.* In In J. J. Grefenstette, editor, Proceedings of the 1^{st} International Conference on Genetic Algorithms, pages 183–187, 1985. (Cité en pages 25 et 28.)

[Crepeau 1995] Ronald L. Crepeau. *Genetic Evolution of Machine Language Software*. In Justinian P. Rosca, editeur, Proceedings of the Workshop on Genetic Programming : From Theory to Real-World Applications, pages 121–134, Tahoe City, California, USA, 9 Juillet 1995. (Cité en page 43.)

[Crepinsek *et al.* 2013] Matej Crepinsek, Shih-Hsi Liu et Marjan Mernik. *Exploration and exploitation in evolutionary algorithms : A survey*. ACM Comput. Surv., vol. 45, no. 3, pages 35 :1–35 :33, 2013. (Cité en page 14.)

[Curry & Heywood 2004] Robert Curry et Malcolm I. Heywood. *Towards Efficient Training on Large Datasets for Genetic Programming*. In Advances in Artificial Intelligence, 17th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2004, Proc., volume 3060 of *Lecture Notes in Computer Science*, pages 161–174. Springer, 2004. (Cité en pages 81, 82, 83, 84, 85, 95 et 103.)

[Curry *et al.* 2007] Robert Curry, Peter Lichodzijewski et Malcolm I. Heywood. *Scaling Genetic Programming to Large Datasets Using Hierarchical Dynamic Subset Selection*. IEEE Trans. Systems, Man, and Cybernetics, Part B, vol. 37, no. 4, pages 1065–1073, 2007. (Cité en pages 83, 84 et 89.)

[Dabhi & Chaudhary 2012] Vipul K. Dabhi et Sanjay Chaudhary. *A Survey on Techniques of Improving Generalization Ability of Genetic Programming Solutions*. CoRR, vol. abs/1211.1119, 2012. (Cité en page 54.)

[De Jong & Sarma 1995] Kenneth A. De Jong et Jayshree Sarma. *On Decentralizing Selection Algorithms*. In Larry J. Eshelman, editeur, Proceedings of the 6th International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 15-19, 1995, pages 17–23. Morgan Kaufmann, 1995. (Cité en page 24.)

[De Jong & Spears 1992] Kenneth A. De Jong et William M. Spears. *A formal analysis of the role of multi-point crossover in genetic algorithms*. Annals of Mathematics and Artificial Intelligence, vol. 5, no. 1, pages 1–26, 1992. (Cité en page 19.)

[De Jong 1975] Kenneth Allan De Jong. *Analysis of Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, The University of Michigan, 1975. (Cité en page 24.)

[de la Maza & Tidor 1993] Michael de la Maza et Bruce Tidor. *An Analysis of Selection Procedures with Particular Attention Paid to Proportional and Boltzmann Selection*. In Stephanie Forrest, editeur, Proceedings of the 5th International Conference on Genetic Algorithms, Urbana-Champaign, IL, USA, June 1993, pages 124–131. Morgan Kaufmann, 1993. (Cité en page 24.)

[Dean & Ghemawat 2004] Jeffrey Dean et Sanjay Ghemawat. *MapReduce : Simplified Data Processing on Large Clusters*. In Eric A. Brewer et Peter Chen, editeurs, 6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004, pages 137–150. USENIX Association, 2004. (Cité en pages 3, 63 et 124.)

[Deb 2001] Kalyanmoy Deb. Multi-objective optimization using evolutionary algorithms. John Wiley & Sons, Inc., New York, NY, USA, 2001. (Cité en page 35.)

[DeJong 1988] Kenneth DeJong. *Learning with Genetic Algorithms : An Overview*. Machine Learning, vol. 3, pages 121–138, 1988. (Cité en page 55.)

[Dietterich 2000] Thomas G. Dietterich. *Ensemble Methods in Machine Learning*, 2000. (Cité en page 63.)

[Eiben & Schippers 1998] A. E. Eiben et C. A. Schippers. *On Evolutionary Exploration and Exploitation*. Fundam. Inform., vol. 35, no. 1-4, pages 35–50, 1998. (Cité en page 13.)

[Eiben & Smit 2011] A. E. Eiben et Selmar K. Smit. *Parameter tuning for configuring and analyzing evolutionary algorithms*. Swarm and Evolutionary Computation, vol. 1, no. 1, pages 19–31, 2011. (Cité en page 36.)

[Eiben *et al.* 2007] A. E. Eiben, Zbigniew Michalewicz, Marc Schoenauer et James E. Smith. *Parameter Control in Evolutionary Algorithms*. In Fernando G. Lobo, Cláudio F. Lima et Zbigniew Michalewicz, editeurs, Parameter Setting in Evolutionary Algorithms, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer, 2007. (Cité en page 36.)

[Eklund 2002] Sven E. Eklund. *A Massively Parallel GP Engine in VLSI*. In David B. Fogel *et al.*, editeurs, Proceedings of the 2002 Congress on Evolutionary

Computation CEC2002, pages 629–633. IEEE Press, 12-17 Mai 2002. (Cité en page 43.)

[Eshelman & Schaffer 1993] Larry J. Eshelman et J. David Schaffer. *Real-Coded Genetic Algorithms and Interval-Schemata*. In L. Darrell Whitley, editeur, Proceedings of the Second Workshop on Foundations of Genetic Algorithms. Vail, Colorado, USA, July 26-29 1992., pages 187–202. Morgan Kaufmann, 1993. (Cité en page 19.)

[Espejo *et al.* 2010] Pedro G. Espejo, Sebastián Ventura et Francisco Herrera. *A Survey on the Application of Genetic Programming to Classification*. IEEE Trans. Systems, Man, and Cybernetics, Part C, vol. 40, no. 2, pages 121–144, 2010. (Cité en page 45.)

[Fillon & Bartoli 2007] Cyril Fillon et Alberto Bartoli. *Multi-objective Genetic Programming for Improving the Performance of TCP*. In Marc Ebner *et al.*, editeurs, Genetic Programming, 10th European Conference, EuroGP 2007, Valencia, Spain, April 11-13, 2007, Proceedings, volume 4445 of *Lecture Notes in Computer Science*, pages 170–180. Springer, 2007. (Cité en page 35.)

[Fogel *et al.* 1965] Lawrence J. Fogel, Alvin J. Owens et Micheal J. Walsh. Artificial intelligence through simulated evolution. John Wiley & Sons, New York, 1965. (Cité en pages 10, 11 et 15.)

[Fogel 1992a] David B. Fogel. *An Analysis of Evolutionary Programming*. In D. B. Fogel et W. Atmar, editeurs, Proceedings of the First Annual Conference on Evolutionary Programming, pages 43–51, La Jolla, California, 1992. (Cité en page 16.)

[Fogel 1992b] David B. Fogel. *Evolving Artificial Intelligence*. PhD thesis, University of California, San Diego, 1992. (Cité en page 16.)

[Fogel 1995] David B. Fogel. Evolutionary computation - toward a new philosophy of machine intelligence. IEEE, 1995. (Cité en page 11.)

