N^o d'ordre : 87/2019_C/INF

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Thèse DE DOCTORAT
Présentée pour l'obtention du grade de DOCTEUR
en : Informatique

Option : Intelligence Artificielle

Par : Célia HIRECHE

Techniques de Datamining, Approches intelligentes et calcul GPU pour la résolution de Problèmes : Application au problème de satisfiabilité.

Soutenue publiquement, le 20/10/2019, devant le jury composé de :

Mr. H. AZZOUNE	Professeur à l'USTHB	Président du jury
Mme. H. DRIAS	Professeur à l'USTHB	Directrice de thèse
Mme. D. BOUGHACI	Professeur à l'USTHB	Examinatrice
Mme. N. KAMEL	Professeur à l'Univ Sétif 1	Examinatrice
Mr. D. ZOUACHE	Maître de Conférences A à l'Univ Bordj Bouareredj	Examinateur

A mes parents...

Dedicaces

À mes parents qui m'ont toujours soutenu et encouragé, qui ont tout fait pour me permettre de toujours avancer et aller de l'avant. Les mots ne sauraient suffire pour vous remercier. Je vous serai éternellement reconnaissante.

À toi Yema. Je ne sais pas si tu as connaissance de ce qui se passe dans ce bas monde, mais j'espère que si c'est le cas, tu es fière de moi, même si je n'ai pas tenu ma promesse d'être médecin. J'ai trouvé la voie qui me correspond.

À toi, Latefa, qui est parti si tôt. Tu étais si contente que j'obtienne mon concours d'accès au doctorat, je te dédie ce travail. Tu resteras à jamais la personne la plus gentille au monde, la personne qui m'aura le plus inspiré. Puisses-tu reposer en paix.

À l'ensemble de ma petite famille. À mon petit frère Hocine a qui j'espère donner l'envie de faire un peu plus ;)

À l'ensemble de ma famille. À ma grand-mère, tantes, oncles. À vous tous qui m'avez encouragé, chacun à sa manière, je vous remercie et vous dédie ce travail.

À Adel et Wahiba, qui sont toujours à l'écoute et qui ont toujours le bon mot pour m'encourager et me faire avancer. Un grand merci.

À mon ami Nadjib, je te remercie de toujours répondre à mes questions nounoursiya.

Célia HIRECHE.

Alger, le 29 octobre 2019.

Remerciements

Je tiens à remercier vivement le professeur Habiba DRIAS pour ce sujet si passionnant d'une part, et pour son encadrement, sa disponibilité et ses encouragements qui ne cessent de me faire avancer. Je lui en serai toujours reconnaissante.

J'adresse tous mes remerciements au professeur Hamid AZZOUNE pour m'avoir fait l'honneur de présider le jury de soutenance.

Je remercie les professeurs Dalila BOUGHACI et Nadjet KAMEL ainsi que le docteur Djaafar ZOUACHE de l'honneur qu'ile me font de faire partie du jury de soutenance. Qu'ils trouvent l'expression de mon respect.

Je tiens à remercier l'ensemble des enseignants m'ayant permis d'atteindre ce niveau.

Un remerciement particulier aux professeurs Ahmed GUESSOUM, Kamel BOUKHALFA et madame Hadia MOSTEGHANEMI pour leur confiance et encouragement que je n'oublierai pas.

Célia HIRECHE.

Alger, le 29 octobre 2019.

Table des matières

Liste des abréviations

BeeInit : Abeille initiale.

BD : Base de données (transactions).

BDD : Binary Decision Diagram.

BMC : Bounded Model Checking.

BIRCH : Balanced Iterative Reducing and Clustering using Hierarchies.

BSO : Bees Swarm Optimization.

CAH : Classification Ascendante Hiérarchique.

CDCL : Conflict-Driven Clause Learning.

CF : Clustering feature.

CF-Tree : Clustering feature tree.

CLARANS : Clustering Large Applications based on RANdomized Search.

CLIQUE : Clustering In QUEst.

CNF : conjonctive normale forme.

CPU : Central Processing Unit.

CUDA : Compute Unified Device Architecture.

DBSCAN : Density Based Spatial Clustering of Application with Noise.

DLIS : Dynamic Largest Individual Sum.

DPLL : Davis-Putnam-Logemann-Loveland.

DPP : Davis Putnam Procedure.

DRAM : Dynamic Random-Access Memory.

GA : Genetic Algorithm (algorithme génétique).

GA-k-means : Génétique Algorithme – k-means.

GPC : Graphic Processing Cluster.

GPGPU : General Purpose computation on GPU. GPU : Graphics Processing Unit.

GSAT : Greedy SAT.

IA : intelligence artificielle.

IQR : InterQuartile Range.

JW : Jeroslow-Wang.

LT : Liste taboue.

Max-SAT : Maximal Satisfiability.

Max-W-SAT : Maximal Weighted Satisfiability.

MinPts : Minimum of Points.

MinSup : Minimum Support.

MOMS : Minimum Occurrence in clauses of Minimum Sizes.

NP (classe) : Non deterministic Polynomial.

P (classe) : Polynomial.

PAM : Partitioning Around Medoids.

SAT : Problème de satisfiabilité booléenne.

Sb : Best solution.

SIMD : Single Instruction Multiple Data.

SLS : Stochastic Local Search.

SM : streaming multiprocessor.

SP : streaming processor.

Sref : Solution de référence.

STING : STatistical INformation Grid

TPC : Texture Processing Cluster.

VSIDS : Variable State Independent Decaying Sum.

Liste des figures

Liste des tableaux

Liste des algorithmes

Liste des Equations

Résumé

L'originalité de ce travail réside dans l'exploitation des technologies de datamining pour la résolution de problèmes. Deux grandes phases définissent ce travail. La première consiste à déterminer la technique de clustering qui convient le mieux à chaque instance SAT en fonction de la distribution de celle-ci. Cette technique est ensuite appliquée dans le but de réduire la complexité de chaque instance en créant des sous-instances pouvant être résolues indépendamment dans la deuxième phase. Dans l'étape de résolution, les algorithmes DPLL ou BSO sont exécutés en fonction du nombre de variables à instancier dans chaque cluster. Cette stratégie de résolution en deux phases fournit une meilleure efficience de la résolution. Le problème de la satisfiabilité booléenne (SAT) est considéré dans cette étude en raison de son importance en intelligence artificielle (IA) et l'impact de sa résolution sur l'ensemble des problèmes complexes.
Deux principales distributions ont été observées lors de l'exploration des instances SAT. La première définit un espace où les variables sont dispersées formant des régions de densité considérable, parsemées de régions de faible densité ou de régions vides. La seconde distribution ne révèle aucune région de densité significative, les variables sont dispersées aléatoirement. Par ailleurs, certaines de ces instances possèdent une particularité dont les variables sont de haute occurrence.
Pour chaque distribution, une technique de clustering est associée. Les techniques de clustering basées sur la densité constituent le type de clustering le plus approprié pour la première distribution. Par ailleurs, le clustering basé sur les grilles et les règles d'associations semblent être les techniques de regroupement les plus appropriées pour la deuxième distribution ainsi que pour les instances dont la particularité est d'avoir des variables de haute fréquence. Nous proposons dans cette thèse une adaptation de ces techniques pour chacune des distributions. De plus, une parallélisation de la résolution est proposée par le biais de l'utilisation du calcul CPU/ GPU.

Mots clés :
Problème de Satisfiabilité, Complexité computationnelle, Résolution de problème, Datamining, Etude Statistique, Distribution, BSO, DPLL, Parallèlisme, Calcul GPU.

Abstract

The originality of this work lies in the exploitation of datamining technologies for problem solving. Two major phases define this work. The first one is to determine the best clustering technique for each SAT instance based on its distribution. This technique is then applied to reduce the complexity of each instance by creating sub-instances that can be solved independently in the second phase. In the resolution step, the DPLL or BSO algorithms are executed according to the number of variables to be instantiated in each cluster. This two-phase resolution strategy provides better resolution efficiency. The problem of Boolean satisfiability (SAT) is considered in this study because of its importance in artificial intelligence (AI) and the impact of its resolution on all complex problems. Two main distributions were observed during SAT instance exploration. The first defines a space where the variables are scattered forming regions of considerable density interspersed with low density regions or empty regions. The second distribution does not reveal any region of significant density, the variables are scattered randomly. Moreover, some of these instances have the particularity that almost all their variables are of high occurrence. For each distribution, a clustering technique is associated. Density-based clustering techniques are the most appropriate type of clustering for the first distribution. Meanwhile, clustering based on grids and mining frequent patterns seem to be the most appropriate clustering techniques for the second distribution as well as for instances whose particularity is to have high frequency variables. We propose in this thesis a modelling of these techniques for each of the distributions. Moreover, a parallelization of the resolution is proposed through the use of the CPU/GPU calculation.

Keywords :
Satisfiability problem, Computational Complexity, Problem solving, Datamining, Statistical study, Distribution, BSO, DPLL, Parallelism, GPU.

Introduction générale

La résolution de problèmes est l'un des principaux volets de l'intelligence artificielle.
Plusieurs approches ont été proposées au cours des années précédentes. Les méthodes de résolution complètes ou exactes garantissent de trouver une solution à un problème ou de démontrer que celui-ci ne peut pas en avoir. À l'inverse, les algorithmes incomplets sont basés sur l'approximation et permettent d'obtenir la solution la plus optimale possible. Pour des instances de taille importante, la première catégorie engendre une explosion combinatoire et un temps de calcul considérable. La seconde catégorie ne peut pas garantir une solution même s'il en existe une.
L'objectif de cette thèse est d'apporter une nouvelle approche intelligente de résolution basée sur l'exploration des instances de problèmes. L'approche proposée s'articule en deux phases. Dans la première, une étude exploratoire des instances est réalisée et permet de déterminer différents modèles de distribution des variables. Un algorithme de clustering est désigné comme étant approprié à chacune de ces distributions. Le but de cette étape est d'appliquer le clustering le plus approprié à l'instance afin d'en amoindrir sa complexité. Ces sous-instances sont alors résolues indépendamment en utilisant les algorithmes Davis-Putnam-Logemann-Loveland DPLL Davis et al. [1962] et Bees Swarm Optimization BSO Drias et al. [2005] dans la seconde phase.
L'étude de la distribution des variables sur les clauses nous a permis de déterminer deux principaux modèles. Un modèle où les variables sont dispersées dans l'espace de manière à former des régions de forte densité parsemées de régions de moindre densité voire de densité nulle. Le second modèle définit une dispersion aléatoire des variables sans formation de région de densité considérable. Par ailleurs, certaines instances proposant ce deuxième modèle de distribution possèdent la particularité que leurs variables aient une fréquence considérable.
Dans le premier cas, l'algorithme Density Based Spatial Clustering of Application with Noise (DBSCAN) Han et al. [2011] Ester et al. [1996] semble être le plus approprié et fait l'objet de notre première adaptation pour les instances SAT. Deux différentes adaptations ont été proposées. Une adaptation dans un espace bidimensionnel et une dans un espace multidimensionnel.
Pour la seconde distribution des variables, l'algorithme STatistical INformation Grid (STING) Han et al. [2011] Wang et al. [1997] est modélisé de différentes manières.
Une solution d'amélioration de l'algorithme Apriori est proposée, ainsi qu'une hybridation de l'algorithme Apriori Agrawal et al. [1994] avec k-means Lloyd [1982] pour le clustering des instances présentant la particularité que l'ensemble de leurs variables aient une importante occurrence.