[Forrest *et al.* 2009] Stephanie Forrest, ThanhVu Nguyen, Westley Weimer et Claire Le Goues. *A genetic programming approach to automated software repair*. In Franz Rothlauf, editeur, Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009, pages 947–954. ACM, 2009. (Cité en page 47.)

[Fortin *et al.* 2012] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau et Christian Gagné. *DEAP : Evolutionary Algorithms Made Easy*. Journal of Machine Learning Research, vol. 13, pages 2171–2175, jul 2012. (Cité en page 132.)

[Fraser 1957] Alex S. Fraser. *Simulation of Genetic Systems by Automatic Digital Computers*. Australian Journal of Biological Sciences, vol. 10, pages 484–491, 1957. (Cité en page 14.)

[Freitas 2002] Alex A. Freitas. Data mining and knowledge discovery with evolutionary algorithms. Springer-Verlag, Berlin, Heidelberg, 2002. (Cité en page 68.)

[Funika & Koperek 2016] Wlodzimierz Funika et Pawel Koperek. *Scaling Evolutionary Programming with the Use of Apache Spark*. Computer Science (AGH), vol. 17, no. 1, pages 69–82, 2016. (Cité en pages 37 et 131.)

[Gandomi *et al.* 2015] Amir Hossein Gandomi, Amir Hossein Alavi et Conor Ryan, editeurs. Handbook of genetic programming applications. Springer, 2015. (Cité en page 45.)

[Gathercole & Ross 1994] Chris Gathercole et Peter Ross. *Dynamic Training Subset Selection for Supervised Learning in Genetic Programming*. In Parallel Problem Solving from Nature - PPSN III, International Conference on Evolutionary Computation, Proc., volume 866 of *Lecture Notes in Computer Science*, pages 312–321. Springer, 1994. (Cité en pages 71, 72, 73, 74, 84 et 136.)

[Gathercole & Ross 1997] Chris Gathercole et Peter Ross. *Small Populations over Many Generations can beat Large Populations over Few Generations in Genetic Programming*. In Genetic Programming 1997 : Proc. of the Second Annual Conf., pages 111–118, San Francisco, CA, 1997. Morgan Kaufmann. (Cité en page 73.)

[Gathercole 1998] Chris Gathercole. *An Investigation of Supervised Learning in Genetic Programming*. Thesis, University of Edinburgh, 1998. (Cité en pages 35 et 73.)

[Gite *et al.* 2017] Balasaheb Gite, Khalid Sayed, Navin Mutha, Saurabhkumar Marpadge et Kshitij Patil. *Surveying various genetic programming (GP) approaches to forecast real-time trends amp ; prices in the stock market*. In 2017 Computing Conference, pages 131–134, July 2017. (Cité en page 2.)

[Goldberg 1989] David E. Goldberg. Genetic algorithms in search optimization and machine learning. Addison-Wesley, 1989. (Cité en pages 10, 14, 17, 20 et 22.)

[Gonçalves & Silva 2013] Ivo Gonçalves et Sara Silva. *Balancing Learning and Overfitting in Genetic Programming with Interleaved Sampling of Training Data*. In Krzysztof Krawiec *et al.*, editeurs, Genetic Programming - 16th European Conference, EuroGP 2013, Vienna, Austria, April 3-5, 2013. Proceedings, volume 7831 of *Lecture Notes in Computer Science*, pages 73–84. Springer, 2013. (Cité en pages 54, 80 et 84.)

[Gonçalves *et al.* 2012] Ivo Gonçalves, Sara Silva, Joana B. Melo et João Manuel de Brito Carreiras. *Random Sampling Technique for Overfitting Control in Genetic Programming*. In Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado et Carlos Cotta, editeurs, Genetic Programming - 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings, volume 7244 of *Lecture Notes in Computer Science*, pages 218–229. Springer, 2012. (Cité en page 54.)

[Grefenstette 1986] John J. Grefenstette. *Optimization of Control Parameters for Genetic Algorithms*. IEEE Trans. Systems, Man, and Cybernetics, vol. 16, no. 1, pages 122–128, 1986. (Cité en page 25.)

[Hansen *et al.* 2015] Nikolaus Hansen, Dirk V. Arnold et Anne Auger. Evolution strategies, pages 871–898. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. (Cité en page 15.)

[Harding & Banzhaf 2011] Simon Harding et Wolfgang Banzhaf. *Implementing cartesian genetic programming classifiers on graphics processing units using GPU.NET*. In Simon Harding *et al.*, editeurs, GECCO 2011 Computational intelligence on consumer games and graphics hardware (CIGPU), pages 463–470, Dublin, Ireland, 12-16 July 2011. ACM. (Cité en page 63.)

[Hillis 1990] W.Daniel Hillis. *Co-evolving parasites improve simulated evolution as an optimization procedure*. Physica D : Nonlinear Phenomena, vol. 42, no. 1, pages 228 – 234, 1990. (Cité en page 133.)

[Hmida *et al.* 2016a] Hmida Hmida, Sana Ben Hamida, Amel Borgi et Marta Rukoz. *Hierarchical Data Topology Based Selection for Large Scale Learning*. In 2016 Intl IEEE Conference on Cloud and Big Data Computing, Toulouse, France, July 18-21, 2016, pages 1221–1226. IEEE, 2016. (Cité en pages 5, 7, 94, 98 et 105.)

[Hmida *et al.* 2016b] Hmida Hmida, Sana Ben Hamida, Amel Borgi et Marta Rukoz. *Sampling Methods in Genetic Programming Learners from Large Datasets : A Comparative Study*. In Plamen Angelov *et al.*, editeurs, Advances in Big Data - Proceedings of the 2nd INNS Conference on Big Data, October 23-25, 2016, Thessaloniki, Greece, volume 529 of *Advances in Intelligent Systems and Computing*, pages 50–60, 2016. (Cité en pages 5 et 7.)

[Hmida *et al.* 2018] Hmida Hmida, Sana Ben Hamida, Amel Borgi et Marta Rukoz. *Scale Genetic Programming for large Data Sets : Case of Higgs Bosons Classification*. In Robert J. Howlett *et al.*, editeurs, Knowledge-Based and Intelligent Information & Engineering Systems : Proceedings of the 22nd International Conference KES-2018, Belgrade, Serbia, 3-5 September 2018., volume 126 of *Procedia Computer Science*, pages 302–311. Elsevier, 2018. (Cité en pages 5, 7 et 145.)

[Hmida *et al.* 2019a] Hmida Hmida, Sana Ben Hamida, Amel Borgi et Marta Rukoz. *Genetic Programming over Spark for Higgs Boson Classification*. In Witold Abramowicz et Rafael Corchuelo, editeurs, Business Information Systems - 22nd International Conference, BIS 2019, Seville, Spain, June 26-28, 2019, Proceedings, Part I, volume 353 of *Lecture Notes in Business Information Processing*, pages 300–312. Springer, 2019. (Cité en pages 5 et 6.)

[Hmida *et al.* 2019b] Hmida Hmida, Sana Ben Hamida, Amel Borgi et Marta Rukoz. *A new adaptive sampling approach for Genetic Programming*. In Proceedings of the 3rd International Conference on Intelligent Computing in Data Sciences, ICDS 2019, October 28-30, 2019, Marrakech, Morocco, 2019. IEEE. (accepté). (Cité en pages 5 et 6.)

[Hofmann 2015] David M. Hofmann. *A Genetic Programming Approach to Generating Musical Compositions*. In Colin Johnson, Adrian Carballal et João Correia,

editeurs, Evolutionary and Biologically Inspired Music, Sound, Art and Design, pages 89–100, Cham, 2015. Springer International Publishing. (Cité en page 47.)

[Hogg 1996] Tad Hogg. *Refining the Phase Transition in Combinatorial Search*. Artif. Intell., vol. 81, no. 1-2, pages 127–154, 1996. (Cité en page 77.)

[Hold-Geoffroy *et al.* 2014] Yannick Hold-Geoffroy, Olivier Gagnon et Marc Parizeau. *Once you SCOOP, no need to fork*. In Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment, page 60. ACM, 2014. (Cité en page 132.)

[Holland 1962] John H. Holland. *Outline for a Logical Theory of Adaptive Systems*. Journal of the ACM, vol. 9, no. 3, pages 297–314, 1962. (Cité en pages 11, 14 et 19.)

[Holland 1975] John H. Holland. Adpatation in natural and artificial systems. University of Michigan Press, Ann Arbor, MI, 1975. (Cité en page 10.)

[Holland 1986] John H. Holland. *Escaping Brittleness : The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems*. In R. S. Michalski, J. G. Carbonell et T. M. Mitchell, editeurs, Machine Learning : An Artificial Intelligence Approach, volume 2. Morgan Kaufmann, Los Altos, CA, 1986. (Cité en page 56.)

[Hunt *et al.* 2010] Rachel Hunt, Mark Johnston, Will N. Browne et Mengjie Zhang. *Sampling Methods in Genetic Programming for Classification with Unbalanced Data*. In AI 2010 : Advances in Artificial Intelligence - 23rd Australasian Joint Conference, Proc., volume 6464 of *Lecture Notes in Computer Science*, pages 273–282. Springer, 2010. (Cité en pages 78, 84 et 85.)

[Iba 1999] Hitoshi Iba. *Bagging, Boosting, and Bloating in Genetic Programming*. In The 1st Annual Conference on Genetic and Evolutionary Computation, Proc., volume 2 of *GECCO'99*, pages 1053–1060, San Francisco, CA, USA, 1999. Morgan Kaufmann. (Cité en pages 66, 72 et 84.)

[Ilhan & Fatih Amasyal 2014] Hamza Ilhan et Mehmet Fatih Amasyal. *Active Learning as a Way of Increasing Accuracy*. International Journal of Computer Theory and Engineering, vol. 6, pages 460–465, 12 2014. (Cité en page 53.)

[Kallel 1999] Leila Kallel. *Convergence des algorithmes génétiques : Aspects spatiaux et temporels*. PhD thesis, Ecole Polytechnique, 1999. Thèse de doctorat Sciences et techniques ECOLE POLYTECHNIQUE 1999 sous la direction de Marc Schoenauer. (Cité en page 20.)

[Karau & Warren 2017] Holden Karau et Rachel Warren. High performance spark. O'reilly, USA, 1st édition, 2017. (Cité en page 138.)

[Keijzer 2003] Maarten Keijzer. *Improving Symbolic Regression with Interval Arithmetic and Linear Scaling*. In Conor Ryan *et al.*, editeurs, Genetic Programming, 6th European Conference, EuroGP 2003, Essex, UK, April 14-16, 2003. Proceedings, volume 2610 of *Lecture Notes in Computer Science*, pages 70–82. Springer, 2003. (Cité en page 33.)

[Keijzer 2004] Maarten Keijzer. *Alternatives in Subtree Caching for Genetic Programming*. In Maarten Keijzer *et al.*, editeurs, Genetic Programming, 7th European Conference, EuroGP2004, Coimbra, Portugal, April 5-7, 2004, Proceedings, volume 3003 of *Lecture Notes in Computer Science*, pages 328–337. Springer, 2004. (Cité en page 35.)

[Khan *et al.* 2019] Asifullah Khan, Aqsa Saeed Qureshi, Noorul Wahab, Mutawarra Hussain et Muhammad Yousaf Hamza. *A Recent Survey on the Applications of Genetic Programming in Image Processing*. CoRR, vol. abs/1901.07387, 2019. (Cité en page 2.)

[Kienzler 2017] Romeo Kienzler. Mastering Apache Spark 2.x. Packt Publishing, 2017. (Cité en page 129.)

[Kilyen & Letia 2018] Attila O. Kilyen et Tiberiu S. Letia. *Hybrid robot controller synthesis with GP and UETPN*. In 2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), pages 1–6, May 2018. (Cité en page 2.)

[Kotecha & Garg 2016] Radhika Kotecha et Sanjay Garg. *Genetic programming-based evolution of classification trees for decision support in banking sector*. IJKESDP, vol. 5, no. 3/4, pages 186–204, 2016. (Cité en page 2.)

[Koza *et al.* 1999] John R. Koza, David Andre, Forrest H Bennett III et Martin Keane. Genetic programming III : Darwinian invention and problem solving. Morgan Kaufman, 1999. (Cité en page 46.)

[Koza *et al.* 2003] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu et Guido Lanza. Genetic programming IV : Routine human-competitive machine intelligence. Kluwer Academic Publishers, 2003. (Cité en page 46.)

[Koza 1992] John R. Koza. Genetic programming - On the programming of computers by means of natural selection. Complex adaptive systems. MIT Press, 1992. (Cité en pages 2, 10, 11, 16, 25, 27, 29, 31 et 47.)

[Kumari *et al.* 2018] Madhulata Kumari, Neeraj Tiwari et Naidu Subbarao. *A genetic programming-based approach and machine learning approaches to the classification of multiclass anti-malarial datasets*. I. J. Computational Biology and Drug Design, vol. 11, no. 4, pages 275–294, 2018. (Cité en page 2.)

[Laney 2001] Douglas Laney. *3D Data Management : Controlling Data Volume, Velocity, and Variety*. Rapport technique, META Group, February 2001. (Cité en page 1.)

[Lang & Witbrock 1988] Kevin J. Lang et Michael J. Witbrock. *Learning to Tell Two Spirals Apart*. In The 1988 Connectionist Models Summer School, pages 52–59. Morgan Kaufmann, 1988. (Cité en pages 78 et 98.)

[Langdon & Banzhaf 2005] William B. Langdon et Wolfgang Banzhaf. *Repeated Sequences in Linear Genetic Programming Genomes*. Complex Systems, vol. 15, no. 4, pages 285–306, 2005. (Cité en page 44.)

[Langdon 2011] William B. Langdon. *Graphics processing units and genetic programming : an overview*. Soft Computing, vol. 15, no. 8, pages 1657–1669, 2011. (Cité en page 63.)