Enfin, une parallélisation CPU/GPU est proposée pour la résolution parallèle des sous-instances obtenues après clustering.
Les résultats des expérimentations réalisées sur des instances de problème SAT ont démontré l'efficacité de ces techniques et l'impact de l'exploration de la technologie offerte par le datamining sur l'efficience des algorithmes de résolution.
Ce document se présente en deux principales parties. Dans la première, le problème de satisfiabilité est d'abord introduit, suivi par un chapitre dédié à l'exploration des données et des différentes techniques datamining. Un troisième chapitre introduit la technologie GPU. Les différentes contributions sont introduites dans la seconde partie de la thèse. On commence par une étude exploratoire des instances SAT, suivi par la adaptation des différentes techniques de clustering. À savoir, l'adaptation bidimensionnelle et multidimensionnelle du DBSCAN pour les instances SAT, l'approche de clustering basée sur les grilles et enfin l'approche de clustering basée sur les règles d'association. Le dernier chapitre introduit le calcul GPU pour la résolution de SAT.
Une conclusion générale de ce travail est présentée ainsi que quelques perspectives.

1 Le problème de Satisfiabilité entre Théorie et Technologies

1.1 Introduction

Le problème de satisfiabilité booléenne SAT est l'un des problèmes les plus étudiés de nos jours. En effet, ce dernier représente le premier problème à avoir été démontré NP-Complet Cook [1971].
Une instance de problème SAT se définit comme une fonction booléenne sous forme normale conjonctive (CNF) dont le but est de trouver une instanciation de variables satisfaisant l'ensemble de l'instance.
Ce problème a suscité un grand intérêt de la part de la communauté de l'Intelligence Artificielle, diverses études ont été menées dans le but de trouver le meilleur solveur pour ce problème. Plusieurs catégories d'algorithmes et de solveurs ont donc vu le jour.
Dans ce chapitre, après une définition des notions de base sur les problèmes NP-Complets Garey et Johnson [1979] en général et le problème SAT en particulier, les algorithmes et solveurs les plus populaires seront présentés pour ce problème.

1.2 Problème, Complexité et NP-Complétude

Problème

Un problème Garey et Johnson [1979] est défini par une question générale sur des données dont les valeurs sont inconnues. Formellement, un problème est décrit par **une instance** spécifiant des données et **une question** correspondant à l'état final à atteindre ou des conditions à satisfaire.
Le problème SAT est défini comme suit :

- **Instance :** m clauses C_i formées à partir de n variables.
- **Question :** la formule $\phi = C_1, C_2, \ldots, C_m$ est-elle satisfiable ? Existe-t-il une interprétation pour laquelle la formule est vraie ?

Un problème est dit *problème de décision* lorsque la question correspondante a pour réponse : Oui ou Non.

Complexité des problèmes

Dans la théorie de la complexité Garey et Johnson [1979], les problèmes sont classés selon la difficulté de leur résolution. Cette complexité est relative à la complexité des algorithmes de résolution de ces problèmes.
La complexité spatiale d'un algorithme est définie par la quantité de l'espace mémoire que peut occuper le programme lors de son exécution. La complexité temporelle est définie par le temps que met le programme à s'exécuter et est exprimée en nombre d'instructions exécutées par l'algorithme. Les deux types de complexité se mesurent en fonction de la taille de l'instance à résoudre.
Remarque : Par abus de langage, lorsque l'on parle de complexité d'un algorithme, on parle de sa complexité temporelle au pire cas.
Un problème est dit solvable en un temps polynomial s'il possède un algorithme de résolution dont la complexité est polynomiale et plus précisément non exponentielle en fonction de la taille du problème.

NP-Complétude

À partir de cette définition de la complexité, deux grandes classes de complexité ont été définies, à savoir la classe P et la classe NP.
La classe P représente les problèmes pouvant être résolus en temps polynomial par une machine de Turing déterministe. Contrairement à la classe NP qui regroupe les problèmes pouvant être résolus en un temps polynomial par une machine de Turing non déterministe.
Un algorithme non déterministe comporte deux phases : une phase de génération d'une solution quelconque suivie d'une phase de vérification si la solution engendrée est correcte ou non.

En 1971, Stephen COOK Cook [1971] a montré que le problème SAT est NP-complet et depuis l'émergence de la classe des problèmes NP-Complets regroupant les problèmes de décision les plus difficiles à résoudre. Un problème est NP-Complet si les deux conditions suivantes sont vérifiées :

- π appartient à la classe NP.
- Pour un problème π' connu comme étant NP-Complet, il existe une transformation polynomiale entre π' et π.

Soient deux problèmes π_1 et π_2 , la réduction polynomiale de π_1 à π_2 est définie par une fonction associant à toute instance de π_1 une instance de π_2 $f : D_{\pi_1} \rightarrow D_{\pi_2}$, et devant satisfaire les deux conditions suivantes :

- f est exécutable en un temps polynomial.
- Pour toute instance I de π_1, $I \in Y_1$ si et seulement si $f(I) \in Y_{\pi_2}$. Avec Y_{π_i} l'ensemble des instances dont la réponse au problème π_i est *oui*.

1.3 Problème de Satisfiabilité booléenne –SAT-

1.3.1 Notions de logique

En logique mathématique, la satisfiabilité est l'un des concepts élémentaires consistant à déterminer si une formule est satisfiable ou pas.

Les structures logiques nécessaires à la définition du problème SAT sont représentées par :

- **Un littéral** qui est une variable propositionnelle x_i ou sa négation $\overline{x_i}$.
- **Une clause** qui est une disjonction de littéraux.
- Une conjonction de **clauses**, autrement dit une conjonction de disjonctions de littéraux (forme CNF).

1.3.2 Description du problème SAT

Le problème SAT se définit par un ensemble de clauses sous forme normale conjonctive (CNF) dont le but est de trouver une assignation aux différentes variables de manière à ce que toutes les clauses soient satisfaites. Une clause C_i est satisfaite si et seulement si au moins un de ses littéraux est satisfait (sa valeur est à un 1). Par conséquent, une clause est dite insatisfiable si tous ses littéraux sont insatisfaits (leur valeur est à 0). Les deux conditions suivantes doivent être respectées dans une instance de problème SAT :

- Une clause ne doit pas contenir un littéral et sa négation (Tautologie).
- Une clause ne doit pas contenir un même littéral plus d'une fois.

Le problème SAT se définit formellement, comme suit Garey et Johnson [1979] :

- **Instance :** un ensemble de m clauses C_i formées à partir de n variables.
- **Question :** la formule $\phi = C_1, C_2, \ldots, C_m$ est-elle satisfiable ? Existe-t-il une interprétation pour laquelle la formule est vraie ?

Une instance de problème est représentée en pratique, par une table ou une matrice de contingence qui détermine la relation entre les variables et les clauses. Chaque case X_{ij} de cette matrice est représentée par la ligne i correspondant à la clause C_i, et la colonne j correspondant à un littéral appartenant à la clause. Le littéral X_{ij} prend la valeur 1 lorsqu'il se présente sous forme positive et -1 lorsqu'il se présente sous forme négative. Il prend la valeur 0 lorsqu'il n'existe pas dans la clause.

Exemple 1.1 : Soient l'ensemble de variables $V = x_1, x_2, x_3, x_4$, l'ensemble des clauses $C = C_1, C_2, C_3$ et l'instance SAT suivante :
$C_1 = x_1, x_2$
$C_2 = \overline{x_1}, x_2, x_3$
$C_3 = x_1, x_2, \overline{x_4}$
Cette instance est représentée comme suit, par la matrice de contingence :

$$\text{Mc=} \quad \begin{pmatrix} 1 & 1 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ 1 & 1 & 0 & -1 \end{pmatrix}$$

Une solution potentielle à cette instance est : $\{x_1 = 1; x_2 = 1; x_3 = 1; x_4 = 0\}$.

1.4 Variantes du problème SAT

Dans le but d'explorer le problème SAT, des variantes de complexité moindre ont été introduites. Les exemples comme k-SAT, Horn-SAT décrit ci-après correspondent à des formats particuliers d'instance alors que le problème Max-SAT peut être associé à n'importe quel type d'instance.

k-SAT

Dans une instance k-SAT, k est un entier supérieur ou égal à 2. Il exprime la longueur des clauses, autrement dit le nombre de littéraux par clauses. Nous savons que le problème 2-SAT est polynomial et que pour $k \geq 3$, le problème k-SAT est NP-complet. Un exemple d'une instance 2-SAT en considérant V de l'exemple 1.1 est :
$C_1 = x_1, x_2$
$C_2 = \overline{x_1}, x_2$
$C_3 = x_3, \overline{x_4}$
Une solution potentielle à cette instance est : $\{x_1 = 0; x_2 = 1; x_3 = 0; x_4 = 0\}$.
Un exemple d'instance 3-SAT (k=3) est :
$C_1 = x_1, x_2, \overline{x_3}$
$C_2 = \overline{x_1}, x_2, x_4$
$C_3 = \overline{x_2}, x_3, \overline{x_4}$

Une solution potentielle à cette instance est : $\{x_1 = 0; x_2 = 1; x_3 = 1; x_4 = 1\}$.

Horn-SAT

Une instance Horn-SAT est définie par un ensemble de clauses où chaque clause comporte au plus un littéral sous forme positive ou négative. Un exemple d'instance édifiant d'une instance Horn-SAT est :
$C_1 = x_1, \overline{x_2}, \overline{x_3})$
$C_2 = \overline{x_1}, \overline{x_2}, x_4$
$C_3 = \overline{x_2}, x_3$
Une solution potentielle à cette instance est : $\{x_1 = 0; x_2 = 0; x_3 = 1; x_4 = 1\}$.
Le problème Horn-SAT a été démontré comme appartenant à la classe P.

Max-SAT

Quand une instance SAT est contradictoire, une question serait de déterminer le maximum de clauses satisfiables afin peut-être d'éliminer les clauses qui causent l'état d'insatisfiabilité. Le problème associé à cette question est appelé Max-SAT. Formellement, Max-SAT est défini comme suit :

— Instance : un ensemble de m clauses C_i formées à partir de n variables et k un nombre entier.

— Question : Pour la formule $\phi = C_1, C_2, \ldots, C_m$, existe-t-il une interprétation qui satisfasse un nombre de clauses supérieur ou égal à k ?

Max-w-SAT

Le Max-w-SAT est une version de Max-SAT où les clauses sont pondérées. À chaque clause C_i, est associée un poids p_i strictement positif. Dans ce cas, le problème est défini comme suit :

— Instance : un ensemble de m clauses C_i formées à partir de n variables, à chaque clause est associé un poids $p \geq 1$ et k un nombre entier.

— Question : soit la formule $\phi = C_1, C_2, \ldots, C_m$ et k un nombre entier, existe-t-il une interprétation qui satisfasse un nombre de clauses dont la somme des poids est supérieure ou égale à k.

1.5 Les Solveurs SAT

étant le premier problème à avoir été prouvé comme étant NP-complet, le problème SAT est considéré comme le pilier des problèmes NP-complets. En effet, sa résolution aurait un impact sur l'ensemble des problèmes NP-complets.
Un nombre important de solveurs SAT ont été développés depuis les années 60 pour contribuer à la résolution d'autres problèmes complexes. Ils sont divisés en deux grandes catégories selon qu'ils se basent sur des méthodes complètes ou sur des méthodes incomplètes. Les méthodes complètes consistent à déterminer une solution effective et exacte. Ces solveurs sont par conséquent aptes à trouver la bonne solution ou à prouver dans le cas échéant que l'instance ne peut avoir de solution. Par ailleurs, les méthodes incomplètes, contrairement à la première catégorie, ne sont pas capables de déterminer à chaque fois une bonne solution ni de prouver l'existence d'une solution. Elles engendrent souvent des solutions approximatives.