[Larrañaga *et al.* 1999] Pedro Larrañaga, Cindy M. H. Kuijpers, Roberto H. Murga, Iñaki Inza et S. Dizdarevic. *Genetic Algorithms for the Travelling Salesman Problem : A Review of Representations and Operators*. Artif. Intell. Rev., vol. 13, no. 2, pages 129–170, 1999. (Cité en page 17.)

[Lasarczyk *et al.* 2004] Christian Lasarczyk, Peter Dittrich et Wolfgang Banzhaf. *Dynamic Subset Selection Based on a Fitness Case Topology*. Evolutionary Computation, vol. 12, no. 2, pages 223–242, 2004. (Cité en pages 77, 84, 85, 98 et 99.)

[L'Heureux *et al.* 2017] Alexandra L'Heureux, Katarina Grolinger, Hany F. ElYamany et Miriam A. M. Capretz. *Machine Learning With Big Data : Challenges and Approaches*. IEEE Access, vol. 5, pages 7776–7797, 2017. (Cité en pages 60 et 62.)

[Liu & Khoshgoftaar 2004] Yi Liu et Taghi M. Khoshgoftaar. *Reducing Overfitting in Genetic Programming Models for Software Quality Classification*. In 8th IEEE International Symposium on High-Assurance Systems Engineering (HASE 2004), 25-26 March 2004, Tampa, FL, USA, pages 56–65. IEEE Computer Society, 2004. (Cité en page 66.)

[Lohn *et al.* 2005] Jason D. Lohn, Gregory S. Hornby et Derek S. Linden. *An Evolved Antenna for Deployment on Nasaś Space Technology 5 Mission*. In Una-May OŔeilly, Tina Yu, Rick Riolo et Bill Worzel, editeurs, Genetic Programming Theory and Practice II, pages 301–315. Springer US, Boston, MA, 2005. (Cité en page 47.)

[Lones *et al.* 2017] Michael A. Lones, Jane E. Alty, Jeremy Cosgrove, Philippa Duggan-Carter, Stuart Jamieson, Rebecca F. Naylor, Andrew J. Turner et Stephen L. Smith. *A New Evolutionary Algorithm-Based Home Monitoring Device for Parkinson's Dyskinesia*. J. Medical Systems, vol. 41, no. 11, pages 176 :1–176 :8, 2017. (Cité en pages 2 et 47.)

[Louis & Rawlins 1991] Sushil J. Louis et Gregory J. E. Rawlins. *Designer Genetic Algorithms : Genetic Algorithms in Structure Design*. In Richard K. Belew et Lashon B. Booker, editeurs, Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, July 1991, pages 53–60. Morgan Kaufmann, 1991. (Cité en page 38.)

[Luke 2017] Sean Luke. *ECJ Homepage : http : // cs. gmu. edu/ ~eclab/projects/ ecj/*, 2017. En ligne : visité le 27 mars 2017. (Cité en page 86.)

[Maitre 2013] Ogier Maitre. *Genetic Programming on GPGPU cards using EASEA*. In Shigeyoshi Tsutsui et Pierre Collet, editeurs, Massively Parallel Evolutionary Computation on GPGPUs, Natural Computing Series, chapitre 11, pages 227–248. Springer, 2013. (Cité en page 63.)

[Martínez *et al.* 2017] Yuliana Martínez, Enrique Naredo, Leonardo Trujillo, Pierrick Legrand et Uriel López. *A comparison of fitness-case sampling methods for genetic programming*. J. Exp. Theor. Artif. Intell., vol. 29, no. 6, pages 1203–1224, 2017. (Cité en page 81.)

[McDermott & O'Reilly 2015] James McDermott et Una-May O'Reilly. *Genetic Programming*. In Janusz Kacprzyk et Witold Pedrycz, editeurs, Springer Handbook of Computational Intelligence, pages 845–869. Springer, 2015. (Cité en page 50.)

[McKay *et al.* 2010] Robert I. McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan et Michael O'Neill. *Grammar-based Genetic Programming : a survey*. Genetic Programming and Evolvable Machines, vol. 11, no. 3-4, pages 365–396, 2010. (Cité en page 37.)

[Melis 2014] Gábor Melis. *Dissecting the Winning Solution of the HiggsML Challenge.* In HEPML@ NIPS, pages 57–67, 2014. (Cité en page 106.)

[Michalewicz 1996] Zbigniew Michalewicz. Genetic algorithms and data structures - evolution programs (3. ed.). Springer, 1996. (Cité en page 20.)

[Miller & Harding 2012] Julian Francis Miller et Simon Harding. *GECCO 2012 tutorial : cartesian genetic programming*. In Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012, Companion Material Proceedings, pages 1093–1116. ACM, 2012. (Cité en pages 40 et 41.)

[Miller & Thomson 2000] Julian Francis Miller et Peter Thomson. *Cartesian Genetic Programming*. In Genetic Programming, European Conference, Proc., volume 1802 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2000. (Cité en page 39.)

[Miller *et al.* 1998] Julian-Francis Miller, Peter Thomson et Terrence Fogarty. Genetic algorithms and evolution strategies in engineering and computer science : Recent advancements and industrial applications, chapitre Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits : A Case Study, pages 105–131. D. Quagliarella, J. Periaux, C. Poloni,G. Winter, Wiley, first édition, 1998. (Cité en page 39.)

[Mitchell 1997] Tom M. Mitchell. Machine learning. McGraw-Hill, 1997. (Cité en pages 50 et 53.)

[Mittal *et al.* 2018] M. Mittal, V. E. Balas, D. J. Hemanth et R. Kumar. Data intensive computing applications for big data. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1st édition, 2018. (Cité en page 1.)

[Montana 1995] David J. Montana. *Strongly Typed Genetic Programming*. Evolutionary Computation, vol. 3, no. 2, pages 199–230, 1995. (Cité en page 32.)

[Nordin & Banzhaf 1997] Peter Nordin et Wolfgang Banzhaf. *An On-Line Method to Evolve Behavior and to Control a Miniature Robot in Real Time with Genetic Programming*. Adaptive Behaviour, vol. 5, no. 2, pages 107–140, 1997. (Cité en pages 73 et 84.)

[Nordin *et al.* 1999] Peter Nordin, Wolfgang Banzhaf et Frank D. Francone. *Efficient Evolution of Machine Code for CISC Architectures using Instruction Blocks and Homologous Crossover*. In Lee Spector, William B. Langdon, Una-May O'Reilly et Peter J. Angeline, editeurs, Advances in Genetic Programming 3, chapitre 12, pages 275–299. MIT Press, Cambridge, MA, USA, Juin 1999. (Cité en page 43.)

[Nordin 1994] Peter Nordin. *A Compiling Genetic Programming System that Directly Manipulates the Machine Code*. In Kenneth E. Kinnear, Jr., editeur, Advances in Genetic Programming, chapitre 14, pages 311–331. MIT Press, 1994. (Cité en pages 16, 25, 42 et 43.)

[Openshaw & Turton 1994] Stan Openshaw et Ian Turton. *Building New Spatial Interaction Models Using Genetic Programming*. In T. C. Fogarty, editeur, Evolutionary Computing, AISB workshop, Leeds, UK, 11-13 Avril 1994. unpublished. (Cité en page 42.)