1.5.1 Solveurs complets

Les solveurs complets sont basés sur des algorithmes exacts capables de déterminer une bonne solution. Le principe général de ces algorithmes est de rechercher dans l'ensemble des solutions possibles, celle qui répond exactement au problème. Souvent la recherche de la solution passe par la construction d'une arborescence débutant par un élément de la solution et aboutissant à un autre et ainsi déterminer tous les éléments de la solution.

L'algorithme de Davis-Putnam-Logemann-Loveland (DPLL)

L'algorithme DPLL ou Davis-Putnam-Logemann-Loveland Davis et al. [1962], extension et amélioration du Davis-Putnam (DPP) Davis et Putnam [1960], est l'un des algorithmes complets les plus utilisés dans la résolution du problème SAT. Il consiste en une recherche dans un graphe où les

nœuds représentent les appels à la fonction DPLL alors que les feuilles représentent un état de satisfaction ou d'échec. C'est un algorithme récursif utilisant le principe de retour arrière. En cas d'incohérence ou de conflit défini par la présence d'une clause vide après l'assignation à un niveau donné, des deux valeurs de vérité possibles d'une variable, l'algorithme effectue un retour arrière chronologique partant du premier niveau précédant le conflit. L'instance de problème est dite insatisfiable si l'algorithme retourne jusqu'à la racine et que les deux valeurs de vérité de cette variable induisent à une incohérence.
Deux procédures caractérisent l'algorithme DPLL. La procédure de *propagation unitaire* qui consiste en la satisfaction des clauses unitaires. Une clause unitaire désigne une clause ne possédant qu'un et un seul littéral non encore assigné. Ce littéral doit alors prendre la valeur vraie pour que la clause soit satisfaite. La procédure de *propagation de littéraux* pures qui consiste à assigner à vrai tout littéral présent dans l'instance sous une et une seule forme (positive ou négative).

Algorithm 1 Algorithme Davis-Putnam-Logemann-Loveland

Entrées Lcl : Liste de clauses
Debut
Si Lcl est vide Alors SAT
Sinon
    Si Lcl contient des clauses vides alors UNSAT
    Sinon
        Assigner toutes les variables impliquées dans des clauses unitaires et propager
        Assigner tous les litéraux pures et propager
        Pour toutes les variables restantes, tester les deux valeurs possibles
Fin

Complexité :

La complexité du DPLL est de $\theta(2^n)$. En effet, le pire cas de cet algorithme est représenté par le cas où aucune clause unitaire ni aucun littéral pur n'existe. Dans ce cas, l'algorithme génère toutes les solutions possibles. Si l'on considère n variables, le nombre de solutions possibles est de 2^n.

De nombreuses améliorations et variantes du DPLL ont été proposées. Parmi lesquelles, nous citons deux grandes catégories, les algorithmes utilisant des heuristiques de choix de variable de décision ainsi que les approches basées sur l'apprentissage de clauses à partir de conflit (CDCL).

Heuristiques de choix de variables de décision

Le choix de la variable à instancier pour l'algorithme DPLL est primordial. En effet, un mauvais choix peut conduire à l'exploration de toutes les solutions possibles alors qu'un choix judicieux permettrait une meilleure exploration de l'arbre de recherche. Plusieurs heuristiques ont

été proposées, parmi lesquelles on cite :

— L'heuristique **DLIS** (Dynamic Largest Individual Sum) qui choisit comme variable de décision la variable apparaissant dans le plus grand nombre de clauses non satisfaites. Cette heuristique a été introduite dans le solveur GRASP (Generic seaRch Algorithm for the Satisfiability Problem) Marques-Silva et Sakallah [1999].

— Dans Chaff Moskewicz [2001], l'heuristique **VSIDS** (Variable State Independent Decaying Sum) est utilisée. Cette heuristique prend en compte la polarité des variables dans le choix de la variable de décision. L'heuristique associe alors pour chaque variable et pour chaque polarité un compteur relatif au nombre de clauses dans lesquelles apparaissent ces variables. La variable dont le compteur est maximal est sélectionnée.

— L'heuristique **MOMS** (Minimum Occurrence in clauses of Minimum Sizes) Freeman [1995] où la variable à instancier correspond à la variable la plus répétée dans les clauses de petite taille. Ce choix a pour but de favoriser la *propagation unitaire*.

— Enfin, Jeroslow et Wang introduisent l'heuristique Jeroslow-Wang **JW** Jeroslow et Wang [1990] où à chaque variable est associé un poids correspondant à la somme des poids des deux littéraux formés par cette variable. Le poids d'un littéral est calculé par la formule suivante ; $p(l) = \sum_{l \in c \in C} 2^{-|c|}$
Avec p le poids du littéral. l le littéral. c la clause à laquelle appartient le littéral et C l'ensemble des clauses. La variable ayant le plus grand poids est sélectionnée comme variable de décision.

Conflict-Driven Clause Learning (CDCL)

Les algorithmes CDCL Biere et al. [2009], contrairement au DPLL, offrent la possibilité de retour arrière non chronologique et un apprentissage de clause lors de conflits.
Dans l'approche CDCL Biere et al. [2009], chaque variable est définie par une **valeur** $v \in \{0, i, 1\}$. i désignant le fait que la valeur de la variable est encore indéfinie. Un **antécédent** $\alpha \in \theta \cup NULL$ indiquant la clause ayant induit l'assignation d'une variable impliquée. Les variables de décision ne possèdent pas d'antécédent et prennent par conséquent la valeur NULL. Ainsi qu'un **niveau de décision** $\sigma \in \{-1, 0, 1, \ldots, |X|\}$ indiquant le niveau de profondeur de l'arbre de décision auquel la variable a été assignée. Pour les variables de décisions, une pile de décision fixe ce niveau. À l'opposé, le niveau de décision d'une variable impliquée est défini par $\sigma(x_i) = max(\{0\} \cup \{\sigma(x_j) | x_i \in \omega \wedge x_j \neq x_i\})$, o lorsque la clause est unitaire et le plus haut niveau de décision lorsqu'elle est impliquée par une clause ω (antécédent). Les variables non encore assignées ont un niveau de décision de -1.

Ces différents paramètres relatifs aux variables sont projetés sur un

graphe d'implication Silva et Sakallah [1996] dans le but de détecter les éventuels conflits. Ce graphe $G = \{V, E\}$ est dirigé et acyclique. Les sommets V désignent l'assignation d'une variable. Le nœud spécial κ désigne un conflit. Les arêtes E définissent les implications en reliant une variable x_i à la variable x l'ayant induite par clause unitaire. Ces arêtes portent le nom ou le numéro de la clause d'implication. Ainsi, les sommets du graphe comportent l'écriture suivante $x_i = v@\sigma$ qui signifie que la variable x_i prend la valeur v au niveau de décision σ. Alors que les arêtes prennent la valeur ω ou encore C_j représentant la clause impliquant cette assignation.

Exemple 1.2 : Soit l'instance SAT suivante ;
$C_1 = x_1, x_2$
$C_2 = x_1, \overline{x_2}, x_3$
$C_3 = x_1, \overline{x_3}, \overline{x_4}$
Considérons que la variable x_1 ait la valeur 0 au niveau décisionnel 3 noté $x_1 = 0@3$. Le graphe d'implication de cette instance est comme suit (figure 1.1) ;

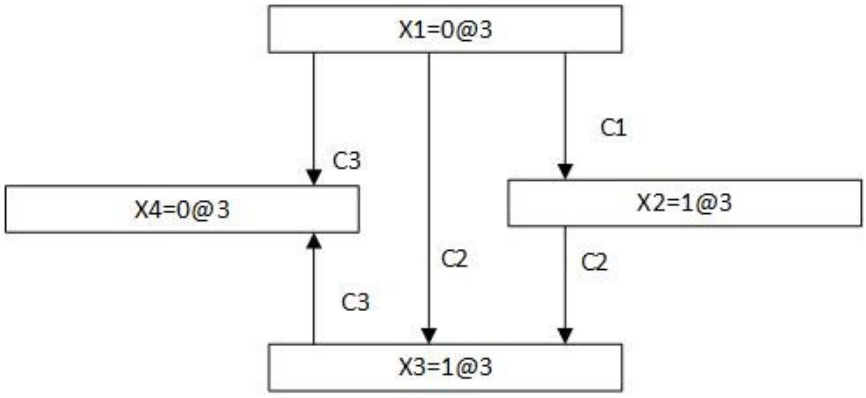


FIGURE 1.1 – *Graphe d'implication (exemple 1.2)*

Dans un graphe d'implication, un conflit dénoté , correspond à un état où la même variable apparait sous les deux formes positive et négative conduisant à une incohérence.

Exemple 1.3 : Soit l'instance SAT suivante ;
$C_1 = x_1, x_2, \overline{x_3}$
$C_2 = x_1, \overline{x_4}$
$C_3 = x_3, x_4, x_5$
$C_4 = x_4, \overline{x_5}$
Supposons que les variables x_1 et x_2 aient la valeur 0 aux niveaux décisionnels 2 et 3 respectivement $(x_1 = 0@2, x_2 = 0@3)$. Le graphe d'implication de cette instance est comme suit (figure 1.2).

Cette figure montre un conflit dans la clause $C_4 = x_4, \overline{x_5}$ où x_4 doit être assignée à 1 ou x_5 à 0, alors que ces derniers ont des valeurs opposées. Une nouvelle clause doit alors être apprise et ajouter à la base de clauses. En présence de conflit, une séquence de processus de résolutions est établie dans le but de générer une nouvelle clause dite *learnt clause*. Un processus de résolution de deux clauses C_i, C_j, noté $C_i \odot C_j$, est défini par

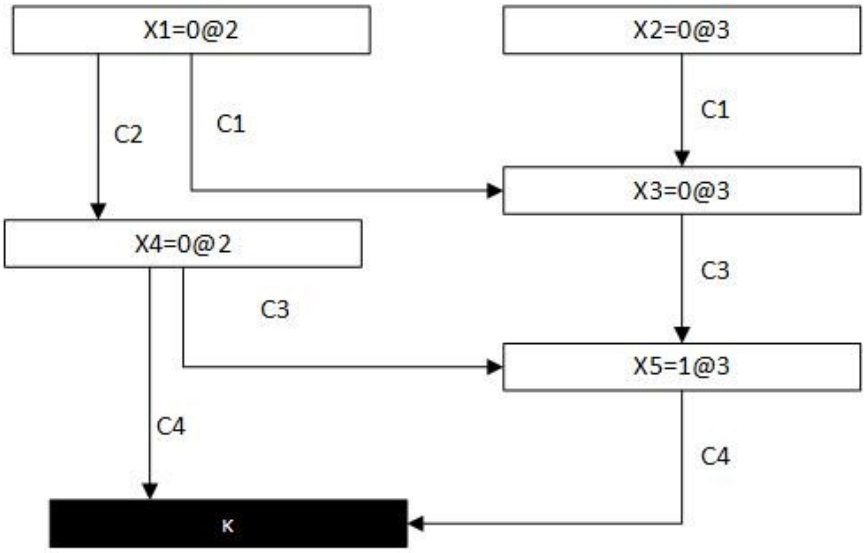


FIGURE 1.2 – *Graphe d'implication avec conflit (exemple 1.3)*

l'union des littéraux de ces clauses à l'exception du littéral x et x, communément appelé résolvante $r_{ij} = \{C_i \cup C_j\} - \{x_k, \overline{x_k}\}$.
Ce processus d'apprentissage de nouvelle clause, passe par plusieurs clauses intermédiaires définies comme suit :

$$C_L^{d,i} = \begin{cases} \alpha(\kappa) & \text{si i=o} \\ C_L^{d,i-1} \odot \alpha(l) & \text{si } i \neq 0 \text{ , } \xi(C_L^{di-1}, l, d) = 1 \\ C_L^{d,i-1} & \text{si } i \neq 0, \forall l, \xi(C_L^{d,i-1}, l, d) = 0 \end{cases}$$

Où la clause de conflit est considérée comme clause de départ du processus $(i = 0)$. Un processus de résolution est effectué à chaque itération entre la clause courante (intermédiaire) et une des clauses antécédentes d'un des littéraux de cette clause assignée par implication à ce niveau de décision.
La condition suivante est appliquée pour vérifier que la clause C est antécédente au littéral l au niveau décisionnel d.

$$\xi(C, l, d) = \begin{cases} 1 & \text{si } l \in C, \delta(l) = d, \alpha(l) \neq NULL \\ 0 & \text{sinon.} \end{cases}$$

Lorsque cette condition n'est plus vérifiée, le processus de résolution est interrompu et la clause apprise est ajoutée à l'ensemble des clauses.

Exemple 1.4 : Le processus d'apprentissage de nouvelle clause de l'exemple 1.3 est comme suit ;

$$\begin{cases} C_L^{3,0} = \{x_4, \overline{x_5}\} & i = 0 \rightarrow \alpha(\kappa) \\ C_L^{3,1} = \{x_3, x_4\} & \alpha(x_5) = C_3 \rightarrow C_L^{3,0} \odot C_3 \\ C_L^{3,2} = \{x_1, x_3\} & \alpha(x_3) = C_1 \rightarrow C_L^{3,1} \\ C_L^{3,3} = \{x_1, x_3\} & \text{plus de résolution} \end{cases}$$

La clause apprise et qui est à ajouter à l'ensemble des clauses est la suivante ; $C_{learnt} = x_1, x_3$. La majorité des solveurs complets sont inspirés

du DPLL ainsi que du CDCL Hamadi et al. [2008] Fu Biere et al. [2009]. Néanmoins, ces solveurs génèrent une explosion combinatoire due à la complexité et la taille des instances à résoudre. En effet, pour une instance comportant n variables, il existe 2^n combinaisons potentielles.

1.5.2 Solveurs incomplets

Les instances de problèmes étant de plus en plus complexes avec une taille considérable, les solveurs complets ne peuvent plus faire face à ces instances générant d'importants dépassements de temps de calcul et une explosion combinatoire. Ce qui a amené les chercheurs à développer de nouvelles approches de recherche de solutions incomplètes. Ces solveurs ne parcourent pas l'ensemble de l'espace de recherche, mais sont plutôt à la fois stochastiques et guidés par des heuristiques à la recherche de la solution la plus optimale. Ces solveurs sont majoritairement des métaheuristiques Glover et Kochenberger [2006].
Avant d'introduire ces solveurs, quelques notions de base sont à définir. Le **voisinage** d'une solution S_1 est défini comme l'ensemble de solutions générées par transformation ou permutation d'une valeur donnée dans la solution S_1. **Une Solution optimale** est une solution dont l'évaluation est la plus haute parmi un ensemble de solutions et par conséquent la plus proche du but. **L'évaluation** d'une solution apprécie le nombre de conditions ou d'attributs satisfaits par celle-ci. Par exemple, le nombre de clauses satisfaites par une assignation de variables donnée. **Un critère d'arrêt** définit une limite à l'exécution d'une fonction. Une limitation du temps de calcul ou d'une taille d'espace mémoire, une fixation du nombre maximal d'itérations à atteindre ou du nombre maximal de solutions ou encore la **stagnation** de la qualité des solutions représentent les critères d'arrêt les plus fréquemment utilisés. La stagnation est un état où la génération de nouvelles solutions n'améliore pas la qualité de solutions. Dans le meilleur des cas, la stagnation indique l'atteinte du but. Enfin, la **diversification** désigne le fait de changer de région dans le but d'explorer d'autres régions. Alors que **l'intensification** vise à explorer une zone donnée et à tendre vers l'optimum local.

Une des premières métaheuristiques à avoir été utilisée dans la résolution de problème est la recherche locale Hoos et Stützle [2004]. Elle fait partie des algorithmes de recherche basés sur la trajectoire et dont le principe est de manipuler une seule solution en l'améliorant à chaque itération, traçant ainsi une trajectoire dans l'espace de recherche vers une solution la plus optimale.
Le principe général de la recherche locale est de générer le voisinage d'une solution aléatoire dans un premier temps. Puis à chaque itération, sélectionner la meilleure solution parmi ce voisinage et générer son voisinage jusqu'à atteindre un certain critère d'arrêt.

Les deux algorithmes de recherche locale les plus utilisés dans la résolution du problème SAT sont le GSAT et le WALKSAT. Introduit en 1992 par B.Selman, H.Levesque et D.Mitchell, le GSAT Selman et al. [1992] est

l'un des premiers algorithmes de recherche locale à avoir fait ses preuves dans la résolution du problème SAT. Dans cet algorithme, la génération du voisinage d'une solution se fait de manière à minimiser le nombre de clauses insatisfaites. Ce qui revient à changer la valeur de vérité de la variable qui minimisera ce nombre jusqu'à atteindre la solution au problème ou que le nombre de changements maximal *MaxFlips* soit atteint. Ce processus est répété jusqu'à atteindre la solution au problème ou un nombre maximal d'essais *MaxTries*.

Algorithm 2 Algorithme Greedy SAT

Entrées E : Liste de clauses
MaxFlips : Nombre maximal de flips
MaxTries : Nombre maximal d'itérations
Debut
Répéter MaxTries fois
Assignation aléatoire des variables
Répéter Maxflips fois
Si l'assignation satisfait E alors SAT
sinon
Selectionner et changer la valeur d'une variables qui maximize le nombre de clauses satisfaites
Fin

Complexité : L'algorithme GSAT est structuré et présente deux grandes boucles. Une boucle interne majorait par *MaxFlips* et une boucle englobante majorée par *MaxTries*. De ce fait, la complexité de l'algorithme GSAT est $\theta(MaxTries * MaxFlips)$.
Contrairement au GSAT, l'algorithme WALKSAT Selman et al. [1993] considère lors de la génération du voisinage d'une solution, le changement de la valeur de vérité d'une variable de clauses non encore satisfaites. Trois types de changement sont définis ; le *freebie move* où la valeur d'une variable est changée si et seulement si le changement de cette dernière ne change pas l'état d'une clause satisfaite en insatisfaite. Le *random walk* dans lequel la variable est choisie au hasard. Et enfin, le *greedy move* qui sélectionne la variable qui minimise le nombre de clauses insatisfaites.
À chaque itération, l'algorithme recherche une variable permettant un *freebie move*. Lorsque celui-ci n'est pas possible, et avec une probabilité p, un *random walk* est effectué. Le *greedy move* est réalisé avec une probabilité $1 - p$. Comme pour GSAT, les paramètres *MaxTries* et *MaxFlips* contrôlent le nombre maximal d'itérations de l'algorithme. Divers solveurs sont inspirés du GSAT ainsi que du WALKSAT Hoos et al. [2002] KhudaBukhsh et al. [2016].

Une deuxième classe de métaheuristique utilisée dans la résolution de problèmes et notamment du problème SAT est la classe des méthodes orientées population. Ces algorithmes manipulent plusieurs solutions à la fois. Inspiré par différents opérateurs génétiques, l'algorithme génétique Davis [1991] est l'algorithme orienté population par excellence. Ces opérateurs génétiques sont le croisement et la mutation.
Avant de définir le principe de l'algorithme génétique, certaines notions

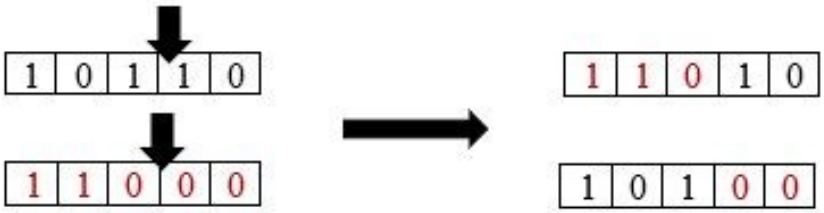
sont requises ;

— Une**population** regroupe un ensemble d'individus. La taille d'une population est représentée par le nombre d'individus.

— Un **individu** est représenté par une solution.

— **Un individu prometteur** est vu comme étant un individu susceptible de survivre dans la population et représente une bonne solution.

— Le **croisement** consiste à hybrider deux individus permettant d'en créer d'autres. Le croisement basique est défini par la permutation entre les deux parties de deux solutions distinctes par rapport à un point sélectionné comme suit ;



Plusieurs types de croisements peuvent être définis en multipliant le nombre de points de croisement.

— La **mutation** apporte une altération (modification) au niveau d'un ou de plusieurs points de mutation, comme suit ;

— Le **remplacement** permet d'éliminer les individus les moins prometteurs de la population, et par conséquent les solutions de moindre évaluation dans le but de maintenir la taille de la population.

Le principe de l'algorithme génétique est de générer une population de solutions aléatoires sur lesquels un certain nombre de croisements et de mutations est réalisé. Ces fonctions engendrent un accroissement de la taille de la population. Le processus de remplacement est alors exécuté, ne gardant que les individus les plus prometteurs. La meilleure solution est retournée à la fin du processus.

Algorithm 3 Algorithme Génétique

Entrées TC= Taux de croisement
TM=Taux de mutation
Taille de la population
Max-iteration
Début
Génération alétoire de la population initiale
Répéter tant que le nombre maximal d'itération n'est pas atteint et qu'il n'y a pas de stagnation
Croisement entre deux solution TC fois
Mutation d'une solution TM fois
Remplacement de la population
Retourner la meilleure solution
Fin

Complexité : L'algorithme génétique comprend trois principales fonctions. Une fonction de croisement répétée au maximum TC fois, une fonction de mutation répétée au maximum TM fois. Ainsi qu'une fonction de remplacement exécutée une seule fois par itération. L'algorithme génétique comprend au maximum MaxItération cycles. Par conséquent, la complexité de ce dernier est de l'ordre de $\theta(Max_iteration * (TC + TM))$.

Une autre classe de métaheuristiques largement utilisée dans la résolution de problèmes est celle des algorithmes bio-inspirés. Ces algorithmes tentent de reproduite certains phénomènes naturels et particulièrement le fonctionnement animal. En effet, certains animaux, bien que dépourvus de toute intelligence, présentent une certaine forme d'intelligence lorsqu'ils sont en groupe dite *Intelligence collective (en essaim)*. Cette forme d'intelligence s'observe particulièrement chez les insectes que l'on nomme *insectes sociaux* tels que les fourmis Dorigo et Birattari [2010], les abeilles Karaboga et Gorkemli [2019] et les termites Hedayatzadeh et al. [2010].
Parmi ces algorithmes, nous nous intéressons aux essaims d'abeilles et particulièrement à l'algorithme Bees Swarm Optimization (BSO) Drias et al. [2005] qui reproduit le fonctionnement des abeilles lors de la recherche de source de nourriture. Il a été démontré par Von Frisch et Chadwick [1967] qu'au moment de la recherche de nourriture, les abeilles indiquent la direction et le distance les séparant d'une source de nourriture en effectuant une danse en huit (8). Plus la source est proche, plus la danse est vigoureuse.
Par analogie à cette recherche de nourriture, la recherche de solution optimale se fait comme suit ; Une solution aléatoire nommée *Sref* est générée par une abeille BeeInit. À partir de cette solution, une zone de recherche *SearchArea* est déterminée et chacune de ces solutions est affectée à une abeille. Chaque abeille effectue une *recherche locale* et retourne, dans une table nommée *Danse*, la meilleure solution du voisinage de la solution qui lui a été octroyée. La meilleure solution de cette table est considérée comme *Sref* et le processus de recherche reprend jusqu'à ce que la solution au problème soit trouvée ou qu'une condition d'arrêt soit atteinte.