[Osman & Laporte 1996] Ibrahim H. Osman et Gilbert Laporte. *Metaheuristics : A bibliography*. Annals of Operations Research, vol. 63, no. 5, pages 511–623, Oct 1996. (Cité en page 11.)

[Paduraru *et al.* 2017] Ciprian Paduraru, Marius-Constantin Melemciuc et Alin Stefanescu. *A distributed implementation using apache spark of a genetic algorithm applied to test data generation*. In Genetic and Evolutionary Computation Conference, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings, pages 1857–1863. ACM, 2017. (Cité en page 130.)

[Paris *et al.* 2001] Grégory Paris, Denis Robilliard et Cyril Fonlupt. *Applying Boosting Techniques to Genetic Programming*. In Pierre Collet *et al.*, editeurs, Artificial Evolution, 5th International Conference, Evolution Artificielle, EA 2001, Le Creusot, France, October 29-31, 2001, Selected Papers, volume 2310 of *Lecture Notes in Computer Science*, pages 267–280. Springer, 2001. (Cité en page 72.)

[Paris *et al.* 2003] Grégory Paris, Denis Robilliard et Cyril Fonlupt. *Exploring Overfitting in Genetic Programming*. In Pierre Liardet others, editeur, Artificial Evolution, 6th International Conference, Evolution Artificielle, EA 2003, Marseilles, France, October 27-30, 2003, volume 2936 of *Lecture Notes in Computer Science*, pages 267–277. Springer, 2003. (Cité en pages 54 et 66.)

[Pawlak *et al.* 2015] Tomasz P. Pawlak, Bartosz Wieloch et Krzysztof Krawiec. *Review and comparative analysis of geometric semantic crossovers*. Genetic Programming and Evolvable Machines, vol. 16, no. 3, pages 351–386, 2015. (Cité en page 17.)

[Peralta *et al.* 2015] Daniel Peralta, Sara del Río, Sergio Ramírez-Gallego, Isaac Triguero, Jose M. Benitez et Francisco Herrera. *Evolutionary Feature Selection for Big Data Classification : A MapReduce Approach*. Mathematical Problems in Engineering, vol. 2015, page 11, 2015. (Cité en pages 131 et 132.)

[Perkis 1994] Tim Perkis. *Stack-Based Genetic Programming*. In Proceedings of the 1994 IEEE World Congress on Computational Intelligence, volume 1, pages 148–153, Orlando, Florida, USA, 27-29 Juin 1994. IEEE Press. (Cité en pages 42 et 44.)

[Petke *et al.* 2014] Justyna Petke, Mark Harman, William B. Langdon et Westley Weimer. *Using Genetic Improvement and Code Transplants to Specialise a C++ Program to a Problem Class*. In Genetic Programming - 17th European Conference, EuroGP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers, volume 8599 of *Lecture Notes in Computer Science*, pages 137–149. Springer, 2014. (Cité en page 2.)

[Poli *et al.* 2008] Riccardo Poli, William B. Langdon et Nicholas Freitag McPhee. A field guide to genetic programming. lulu.com, 2008. (Cité en page 33.)

[Poli 1996] Riccardo Poli. *Discovery of Symbolic, Neuro-Symbolic and Neural Networks with Parallel Distributed Genetic Programming*. Rapport technique CSRP-96-14, University of Birmingham, School of Computer Science, Août 1996. Presented at 3rd International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA'97. (Cité en page 38.)

[Poli 1999] Riccardo Poli. *Parallel Distributed Genetic Programming*. In David Corne, Marco Dorigo et Fred Glover, editeurs, New Ideas in Optimization, Advanced Topics in Computer Science, chapitre 27, pages 403–431. McGraw-Hill, Maidenhead, Berkshire, England, 1999. (Cité en page 38.)

[Preble *et al.* 2005] Stefan Preble, Michal Lipson et Hod Lipson. *Two-dimensional photonic crystals designed by evolutionary algorithms*. Applied Physics Letters, vol. 86, no. 6, page 061111, 2005. (Cité en page 47.)

[Pétrowski & Ben-Hamida 2017] Alain Pétrowski et Sana Ben-Hamida. Evolutionary algorithms. John Wiley & Sons, USA, 2017. (Cité en pages 10 et 17.)

[Purohit *et al.* 2013] Anuradha Purohit, Narendra S. Choudhari et Aruna Tiwari. *Code Bloat Problem in Genetic Programming*. International Journal of Scientific and Research Publications, vol. 3, no. 4, page 1612, apr 2013. (Cité en page 26.)

[Qi *et al.* 2016] Rong-Zhi Qi, Zhi-Jian Wang et Shui-Yan Li. *A Parallel Genetic Algorithm Based on Spark for Pairwise Test Suite Generation*. J. Comput. Sci. Technol., vol. 31, no. 2, pages 417–427, 2016. (Cité en page 130.)

[Rechenberg 1973] Ingo Rechenberg. Evolution strategy : Optimization of technical systems by means of biological evolution. Fromman-Holzboog, Stuttgart, 1973. (Cité en pages 10, 11, 15, 20 et 21.)

[Reed *et al.* 1967] Jon Reed, Robert Toombs et Nils Aall Barricelli. *Simulation of Biological Evolution and Machine Learning : I. Selection of Self-reproducing Numeric Patterns by Data Processing Machines, Effects of Hereditary Control, Mutation Type and Crossing*. Journal of Theoretical Biology, vol. 17, pages 319–342, 1967. (Cité en page 20.)

[Reinsel *et al.* 2018] David Reinsel, John Gantz et John Rydning. *The Digitization of the World From Edge to Core*. Rapport technique US44413318, International Data Corporation, November 2018. (Cité en page 60.)

[Robilliard & Fonlupt 2001] Denis Robilliard et Cyril Fonlupt. *Backwarding : An Overfitting Control for Genetic Programming in a Remote Sensing Application*. In Pierre Collet, Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton et Marc Schoenauer, editeurs, Artificial Evolution, 5th International Conference, Evolution Artificielle, EA 2001, Le Creusot, France, October 29-31, 2001, Selected Papers, volume 2310 of *Lecture Notes in Computer Science*, pages 245–254. Springer, 2001. (Cité en page 54.)

[Rothlauf 2006] Franz Rothlauf. Representations for genetic and evolutionary algorithms (2. ed.). Springer, 2006. (Cité en page 17.)

[Ryan *et al.* 1998] Conor Ryan, J.J. Collins et Michael O'Neill. *Grammatical Evolution : Evolving Programs for an Arbitrary Language*. In Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming, pages 83–95. Springer-Verlag, 1998. (Cité en pages 16, 25 et 37.)

[Ryan *et al.* 2018] Conor Ryan, Michael O'Neill et J. J. Collins, editeurs. Handbook of grammatical evolution. Springer, 2018. (Cité en page 37.)

[Samuel 1959] Arthur L. Samuel. *Some Studies in Machine Learning Using the Game of Checkers*. IBM Journal of Research and Development, vol. 3, no. 3, pages 210–229, 1959. (Cité en page 50.)

[Samuel 1983] Arthur L. Samuel. *AI, Where It Has Been and Where It Is Going*. In Alan Bundy, editeur, Proceedings of the 8th International Joint Conference on Artificial Intelligence. Karlsruhe, FRG, August 1983, pages 1152–1157. William Kaufmann, 1983. (Cité en page 10.)

[Schoenauer *et al.* 1996] Marc Schoenauer, Michele Sebag, Francois Jouve, Bertrand Lamy et Habibou Maitournam. *Evolutionary Identification of Macro-Mechanical Models*. In Peter J. Angeline et K. E. Kinnear, Jr., editeurs, Advances in Genetic Programming 2, chapitre 23, pages 467–488. MIT Press, Cambridge, MA, USA, 1996. (Cité en page 30.)

[Schwefel 1965] Hans-Paul Schwefel. Kybernetische evolution als strategie der experimentelen forschung in der stromungstechnik. Master's thesis, Technical University of Berlin, mar 1965. (Cité en page 14.)

[Schwefel 1977] Hans-Paul Schwefel. Numerische optimierung von computer-modellen mittels der evolutionsstrategie. Birkhäuser, Basel, 1977. (Cité en page 18.)

[Schwefel 1981] Hans-Paul Schwefel. Numerical optimization of computer models. John Wiley & Sons, Ltd., Chichester, 1981. (Cité en pages 20, 21 et 24.)

[Schwefel 1987] Hans Paul Schwefel. Collective phenomena in evolutionary systems. Universität Dortmund. Abteilung Informatik, 1987. (Cité en page 20.)

[Settles 2009] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009. (Cité en page 53.)

[Shashidhara *et al.* 2015] B. M. Shashidhara, S. Jain, V. D. Rao, N. Patil et G. S. Raghavendra. *Evaluation of Machine Learning Frameworks on Bank Marketing and Higgs Datasets*. In 2015 Second International Conference on Advances in Computing and Communication Engineering, pages 551–555, May 2015. (Cité en pages 106, 110, 141, 142 et 144.)

[Simon 2013] Dan Simon. Evolutionary optimization algorithms. John Wiley & Sons, USA, 2013. (Cité en pages 10, 11 et 17.)

[Sims 1991] Karl Sims. *Artificial evolution for computer graphics*. In James J. Thomas, editeur, Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1991, Providence, RI, USA, April 27-30, 1991, pages 319–328. ACM, 1991. (Cité en page 47.)

[Sipper *et al.* 2018] Moshe Sipper, Weixuan Fu, Karuna Ahuja et Jason H. Moore. *Investigating the parameter space of evolutionary algorithms*. BioData Mining, vol. 11, no. 1, pages 2 :1–2 :14, 2018. (Cité en page 36.)

[Smith 1983] Stephen F. Smith. *Flexible Learning of Problem Solving Heuristics Through Adaptive Search*. In Alan Bundy, editeur, Proceedings of the 8th International Joint Conference on Artificial Intelligence. Karlsruhe, FRG, August 1983, pages 422–425. William Kaufmann, 1983. (Cité en page 56.)

[Spears & De Jong 1991] William M. Spears et Kennth A. De Jong. *An Analysis of Multi-Point Crossover*. In Gregory J. E. Rawlins, editeur, Foundations of Genetic Algorithms, pages 301–315, San Mateo, 1991. Morgan Kaufmann. (Cité en page 19.)

[Spector & Robinson 2002] Lee Spector et Alan J. Robinson. *Genetic Programming and Autoconstructive Evolution with the Push Programming Language*. Genetic Programming and Evolvable Machines, vol. 3, no. 1, pages 7–40, 2002. (Cité en page 45.)

[Spector *et al.* 2005] Lee Spector, Jon Klein et Maarten Keijzer. *The Push3 execution stack and the evolution of control*. In Hans-Georg Beyer et Una-May O'Reilly, editeurs, Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, 2005, pages 1689–1696. ACM, 2005. (Cité en page 45.)

[Spector *et al.* 2008] Lee Spector, David M. Clark, Ian Lindsay, Bradford Barr et Jon Klein. *Genetic programming for finite algebras*. In Conor Ryan et Maarten Keijzer, editeurs, Genetic and Evolutionary Computation Conference, GECCO 2008, Proceedings, Atlanta, GA, USA, July 12-16, 2008, pages 1291–1298. ACM, 2008. (Cité en page 47.)

[Spector 2004] Lee Spector. Evolved quantum programs, pages 87–121. Springer US, Boston, MA, 2004. (Cité en page 47.)

[Surry & Radcliffe 1996] Patrick D. Surry et Nicholas J. Radcliffe. *Formal Algorithms + Formal Representations = Search Strategies*. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg et Hans-Paul Schwefel, editeurs, The 4th International Conference on Parallel Problem Solving from Nature, Berlin, Germany, September 22-26, 1996, Proceedings, volume 1141 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 1996. (Cité en page 18.)

[Syswerda 1989] Gilbert Syswerda. *Uniform Crossover in Genetic Algorithms*. In J. David Schaffer, editeur, Proceedings of the 3rd International Conference on Genetic Algorithms, George Mason University, Fairfax, Virginia, USA, June 1989, pages 2–9. Morgan Kaufmann, 1989. (Cité en page 19.)

[Tang *et al.* 2016] Yan Tang, Zhuoming Xu et Yuanhang Zhuang. *Bayesian Network Structure Learning from Big Data : A Reservoir Sampling Based Ensemble Method*. In Hong Gao, Jinho Kim et Yasushi Sakurai, editeurs, Database Systems for Advanced Applications - DASFAA 2016 International Workshops : BDMS, BDQM, MoI, and SeCoP, Dallas, TX, USA, April 16-19, 2016, Proceedings, volume 9645 of *Lecture Notes in Computer Science*, pages 209–222. Springer, 2016. (Cité en page 63.)

[Teller & David 1997] Astro Teller et Andre David. *Automatically Choosing the Number of Fitness Cases : The Rational Allocation of Trials*. In Genetic Programming 1997 : Proceedings of the Second Annual Conference, pages 321–328. Morgan Kaufmann, 1997. (Cité en page 80.)

[Teller & Veloso 1995] Astro Teller et Manuela M. Veloso. *PADO : learning tree structured algorithms for orchestration into an object recognition system*. Rapport technique CMU-CS-95-101, Carnegie-Mellon University.Computer science. Pittsburgh (PA US), 1995. (Cité en pages 16, 25 et 38.)

[Teller & Veloso 1997] Astro Teller et Manuela M. Veloso. *PADO : A new learning architecture for object recognition*. In K. Ikeuchi et M. Veloso, editeurs, Symbolic Vision and Learning, pages 77–112. Oxford Press, 1997. (Cité en pages 38 et 39.)

[UCI 1999a] UCI. *KDD Cup Dataset : https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data*, 1999. En ligne : visité le 16 mai 2019. (Cité en pages 66, 87 et 171.)

[UCI 1999b] UCI. *Spambase Dataset : https://archive.ics.uci.edu/ml/datasets/spambase*, 1999. En ligne : visité le 16 mai 2019. (Cité en page 56.)