Algorithm 4 Algorithme Bees Swarm Optimization

Entrées TL : Liste taboue
Table Danse
Sref : Solution de référence
Bees : Liste d'abeilles
Debut
Génération d'une solution aléatoire initiale $Sref$
Tant que la condition d'arrêt n'est pas atteinte
Mettre $Sref$ dans la TL
Générer une zone de recherche à partir de $Sref$
Assigner une solution à chaque abeille
Effectuer une recherche par chaque abeiller et retourner la meilleure solution dans la table danse
Prendre la meilleure solution de la table danse comme $Sref$
Retourner la meilleure solution de la TL
Fin

Complexité : La complexité de l'algorithme BSO est de (MaxIterBSO*NbrBees*MaxIterSLS). En effet, l'algorithme BSO s'exécute en $MaxIterBSO$ cycles où chaque cycle comporte une recherche locale effectuée par chacune des abeilles de la colonie. Cette recherche locale est majorée à $MaxIterSLS$.

Il existe d'autres catégories de solveurs SAT dont les solveurs parallèles sont les plus utilisés. Ces derniers combinent différents algorithmes avec différentes configurations dans le but d'obtenir la meilleure solution possible. On distingue les algorithmes *Portfolio* tels que *ManySAT* Hamadi et al. [2008] qui permet l'utilisation de quatre configurations différentes du CDCL dans un même solveur. Ces différents algorithmes communiquent entre eux via une architecture multicœur permettant un gain de temps considérable en partageant les clauses apprises après conflit.
Certains solveurs tels que *SATenstein* Hoos et al. [2002] Cai et al. [2015] combinent des algorithmes incomplets.

1.6 Conclusion

A travers ce premier chapitre, nous avons introduit le problème de Satisfiabilité booléenne ainsi que l'intérêt qui lui est porté par la communauté de l'intelligence artificielle.
étant le premier problème à être démontré NP-complet, de nombreuses recherches ont été entreprises dans le but de résoudre ce dernier pour l'impact qu'aurait sa résolution sur l'ensemble des problèmes NP-complets.
Nous avons présenté les algorithmes et solveurs SAT les plus populaires et les plus utilisés dans la résolution du problème SAT. Deux principales catégories ont été définies. La classe des algorithmes complets qui garantit de trouver la bonne solution si celle-ci existe ou prouve sa non-existence dans le cas échéant. L'algorithme central de cette classe est le DPLL. La seconde catégorie est celle des solveurs incomplets. Ces algorithmes sont approximatifs et tentent de trouver la meilleure solution possible au pro-

blème sans garantie d'en trouver la bonne. Le GSAT, dont s'inspirent de nombreux solveurs, est l'un des algorithmes incomplets les plus utilisés.
L'ensemble de ces algorithmes peuvent s'agencer entre eux formant des dizaines d'algorithmes et solveurs tant bien séquentiels que parallèles.
Le but de ce travail de thèse est de contribuer à la résolution des problèmes NP-complets et particulièrement du problème SAT en explorant les zones les plus prometteuses de l'espace de recherche. Et ce en utilisant des méthodes intelligentes d'exploration de données et de datamining.

Techniques de datamining 2

2.1 Introduction

L'exploration des données est l'une des sciences les plus anciennement utilisées. Cette science s'est vue évoluer au fil du temps passant de la simple analyse au datamining. Le datamining regroupe un ensemble d'outils et de fonctions dont le principal but est d'extraire des connaissances à partir de simples données.
Dans ce chapitre, nous allons commencer par une introduction la connaissance de nos données. Puis nous définirons ce qu'est le datamining et ses principales étapes. Nous évoquerons les plus importantes méthodes de datamining et de clustering particulièrement.

2.2 Connaître ses données

Avant tout prétraitement sur une donnée, un minimum de connaissances de celle-ci est nécessaire. Cette étape passe par la définition du type de données Han et al. [2011] ainsi que l'étude de certains paramètres statistiques.

2.2.1 Types de données

Une donnée ou encore un objet est défini par l'ensemble d'attributs qui le composent. Le type de cet attribut est déterminé par les valeurs que celui-ci peut prendre.

Les données qualitatives

Le type de données qualitatives regroupe un ensemble de types où les données sont plutôt décrites que mesurées. En effet, la valeur de ces attributs ne reflète pas la quantité de ces derniers, mais leur qualité. Parmi les types de données qualitatives, on retrouve le type nominal, binaire et ordinal.

— Dans le type nominal, les valeurs que peut prendre un attribut peuvent être des noms, des symboles ou autres. Ces valeurs font références à la catégorie à laquelle fait partie un attribut ou une donnée sans qu'il n'y ait d'ordre.
Ces valeurs peuvent être remplacées par des codifications numériques ou alphanumériques tout en étant considérées comme

nominales et non numériques.

— Le type binaire regroupe les attributs de type nominal dont les deux seules valeurs possibles sont le 0 et le 1. Conventionnellement, 0 désigne l'absence de l'attribut en question contrairement à la valeur 1 indique la présence de celui-ci.
On parle d'attribut binaire symétrique si les valeurs que peut prendre un attribut sont pondérées de façon égale. à l'opposé, un attribut binaire asymétrique désigne un attribut dont une valeur peut être plus importante ou avec un impact plus significatif par rapport à l'autre.

— Les attributs de type ordinal présentent une certaine hiérarchie. En effet, ces attributs prennent des valeurs qui peuvent être classées ou ordonnées.

Les données quantitatives

à l'opposé des données de type qualitatif, le type quantitatif comme son nom l'indique permet de quantifier l'attribut en question.
Le type numérique représente le type qualitatif par excellence. La valeur que peut prendre un attribut peut être unique ou encore être représentée par un intervalle de valeurs.

2.2.2 Etude statistique des données

Dans le but de mieux connaitre nos données et de déterminer le prétraitement le plus adéquat à ces dernières, une étude statistique est indispensable. Cette étude regroupe les paramètres et fonctions suivantes Han et al. [2011] :

Caractéristiques de tendance centrale

Les mesures de tendance centrale permettent comme le nom l'indique de déterminer les valeurs centrales autour desquelles les données ont tendance à se rassembler. On retrouve parmi ces paramètres :

— **La moyenne** qui représente la mesure la plus courante des paramètres de tendance centrale. Elle exprime la moyenne des données, et par définition la donnée autour de laquelle les données affluent. Elle est calculée en utilisant la formule suivante :

$$\overline{x_i} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.1}$$

— **La médiane** représente un meilleur indicateur de tendance centrale que la moyenne en divisant les données en deux parties de même nombre. Elle se calcule en utilisant la formule suivante :

$$Me = \begin{cases} X_{N/2} & \text{N pair} \\ X_{[N/2]+1} & \text{N impair} \end{cases}$$

En pratique, la médiane est déterminée comme l'élément central des données ordonnées et répétées autant de fois que leurs occurrences.

— **Le mode** représente la donnée ayant le plus haut effectif dans une série. Dans certains cas, le plus haut effectif peut correspondre à plusieurs données différentes. On parle alors de données multimodales.

Caractéristiques de dispersion

Les paramètres de dispersion permettent de mesurer et d'apprécier la répartition des données. Ils peuvent être définis par la distance entre deux paramètres de distributions comme pour l'écart interquartile, ou encore, être calculés par rapport à une valeur centrale tel que l'écart type. On définit comme principaux paramètres de dispersion :

— **L'étendue** qui représente l'ensemble des valeurs que peut prendre une variable. Elle est calculée par différence entre la valeur minimale et maximale de l'ensemble des données comme suit :

$$E = max(x_i) - min(x_i) \tag{2.2}$$

— **L'écart interquartile (IQR)** est l'un des principaux indicateurs de la dispersion. Il représente la différence entre le troisième et le premier quartile $(Q_3 - Q_1)$ et regroupe la moitié des effectifs de l'ensemble des données.

— **L'écart type** noté σ est défini par la racine de **la variance**. C'est le paramètre de dispersion le plus utilisé en statistique. Il définit la distribution des variables par rapport à la moyenne. Plus la valeur de la variance est petite, plus les valeurs sont proches les unes des autres, et inversement. L'écart type est calculé par la formule suivante :

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{k} n_i(x_i - \overline{x})^2} \tag{2.3}$$

σ^2 étant la **variance**.
Noter que 68% des données sont reparties entre $\overline{x} - \sigma$ et $\overline{x} + \sigma$, 95% entre $\overline{x} - 2\sigma$ et $\overline{x} + 2\sigma$, 99% entre $\overline{x} - 3\sigma$ et $\overline{x} + 3\sigma$.

2.2.3 Calcul de similarité entre les données

La similarité est l'une des notions les plus importantes lorsque l'on parle de traitement de données et particulièrement de datamining et de clustering. Elle permet une mesure de la distance ou du degré de proximité entre deux voire plusieurs données de même type. Sa valeur varie entre 0 et 1, 0 désigne le fait que les deux objets soient complètement différents alors que 1 indique que les deux objets sont identiques.
La similarité est calculée en fonction du type de données traitées. En effet,

une ou plusieurs fonctions de mesure de similarité sont associées à chaque type Han et al. [2011].

Données nominales

La similarité entre deux objets nominaux est définie par l'équation suivante :

$$S(i,j) = \frac{m}{p} \tag{2.4}$$

Où m représente le nombre d'attributs identiques dans i et j, alors que p désigne le nombre d'attributs décrivant les deux objets.

Données numériques

Les données numériques considèrent plus la notion de distance que celle de similitude. Plusieurs formules et fonctions de calcul de distance existent. Parmi lesquelles nous citons la distance *Euclidienne*, la distance de *Manhattan* et la distance de *Minkowski*. Considérons les deux objets suivants i $(x_{i1}, x_i2, \ldots, x_in)$ et j $(x_{j1}, x_{j2}, \ldots, x_{jn})$. Ces différentes équations sont définies comme suit :

$$d_{Euclidienne}(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + \cdots + (x_{in} - x_{jn})^2} \tag{2.5}$$

$$d_{Manhattan}(i,j) = |x_{i1} - x_{j1}| + \cdots + |x_{in} - x_{jn}| \tag{2.6}$$

$$d_{Minkowski}(i,j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + \cdots + |x_{in} - x_{jn}|^h} \tag{2.7}$$

avec $h > 1$

données binaires

Plusieurs fonctions de calcul de similarité et dissimilarité ont été proposées pour les données de type binaire. Nous avons sélectionné et comparé quelques-unes de ces fonctions Choi et al. [2010] dans le but de déterminer celles qui étaient les plus adaptées au calcul de similarité entre les clauses SAT. En effet, les variables SAT sont de type binaire. Ces fonctions sont les suivantes ;

$$S_{Intersection} = a \tag{2.8}$$

$$S_{Innerproduct} = a + d \tag{2.9}$$

$$S_{Jaccard} = \frac{a}{a+b+c} \tag{2.10}$$

$$S_{Johnson} = \frac{a}{a+b} + \frac{a}{a+c} \tag{2.11}$$

$$S_{BtaunBanquet} = \frac{a}{Max(a+b, a+c)} \tag{2.12}$$

$$S_{Simpson} = \frac{a}{Min(a+b, a+c)} \tag{2.13}$$

$$D_{Hamming} = b + c \tag{2.14}$$

Les six premières fonctions représentent des fonctions de calcul de similarité alors que la dernière mesure la distance. Les éléments a,b,c et d sont définis comme dans le tableau suivant (tableau 2.1) :

	$i = 1$ (Present)	$i = 0$ (Absent)
$j = 1$ (Present)	a	b
$j = 0$ (Absent)	c	d

TABLE 2.1 – *Table de vérité*

La valeur a (respectivement d) indique le nombre d'attributs présents (respectivement absents) dans les deux objets i et j. b et c désignent quant à eux le nombre d'attributs présents dans un objet et pas dans l'autre et inversement.

Nous retenons les deux principales fonctions de *Jaccard* et *Hamming* pour la suite du travail.

2.3 Définition datamining

Le Datamining désigne un ensemble d'outils et d'algorithmes qui permettent l'exploration et l'analyse d'un ensemble important de données. Le but de ce processus est d'extraire des informations et des connaissances à priori inconnues, des associations ainsi que des corrélations entre les données permettant une meilleure prise de décision Han et al. [2011]Stéphane [2012]].

Ce processus définit plusieurs étapes, parmi lesquelles, le prétraitement, le clustering et la classification sont les plus importantes. Le prétraitement a pour objectif de détecter et corriger les données bruitées, manquantes ainsi que les aberrations. Les techniques d'apprentissage non supervisées ou encore clustering visent à regrouper les données similaires dans une même catégorie, alors que les méthodes d'apprentissage supervisées (classification) permettent de classifier un élément dans une classe donnée selon ses attributs.

2.4 Prétraitement des données

Du fait de la multiplicité des sources de données et de l'importante taille de ces dernières, une étape de prétraitement est primordiale. En effet, ces données peuvent présenter des inconsistances ou encore être bruitées. Le prétraitement est donc une étape primordiale de vérification et de préparation des données. Il est composé des étapes suivantes selon Han et al. [2011].

- Le nettoyage des données consiste à détecter et corriger dans la mesure du possible les données bruitées.
- L'intégration des données permet la combinaison des données provenant de sources multiples.
- La réduction des données permet une sélection des données utiles aux différents traitements futurs.
- La transformation des données ou encore normalisation des données dont le but est de satisfaire une certaine norme de valeurs.

Remarque : Toutes les étapes du prétraitement de données ne sont pas obligatoires devant des données bien structurées. Il est toutefois recommandé de faire le prétraitement devant des données provenant de plusieurs sources et des données complexes.

2.5 Techniques de datamining

Les techniques sont organisées en deux grandes catégories, l'apprentissage non supervisé ou encore clustering et l'apprentissage supervisé ou classification.

2.5.1 Clustering

Le clustering Han et al. [2011] Stéphane [2012] désigne le fait de regrouper des données dans différents clusters en fonction de leur similarité. Le but est alors d'obtenir des clusters où les éléments d'un même cluster sont les plus similaires possible et les éléments de clusters différents sont les plus distincts possible. Il existe plusieurs outils et algorithmes de clustering regroupés comme suit :

Les méthodes de partitionnement

Le partitionnement est l'une des méthodes les plus simples et basiques en clustering. Basés sur la distance entre les différents éléments, les algorithmes de partitionnement visent à diviser un ensemble de données en k groupes distincts d'au moins un élément chacun. Ces méthodes itératives tentent d'améliorer le partitionnement en changeant les centres de clusters à chaque itération.
Le **k-means** Lloyd [1982] est l'un des principaux algorithmes de partitionnement. Son principe est de déterminer initialement et aléatoirement k centres de clusters dits *centroïds*. Chaque élément est ensuite assigné au cluster le plus proche en fonction de sa distance avec le *centroïd*. A chaque itération et dans le but d'améliorer le clustering, les centroïds sont recalculés en fonction de la moyenne de chaque cluster. Ce processus d'assignation – recalcul de *centroïds* est poursuivi jusqu'à stagnation du clustering.
Néanmoins, le **k-means** est sensible aux valeurs aberrantes du fait de la sélection initiale des *centroïds*. Plusieurs variantes ont donc été proposées pour pallier à cet inconvénient.
Le **k-means++** Arthur et Vassilvitskii [2007] propose une alternative au choix initial des *centroïds*. Seul le premier centroïd est choisi aléatoirement. L'algorithme procède ensuite au calcul de la distance entre chaque élément et le *centroïd* le plus proche pour sélectionner les $k-1$ *centroïds* restants. Une probabilité proportionnelle à cette distance au carré est utilisée pour le choix du nouveau *centroïd*. Cette opération est exécutée jusqu'à l'obtention des k différents *centroïds* initiaux. Le reste de la procédure est identique à celle du **k-means**. Une autre version améliorée du k-means est **Fuzzy k-means** Bezdek et al. [1999]. Il propose le même fonctionnement que le **k-means** à la différence que chaque élément peut appartenir à plus d'un cluster. L'algorithme commence par initialiser k *centroïds* et à

chaque itération et jusqu'à convergence du clustering, calcul pour chaque pair objet-centroïd la probabilité d'appartenance de l'objet au cluster. Les nouveaux *centroïds* sont calculés comme étant la moyenne pondérée des éléments du cluster.
Le **GA-k-means** Drias et al. [2013] représente l'une des améliorations les plus efficaces et prometteuses du ¸means. L'étape de sélection aléatoire des centroïds est remplacée par l'exécution de l'algorithme génétique et par conséquent l'obtention des *centroïds* les plus optimaux possible.

Outre les variantes directes du k-means, **le Partitioning Around Medoids (PAM)** Stéphane [2012] Kaufman et Rousseeuw [2009] est un algorithme de partitionnement qui présente le même fonctionnement que le k-means. Contrairement au k-means, le PAM considère comme centre de cluster les *medoïds* situés exactement au centre du cluster. A chaque itération, pour un cluster donné, la moyenne de dissimilarité du *medoïd* de ce cluster ainsi que celle d'un élément non-medoïd pris au hasard sont calculées. Cette moyenne est définie par la distance moyenne entre un élément donné et l'ensemble des éléments du cluster auquel il appartient ou doit appartenir. L'élément ayant la plus petite moyenne de dissimilarité est retenu comme nouveau *medoïd* du cluster.
Contrairement au k-means qui ne peut être utilisé que pour des données numériques, PAM peut être utilisé pour différents types de données. PAM reste un algorithme de clustering plus efficace que k-means mais moins efficient.

L'un des algorithmes PAM les plus populaires et les plus efficaces est le **Clustering Large Applications based on RANdomized Search (CLARANS)** Ng et Han [1994]. Il consiste en l'exécution successive de l'algorithme PAM, suivant une stratégie de *Hill Climbing*. Ainsi, chaque exécution du PAM représente un *optimum local. L'optimum global* définit le meilleur clustering.

Algorithm 5 Algorithme CLARANS

Entrées k : Nombre de clusters.
DS : Base contenants n objets.
MaxLocal : Nombre maximal de processus PAM.
MaxAttempts : Nombre maximal de changement de Medoïds.
LR : Liste de medoïds.
LC : Ensemble de clusters.
Début
Répéter
Selectionner aléatoirement k objects (Medoïds)
Répéter
Assigner tous les objets de DS au cluster le plus proche
Choisir un élément medoïd et un non médoid et calculer la moyenne de dissimilarité comme suit ($E = \sum_{(} i = 1)^k \sum_{(} p \in C_i) dist(p, c_i)^2$)
Si l'élément non médoid améliore la moyenne de dissimilarité il devient medoid
Refaire le clustering des éléments
Jusqu'à MaxAttempts
Minimum local atteint
Jusqu'à MaxLocal
Selectionner le meilleur minimum local pour être minimum global = Meilleur clustering
Fin

Les méthodes hiérarchiques

Les techniques de clustering hiérarchique Davidson et Ravi [2005] sont des méthodes de décompositions successives de données selon la distance ou la densité. Deux types de méthodes hiérarchiques sont à définir :

— Les méthodes hiérarchiques ascendantes ou encore agglomératives qui partent d'un état où chaque cluster ne contient qu'un et un seul objet. Ces clusters sont fusionnés deux à deux à chaque itération selon la distance séparant les éléments de ces derniers, jusqu'à ce qu'il n'y ait qu'un seul cluster ou jusqu'à satisfaction d'une certaine condition.

— Les méthodes hiérarchiques descendantes ou encore divisives partent d'un état où tous les éléments appartiennent au même cluster. Ce cluster est scindé en deux parties à chaque itération jusqu'à n'avoir qu'un et un seul élément par cluster. La satisfaction d'une configuration donnée relative au nombre ou à la distance entre les données peut également être un critère d'arrêt.

L'inconvénient de ces méthodes est qu'on ne peut pas revenir en arrière, et par conséquent une décision prise à un niveau l ne peut pas être défaite ou modifiée à un niveau $l + 1$. Une alternative est d'intégrer une méthode de classification au sein du clustering hiérarchique comme dans les algorithmes **BIRCH** Zhang et al. [1996], **CHAMELEON** Karypis et al. [1999] et dans le **clustering hiérarchique probabiliste** Han et al. [2011].

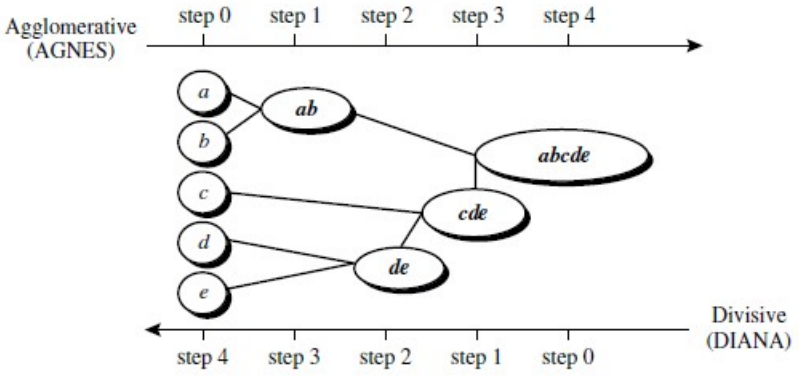


FIGURE 2.1 – *Méthodes hièrarchiques de clustering*

L'algorithme BIRCH pour **Balanced Iterative Reducing and Clustering using Hierarchies** est une méthode de clustering hiérarchique adaptée pour les grandes bases de données. Deux principaux concepts définissent cet algorithme, le Clustering feature (CF) et le Clustering feature tree ($CF-Tree$).
Le CF comporte les informations relatives au cluster et est représenté par un vecteur tridimensionnel regroupant le nombre de données dans le cluster, la somme de ces données ainsi que la somme des carrés de ces dernières $CF =< N, LS, SS >$. Ces CFs présentent la propriété d'additivité qui consiste à additionner les paramètres de deux CFs différents pour n'en créer qu'un seul comme suit $CF3 = CF1 + CF2 =< N_1 + N_2, LS_1 + LS_2, SS_1 + SS_2 >$
Le $CF-Tree$ quant à lui représente un dendrogramme ou arbre équilibré contenant les CFs du clustering hiérarchique. Les nœuds non-feuilles de cet arbre contiennent la somme des CFs de leurs enfants. Deux paramètres caractérisent le $CF-tree$, le $Branchingfactor$ (B) qui limite le nombre de CF par nœud, et le $Threshold$ (T) qui détermine le seuil que le *diamètre* ou le rayon des nœuds feuilles ne doit pas dépasser. Le *diamètre* et le rayon sont des paramètres de clustering définis comme la distance moyenne par paire dans un cluster pour le *diamètre*, et la distance moyenne des données par rapport au centre de cluster pour le rayon.
L'algorithme **BIRCH** s'exécute en deux phases :

- Première étape : Scanner les données et construire le *CF-Tree*.
- Deuxième étape : Appliquer un algorithme de clustering dans le but de classer les noeuds feuilles et éliminer le bruit.

Les méthodes basées sur la densité

Contrairement aux méthodes présentées précédemment, les techniques basées sur la densité ne considèrent pas les notions de distance et de similarité, mais la notion de voisinage et de densité dans la création de clusters. Les données sont alors considérées dans un espace où chaque région de forte densité définit un cluster. Une région est dite dense si et seulement si le nombre d'objets contenus dans celle-ci est supérieur à un certain seuil minimal donné.