[UCI 2014] UCI. *HIGGS Data Set : https://archive.ics.uci.edu/ml/datasets/HIGGS*, 2014. En ligne : visité le 16 mai 2019. (Cité en pages 105 et 175.)

[Umbarkar & Sheth 2015] Anantkumar J. Umbarkar et P. D. Sheth. *Crossover Operators in Genetic Algorithms : A Review*. ICTACT Journal on Soft Computing, vol. 6, pages 1083–1092, October 2015. (Cité en page 17.)

[Vasícek & Sekanina 2015] Zdenek Vasícek et Lukás Sekanina. *Evolutionary Approach to Approximate Digital Circuits Design*. IEEE Trans. Evolutionary Computation, vol. 19, no. 3, pages 432–444, 2015. (Cité en page 2.)

[Veiga *et al.* 2016] Jorge Veiga, Roberto R. Expósito, Xoán C. Pardo, Guillermo L. Taboada et Juan Touriño. *Performance evaluation of big data frameworks for large-scale data analytics*. In James Joshi *et al.*, editeurs, 2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016, pages 424–431. IEEE Computer Society, 2016. (Cité en page 127.)

[Walsh 1999] Toby Walsh. *Search in a Small World*. In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages, pages 1172–1177, 1999. (Cité en page 77.)

[Wasiewicz & Mulawka 2001] Piotr Wasiewicz et Jan J. Mulawka. *Molecular Genetic Programming*. Soft Computing - A Fusion of Foundations, Methodologies and Applications, vol. 2, no. 5, pages 106–113, apr 2001. (Cité en page 25.)

[Whitelaw 2004] Mitchell Whitelaw. Metacreations : Art and artificial life. MIT Press, 2004. (Cité en page 47.)

[Widera *et al.* 2009] Pawel Widera, Jonathan M. Garibaldi et Natalio Krasnogor. *Evolutionary design of the energy function for protein structure prediction*. In Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2009, Trondheim, Norway, 18-21 May, 2009, pages 1305–1312. IEEE, 2009. (Cité en page 47.)

[Wu *et al.* 2016] C. Wu, R. Buyya et K. Ramamohanarao. *Big Data Analytics = Machine Learning + Cloud Computing*. In Rajkumar Buyya, Rodrigo N. Calheiros et Amir Vahid Dastjerdi, editeurs, Big Data Principles and Paradigms, pages 3 – 38. Morgan Kaufmann, 2016. (Cité en page 1.)

[Yang *et al.* 2018] Geunseok Yang, Youngjun Jeong, Kyeongsic Min, Jung-Won Lee et Byungjeong Lee. *Applying Genetic Programming with Similar Bug Fix Information to Automatic Fault Repair*. Symmetry, vol. 10, no. 4, page 92, 2018. (Cité en page 2.)

[Yoo *et al.* 2017] Shin Yoo, Xiaoyuan Xie, Fei-Ching Kuo, Tsong Yueh Chen et Mark Harman. *Human Competitiveness of Genetic Programming in Spectrum-Based Fault Localisation : Theoretical and Empirical Analysis*. ACM Trans. Softw. Eng. Methodol., vol. 26, no. 1, pages 4 :1–4 :30, 2017. (Cité en page 2.)

[Yu & Gen 2010] Xinjie Yu et Mitsuo Gen. Introduction to evolutionary algorithms. Decision Engineering. Springer London, London, 2010. (Cité en pages 10 et 11.)

[Zaharia *et al.* 2012] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker et Ion Stoica. *Resilient Distributed Datasets : A Fault-Tolerant Abstraction for In-Memory Cluster Computing*. In Proceedings of the 9th USENIX Symposium

on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012, pages 15–28. USENIX Association, 2012. (Cité en pages 63 et 128.)

[Zecevic & Bonaci 2016] Petar Zecevic et Marko Bonaci. Spark in action. Manning Publications Co., Greenwich, CT, USA, 1st édition, 2016. (Cité en page 130.)

[Zegklitz & Posík 2015] Jan Zegklitz et Petr Posík. *Model Selection and Overfitting in Genetic Programming : Empirical Study*. In Sara Silva et Anna Isabel Esparcia-Alcázar, editeurs, Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings, pages 1527–1528. ACM, 2015. (Cité en page 54.)

[Zhang & Cho 1999] Byoung-Tak Zhang et Dong-Yeon Cho. Genetic programming with active data selection, volume 1585, chapitre Simulated Evolution and Learning, pages 146–153. Springer, 1999. (Cité en pages 74, 75, 76 et 84.)

[Zhang & Joung 1999] Byoung-Tak Zhang et Je-Gun Joung. *Genetic Programming with Incremental Data Inheritance*. In The Genetic and Evolutionary Computation Conference, Proc., volume 2, pages 1217–1224, Orlando, Florida, USA, 1999. Morgan Kaufmann. (Cité en pages 73, 75, 76 et 84.)

[Zhou *et al.* 2017] Lina Zhou, Shimei Pan, Jianwu Wang et Athanasios V. Vasilakos. *Machine learning on big data : Opportunities and challenges*. Neurocomputing, vol. 237, pages 350–361, 2017. (Cité en page 60.)

ANNEXE A

Attributs de la base KDD'99

TABLE A.1 – Description des attributs KDD'99 [UCI 1999a].

N	Attribut	Description	Type
1	Duration	Duration of the connection in seconds	Cont.
2	Protocol_type	Connection protocol	Disc.
3	Service	Destination network service	Disc.
4	Flag	Status of the connection (Normal or error)	Disc.
5	Src_bytes	Data bytes number transferred from source to destination	Cont.
6	Dst_bytes	Data bytes number transferred from destination to source	Cont.
7	Land	Destination and source port/host are equal	Disc.
8	Wrong_fragment	Total number of wrong fragments	Cont.
9	Urgent	Number of packets with urgent bit activated	Cont.
10	Hot	Number of 'Hot'indicators	Cont.
11	Num_failed _logins	Number of failed login attempts	Cont.
12	Logged_in	Login status (0/1)	Disc.
13	Num _compromised	Number of 'compromised'conditions	Cont.
14	Root_shell	Is root shell obtained (0/1)	Disc.
15	Su_attempted	Is 'su root'command entered (0/1)	Disc.
16	Num_root	Number of operations performed as root	Cont.
17	Num_file _creations	Number of file creations	Cont.
18	Num_shells	Number of shell prompts	Cont.