Proposé par Martin Ester et al. Ester et al. [1996], l'algorithme **DBSCAN (Density Based Spatial Clustering of Application with Noise)** est l'algorithme de clustering basé sur la densité le plus populaire et le plus utilisé. Il commence par considérer aléatoirement un point p comme point central d'une éventuelle région dense. A partir de ce point et en considérant un rayon r, une région avoisinante à p est déterminée. Si la densité de cette région définie par le nombre d'objets qu'elle contient est supérieure ou égale au seuil $MinPts$ prédéfini, la région est dite dense et est considérée comme cluster. Une fois le cluster créé, tous les points inclus dans ce dernier seront considérés comme centre d'une région qui sera évaluée. Si la densité de cette région est supérieure ou égale à $MinPts$, les points de cette région seront additionnés au cluster. Ce processus est répété jusqu'à ce qu'il n'y ait plus de régions denses. Les éléments n'appartenant à aucun cluster sont considérés comme outliers.

Algorithm 6 Algorithme DBSCAN

Données LP : Liste de points
LN : Liste de voisins neighbours
LNo : Liste de points aberrants (noise/ outlier)
LR : Liste des regions denses of regions
Début
Choisir aléatoirement un point p de l'ensemble LP
si le nombre d'élément du voisinage de p (dans un rayon r) est supérieur à $MinPts$ alors
Créer un cluster contenant ces éléments
Pour tout élément de ce cluster
Déterminer le voisinage et l'ajouter au cluster avec la même condition ($MinPts$)
Sinon Désigner ce point comme aberrant
si la liste LP n'est pas vide, reprendre le processus.
Fin

Les méthodes basées grille

Le clustering basé sur les grilles est une méthode guidée par l'espace, contrairement aux méthodes de clustering introduites précédemment et qui sont guidées par les données (*Data driven algorithms et Space driven algorithms*) Han et al. [2011].
Ces méthodes considèrent une grille formée de plusieurs cellules où les données sont projetées et subdivisées en cluster en fonction de ces cellules. La totale indépendance du nombre de données et la totale dépendance au nombre de cellules permettent un clustering plus efficient. Néanmoins, certaines de ces méthodes utilisent la notion de densité lors de la création de clusters.
Les deux algorithmes de clustering basés sur les grilles sont le **CLustering In QUEst (CLIQUE)** et le **STatistical Information Grid (STING)**.
L'algorithme CLIQUE Agrawal et al. [1998] reprend le même fonctionnement que l'algorithme Apriori Agrawal et al. [1994], en effectuant des jointures récursives entre les différentes cellules.

STING, proposé par Wei Wang et al. Wang et al. [1997] est un algorithme de clustering basé sur les grilles. Il considère une grille multicouche hiérarchique dans laquelle les données sont représentées par des paramètres dépendent des attributs des données tels que la valeur minimale et maximale, la valeur moyenne, l'écart type, la distribution des données, et un paramètre indépendant des attributs et représentant le nombre d'objet dans la cellule.
L'algorithme STING s'exécute en deux étapes. La première étape consiste en la création d'une structure hiérarchique composée d'un nœud racine contenant tous les objets. Chaque nœud excepté les feuilles possède quatre enfants correspondant à un quadrant de la cellule mère. Le processus de fractionnement s'arrête lorsqu'une certaine granularité (nombre d'objets par cellule) est atteinte. Dans la seconde phase, une recherche descendante dans la structure précédemment construite est effectuée dans le but de déterminer les cellules satisfaisant les requêtes du clustering. La figure 2.2 illustre l'architecture hiérarchique de STING.

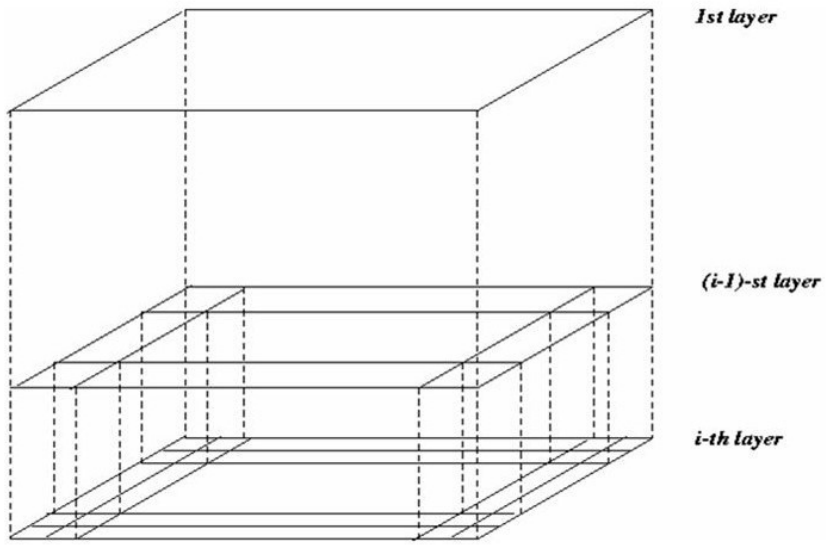


FIGURE 2.2 – *Architecture hiérarchique multicouche STING*

2.5.2 Classification

La classification ou encore la prédiction consiste à réaliser un apprentissage sur un ensemble de données dont on connait au préalable les classes de sorties, dans le but de pouvoir classer de nouvelles données Stéphane [2012]. Les méthodes de classification comportent alors deux phases. La phase d'apprentissage qui a pour objectif d'extraire les caractéristiques communes aux différents éléments d'une même classe dans le but d'établir une base de règles permettant la classification de nouvelles données. La deuxième étape consiste à classer de nouvelles données conformément aux règles précédemment apprises.

Les arbres de décisions

Les arbres de décision représentent l'un des outils majeurs dans l'aide à la décision. Un arbre de décision est par définition un diagramme orienté dont les nœuds non-feuilles représentent les attributs ou les variables de décision. Les arcs sortants de ces nœuds représentent la valeur de ces variables. Les nœuds feuilles représentent la classe de sortie de l'objet dont les valeurs d'attributs correspondent au chemin allant de la racine à la feuille Han et al. [2011]. À partir d'un arbre de décision, un ensemble de règles peut être extrait en convertissant ces différents chemins comme suit :

$$If(attribut1 = val1 and attribut2 = val2 \ldots) then Classe = C_x \tag{2.15}$$

Ces règles sont utilisées dans la classification de nouvelle instance de données.
Plusieurs algorithmes de création ou d'induction de ces arbres existent tels que $ID3$, $CART$, $C4.5$ Han et al. [2011]. La différence entre ces algorithmes réside dans l'ordre d'apparition des attributs et donc de la profondeur de l'arbre.

Classifieurs bayésiens naïfs

Le principe de ces classifieurs consiste en un calcul probabiliste conditionnel de l'appartenance d'une donnée à une certaine classe en utilisant le théorème de Bayes. A chaque nouvelle donnée non classée, une probabilité d'appartenance à chaque classe est calculée et la donnée est assignée à la classe qui maximise cette probabilité.
Rappel du Théorème de Bayés :
Soit : $P(A)$ et $P(B)$, respectivement, les probabilités des événements A et B. Soit la probabilité conditionnelle

$$P(A/B) = \frac{P(A,B)}{P(B)} \tag{2.16}$$

Le théorème de Bayés stipule que :

$$P(A/B) = \frac{P(B/A) * P(A)}{P(B)} \tag{2.17}$$

Généralisation :

$$P(A/B_1, \ldots, B_n) = \frac{P(B1, \ldots, Bn/A) * P(A)}{P(B1, \ldots, Bn)} \tag{2.18}$$

Si l'on considère l'indépendance entre les événements B_i et B_j ,$i \neq j$ on aura

$$P(A/B_1, \ldots, B_n) = \prod_n^1 [P(A/B_i)] = \prod_n^1 \frac{P(B_i/A) * P(A)}{P(B_i)} \tag{2.19}$$

Les réseaux de neurones

Inspirés des réseaux de neurones biologiques, les réseaux de neurones artificiels représentent un réseau dans lequel chaque élément fonctionne parallèlement aux autres et produit un résultat en fonction des entrées qui lui ont été transmises.
En biologie, un neurone reçoit un stimulus nerveux externe ou provenant d'un autre neurone par le biais de neurotransmetteur. Ce dernier permet l'excitation du neurone et l'induction d'un influx nerveux qui sera transmis par l'axone de ce dernier. Par analogie, un Perceptron ou encore neurone artificiel représente un élément ou un processus qui reçoit des variables d'entrées externes ou provenant d'autres neurones. Ces variables sont sommées par le perceptron produisant une sortie transmise à un autre neurone ou une sortie finale désignant la classe à laquelle appartient l'élément traité.

2.5.3 Les règles d'associtations

Une autre fonction du datamining est l'extraction de modèles fréquents ou règles d'associations. Le but de ces méthodes est de découvrir et d'extraire les relations et corrélations existantes entre les données.
Introduit par Agrawal Agrawal et al. [1994], l'algorithme Apriori représente l'algorithme le plus utilisé pour extraire les modèles fréquents. Un modèle est dit fréquent si et seulement si son support et sa *confidence* satisfont les seuils prédéfinis.
Le *Support* représente la probabilité d'apparition de deux *items* A et B dans une même transaction et est défini comme suit

$$S(A \Rightarrow B) = P(A \cap B) \tag{2.20}$$

Alors que la Confidence désigne la probabilité que l'item B appartienne aux transactions qui contiennent A et est définie par la probabilité suivante

$$C(A \Rightarrow B) = P(B|A). \tag{2.21}$$

En pratique, un *itemset* est dit fréquent si et seulement si sa fréquence d'apparition est supérieure ou égale au seuil prédéfini $MinSup$.
L'algorithme Apriori commence par extraire les items fréquents unitaires. Et pour les items de taille $k \geq 2$, effectue une jointure entre les items de taille $k-1$ créant ainsi des candidats à évaluer. Le support de chaque candidat est alors évalué dans le but de déterminer si ce dernier est fréquent ou non.
La propriété stipulant que « Tous les sous-items d'un itemset fréquent doivent également être fréquents » doit être respectée lors de l'évaluation des k-candidats. A titre d'exemple, l'itemset I_1, I_2, I_3 ne peut être considéré comme fréquent que si les itemsetsI_1, I_2,I_1, I_3,I_2, I_3 sont fréquents en plus de I_1, I_2, I_3.

2.6 Conclusion

L'exploration des données et le datamining représentent l'une des sciences les plus utilisées dans divers domaines et notamment dans la prise de décision. En effet, les techniques du datamining permettent l'extraction et l'apprentissage de connaissances ainsi que des corrélations inconnues au préalable et par conséquent, une meilleure utilisation de la donnée et de l'information.
Dans le cadre de ce projet, nous proposerons d'exploiter ces techniques dans la résolution de problèmes complexes.

Parallélisme sur CPU-GPU 3

3.1 Introduction

L'architecture des ordinateurs et particulièrement celle de leurs processeurs a vu une évolution spectaculaire au fil des années les années. En effet, avec l'explosion du nombre de données et d'informations à traiter, une augmentation des capacités de calculs de ces machines est nécessaire. Parallèlement à l'évolution des CPU (Central Processing Unit), on retrouve l'avènement du GPU (Graphic Processing Unit) qui propose une architecture assez différente de celle du CPU. Cette architecture peut être assimilée à un multiprocesseur avec une forte capacité de parallélisation de données et de calcul.
Nous aborderons dans ce chapitre cette évolution des processeurs et particulièrement l'avènement des calculateurs graphiques (GPU) ainsi qu'un aperçu de la programmation parallèle.