Suite à la page suivante

Table A.1 – suite

N	Attribut	Description	Type
19	Num_access _files	Number of operations on access control files	Cont.
20	Num_outbound _cmds	Number of outbound commands in an ftp session	Cont.
21	Is_hot_login	Is the logged user 'root'or 'admin'(0/1)	Disc.
22	Is_guest _login	Is the login a 'guest 'login (0/1)	Disc.
23	Count	Number of connections to the same host as the current connection in the past two seconds	Cont.
24	Srv_count	Number of connections to the same service as the current connection in the past two seconds	Cont.
25	Serror_rate	The percentage of connections that have 'SYN'errors among the connections aggregated in count (23)	Cont.
26	Srv_serror_rate	The percentage of connections that have 'SYN'errors among the connections aggregated in srv_count (24)	Cont.
27	Rerror_rate	The percentage of connections that have 'REJ'errors among the connections aggregated in count (23)	Cont.
28	Srv_rerror _rate	The percentage of connections that have 'SYN'errors among the connections aggregated in srv_count (24)	Cont.
29	Same_srv _rate	The percentage of connections to the same service among the connections aggregated in count (23)	Cont.
30	Diff_srv _rate	The percentage of connections to different services among the connections aggregated in count (23)	Cont.
31	Srv_diff _host_rate	The percentage of connections to different destination hosts among the connections aggregated in srv_count (24)	Cont.
32	Dst_host _count	Number of connections to the same destination host	Cont.
33	Dst_host _srv_count	Number of connections to the same service (port number)	Cont.
34	Dst_host_same_srv_-rate	The percentage of connections to the same service among the connections aggregated in dst_host_count (32)	Cont.

Suite à la page suivante

Table A.1 – suite

N	Attribut	Description	Type
35	Dst_host_diff _srv_rate	The percentage of connections to different services among the connections aggregated in dst_host_count (32)	Cont.
36	Dst_host_same_src_-port _rate	The percentage of connections to the same source port among the connections aggregated in dst_host_srv_count (33)	Cont.
37	Dst_host_srv_diff_host _rate	The percentage of connections to different destination hosts among the connections aggregated in dst_host_srv_count (33)	Cont.
38	Dst_host _serror_rate	The percentage of connections with 'SYN'error among the connections aggregated in dst_host_count (32)	Cont.
39	Dst_host_srv_serror_-rate	The percentage of connections with 'SYN'error among the connections aggregated in dst_host_srv_count (33)	Cont.
40	Dst_host _rerror_rate	The percentage of connections with 'RE-J'error among the connections aggregated in dst_host_count (32)	Cont.
41	Dst_host_srv_rerror_-rate	The percentage of connections with 'RE-J'error among the connections aggregated in dst_host_srv_count (32)	Cont.

ANNEXE B

Attributs de la base HIGGS

Tous les attributs sont de type continu. Les 21 premiers attributs (de 1 à 21) sont dits de bas niveau et concernent les propriétés cinématiques. Les 7 derniers sont de haut niveau. Il sont calculés à partir des autres attributs, par des physiciens, pour la discrimination entre les 2 classes (Signal ou backgroud).

TABLE B.1 – Description des attributs de HIGGS [UCI 2014].

N	Attribut	Description
		Attributs de bas niveau
1	lepton_pT	The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the lepton (electron or muon).
2	lepton_eta	The pseudorapidity η of the lepton.
3	lepton_phi	The azimuth angle ϕ of the lepton.
4	met	The missing transverse energy $\overrightarrow{E}_T^{miss}$.
5	met_phi	The azimuth angle ϕ of the missing transverse energy.
6	jet_1_pt	The transverse momentum of jet 1.
7	jet_1_eta	The pseudorapidity η of jet 1.
8	jet_1_phi	The azimuth angle ϕ of jet 1.
9	jet_1_b-tag	The b-quark tag of jet 1.
10	jet_2_pt	The transverse momentum of jet 2.
11	jet_2_eta	The pseudorapidity η of jet 2.
12	jet_2_phi	The azimuth angle ϕ of jet 2.
13	jet_2_b-tag	The b-quark tag of jet 2.
14	jet_3_pt	The transverse momentum of jet 3.
15	jet_3_eta	The pseudorapidity η of jet 3.
16	jet_3_phi	The azimuth angle ϕ of jet 3.
17	jet_3_b-tag	The b-quark tag of jet 3.

Suite à la page suivante

Table B.1 – suite

N	Attribut	Description
18	jet_4_pt	The transverse momentum of jet 4.
19	jet_4_eta	The pseudorapidity η of jet 4.
20	jet_4_phi	The azimuth angle ϕ of jet 4.
21	jet_4_b-tag	The b-quark tag of jet 4.
	Attributs de haut niveau	
22	m_jj	Calculated invariant mass using this formula ($m_A^2 = m_{B+C}^2 = (E_B + E_C)^2 - \|(p_B + p_C)\|^2$). with j : particle jet, l : lepton v : neutrino, b : heavy quark jet, w : W boson.
23	m_jjj	
24	m_lv	
25	m_jlv	
26	m_bb	
27	m_wbb	
28	m_wwbb	

RÉSUMÉ

Dans cette thèse, nous étudions l'adaptation des Programmes Génétiques (GP) pour surmonter l'obstacle du volume de données dans les problèmes Big Data. GP est une méta-heuristique qui a fait ses preuves pour les problèmes de classification. Néanmoins, son coût de calcul est un frein à son utilisation avec les larges bases d'apprentissage.

Tout d'abord, nous effectuons une revue approfondie enrichie par une étude comparative expérimentale des algorithmes d'échantillonnage utilisés avec GP. Puis, à partir des résultats de l'étude précédente, nous proposons quelques extensions basées sur l'échantillonnage hiérarchique. Ce dernier combine des algorithmes d'échantillonnage actif à plusieurs niveaux et s'est prouvé une solution appropriée pour mettre à l'échelle certaines techniques comme TBS et pour appliquer GP à un problème Big Data (cas de la classification des bosons de Higgs).

Par ailleurs, nous formulons une nouvelle approche d'échantillonnage appelée échantillonnage adaptatif, basée sur le contrôle de la fréquence d'échantillonnage en fonction du processus d'apprentissage, selon les schémas fixe, déterministe et adaptatif.

Enfin, nous présentons comment transformer une implémentation GP existante (DEAP) en distribuant les évaluations sur un cluster Spark. Nous démontrons comment cette implémentation peut être exécutée sur des clusters à nombre de nœuds réduit grâce à l'échantillonnage. Les expériences montrent les grands avantages de l'utilisation de Spark pour la parallélisation de GP.

MOTS CLÉS

GP, Big Data, classification, échantillonnage de la base d'apprentissage, échantillonnage adaptif, Spark

ABSTRACT

In this thesis, we investigate the adaptation of GP to overcome the data Volume hurdle in Big Data problems. GP is a well-established meta-heuristic for classification problems but is impaired with its computing cost.

First, we conduct an extensive review enriched with an experimental comparative study of training set sampling algorithms used for GP. Then, based on the previous study results, we propose some extensions based on hierarchical sampling. The latter combines active sampling algorithms on several levels and has proven to be an appropriate solution for sampling techniques that can't deal with large datatsets (like TBS) and for applying GP to a Big Data problem as Higgs Boson classification.

Moreover, we formulate a new sampling approach called "adaptive sampling", based on controlling sampling frequency depending on learning process and through fixed, determinist and adaptive control schemes.

Finally, we present how an existing GP implementation (DEAP) can be adapted by distributing evaluations on a Spark cluster. Then, we demonstrate how this implementation can be run on tiny clusters by sampling. Experiments show the great benefits of using Spark as parallelization technology for GP.

KEYWORDS

GP, Big Data, classification, training set sampling, adaptive sampling, Spark