3.2 Histoire du GPU – Passage du CPU au GPU

L'histoire des microprocesseurs remonte aux années 70 avec l'introduction, par Intel, du premier microprocesseur 4bits à être commercialisé (Intel 4004) avec une fréquence d'horloge équivalente à 740kHz. Dès lors, et pour satisfaire les besoins de plus en plus importants des utilisateurs, ces microprocesseurs ont connu une grande évolution.
L'un des premiers paramètres à avoir été amélioré dans le but d'augmenter les performances des processeurs est la fréquence de l'horloge. La fréquence de l'horloge est l'un des principaux paramètres mis en jeu dans le calcul du temps d'exécution d'un programme. En effet, le temps d'exécution d'un programme se calcule comme suit :
Runtime=Nbr Instructions ∗ Nbr cycles par Instruction ∗ Temps d' exécution d'un cycle
Où le nombre de cycles par instruction est fonction de l'architecture du processeur ainsi que du programme lui-même. Le temps d'exécution d'un cycle est inversement proportionnel à la fréquence de l'horloge en Hertz. Cependant, pour des raisons de difficultés de dissipation thermique, l'amélioration de cette fréquence d'horloge a été freinée.
L'industrie de l'informatique est donc passée au multiprocessing. Un multiprocesseur est un processeur possédant plusieurs cœurs fonctionnant simultanément. Ce type d'architecture a permis d'augmenter la puissance

de calcul des ordinateurs sans pour autant augmenter la fréquence d'horloge et par conséquent a permis de réduire la production thermique. Le premier modèle de multiprocesseur à deux cœurs (dual core) à avoir vu le jour fut le Power4 de IBM en 2001, suivi par la commercialisation du premier ordinateur portable avec un multiprocesseur dual core en 2005 par Intel.
De nos jours, et avec l'évolution de l'industrie des processeurs, nous disposons d'ordinateur portable avec un multiprocesseur à huit (08) cœurs et une fréquence d'horloge à 4.90 Ghz (Intel i9 9eme génération).

Parallèlement à l'évolution des CPU (Central Processing Unit), l'avènement du GPU (Graphic Processing Unit) propose une architecture assez différente de celle du CPU.
L'architecture GPU peut être assimilée à un multiprocesseur avec une forte capacité de parallélisation des données et du calcul. Ces cartes graphiques n'ont été développées, initialement, que pour décharger les microprocesseurs des calculs graphiques. Néanmoins, et malgré l'architecture multicœur des CPU qui fournissent une puissance de calcul élevée, celle-ci est limitée par l'important temps de latence relatif au transfert des données entre la mémoire et le microprocesseur. L'utilisation des GPU dans les traitements et calculs est devenue une réalité incontournable.

La figure 3.1 Corporation [2012], illustre l'évolution des performances de calcul des processeurs CPU ainsi que GPU(*NVIDIA*) en nombre d'opérations en virgule flottante par seconde (FLOPS).

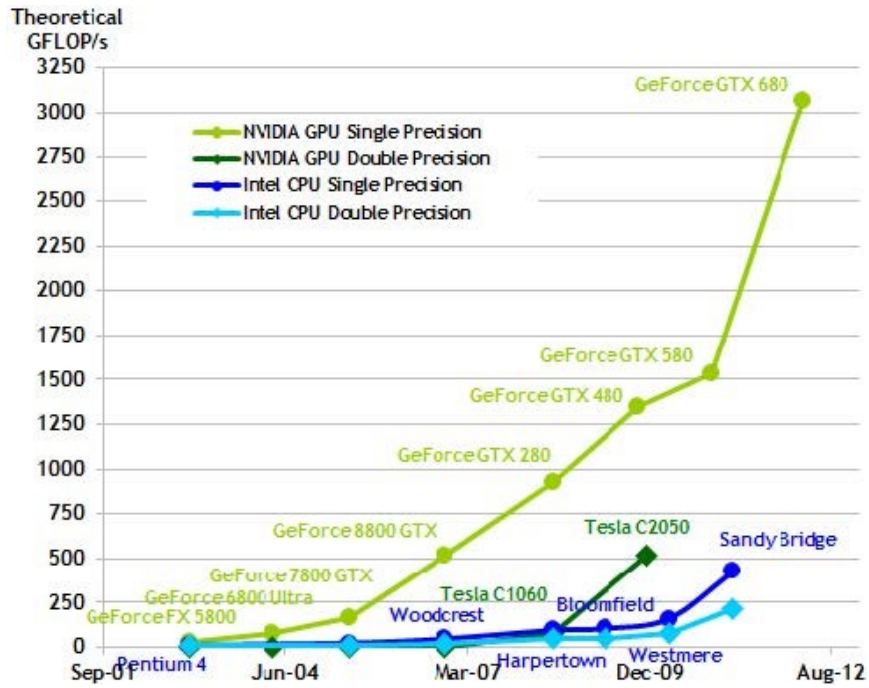


FIGURE 3.1 – *Évolution des performances de calcul du CPU et GPU Corporation [2012]*

Cette augmentation quasi exponentielle de la puissance des GPU est due à l'architecture des GPUs offrant un traitement parallèle intensif

conséquence de la spécialisation et de la gestion des transistors. En effet, l'architecture GPU favorise davantage le nombre de transistors alloués au calcul plutôt qu'à la gestion et le stockage du flux de données Corporation [2012].
L'architecture GPU ainsi qu'une comparaison avec l'architecture CPU est proposée dans la figure suivante (3.2)Corporation [2012].

FIGURE 3.2 – *Architecture CPU et GPU Kirk et Wen-Mei [2016]*

Contrairement au CPU, le GPU présente la particularité que son architecture soit adaptée au parallélisme avec la capacité d'exécuter simultanément un même programme sur une multitude de données. Cette propriété, outre le fait qu'elle permette une parallélisation de l'exécution, nécessite un moindre controle du flux de données. La latence d'accès à la mémoire est alors dite masquée par le calcul parallèle intensif.

3.3 Architecture du GPU

Un GPU moderne est vu comme une grille ou un ensemble de streaming multiprocesseur (SM) où chaque SM est représenté par un certain nombre de streaming processeur (SP) ainsi que d'une mémoire globale partagée DRAM (Dynamic Random Access Memory) avec une bande passante élevée.
Chacun de ses processeurs SM est vu comme une machine SIMD (Single Instruction Multiple Data), pouvant par définition exécuter la même instruction sur différentes données simultanément. En résumé, le GPU est considéré comme une multitude de machines SIMD.
Les figures suivantes (3.3 et 3.4) présentent une schématisation de l'architecture d'un GPU et celle d'une unité SM.

3.3.1 Architecture CUDA

Parallèlement à cette architecture, l'architecture *CUDA (Computer Unified device Architecture)*, introduite en 2006, définit le GPU par une hiérarchisation de structures. En effet, un GPU est considéré comme un ensemble de grilles, où chaque grille est représentée par un ensemble de blocs exécutant le même kernel. Un bloc étant un ensemble de threads coopérant et fonctionnant ensemble. Chaque bloc possède un identifiant relatif à la grille à laquelle il est associé, et peut accéder à la mémoire

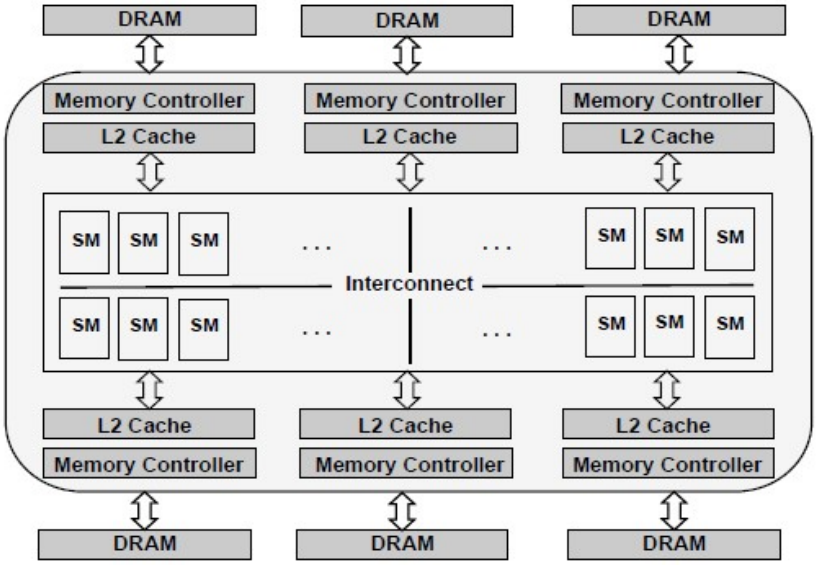


FIGURE 3.3 – *Architecture GPUArora [2012]*

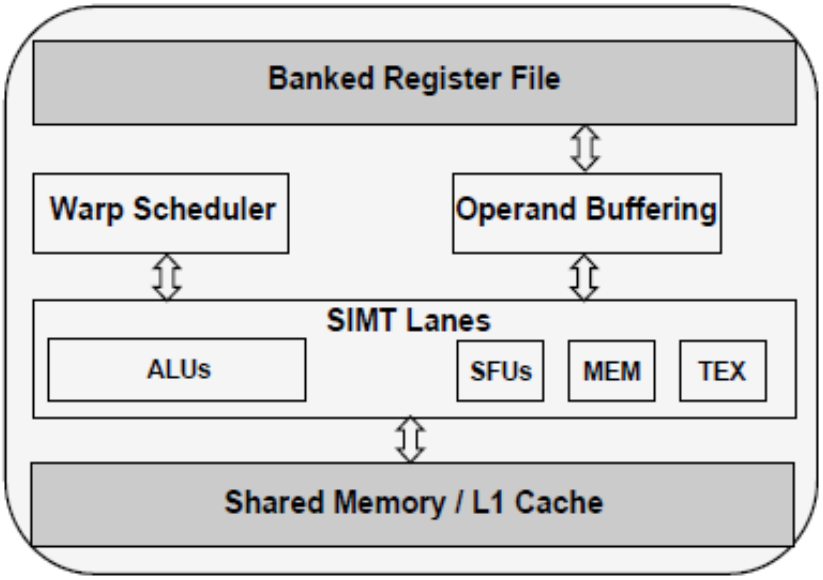


FIGURE 3.4 – *Architecture SMArora [2012]*

globale du GPU. Les threads représentent l'entité élémentaire d'un GPU. Les threads d'un même bloc possèdent une mémoire partagée et ont la capacité d'être synchrones. Un ensemble de 32 threads est appelé *warp.*
Plusieurs architectures CUDA ont été proposées, la première NVIDIA est l'architecture *Tesla* (2006) avec 128 cœurs de calcul (SP) répartis sur 16SM (8 SPs par SM) pour le GeForce 8800 GTX. Cette architecture a été suivie par d'autres, plus innovantes et performantes. On cite dans l'ordre de leurs apparitions, les architectures *Fermi* (2010), *Kepler* (2012), *Maxwell* (2014), *Pascal* (2016), *Volta* (2017), pour finir avec l'architecture *Turing* (2018). Cette dernière (TU102) comporte 6 GPC (Graphics Processing

Clusters) avec 36 Texture Processing Clusters (TPC), 72 SM avec 64SP par SM.
Cette nouvelle architecture adopte une nouvelle architecture des SM qui induit une amélioration de près de 50% des performances en comparaison avec l'architecture *Pascal* Turing relative à la gestion des mémoires et de l'augmentation de la bande passante et de la capacité de calcul pour le cache L1.

3.4 CUDA : Programmation et notion de mémoire

La puissance de calcul offerte par les cartes graphiques a poussé les constructeurs de ces dernières à développer des cartes graphiques programmables, telles que les cartes CUDA de *NVIDIA*. Le GPGPU (General Purpose computation on GPU) Fernando [2004] a donc vu le jour avec comme principal objectif d'améliorer et d'accélérer les calculs jusqu'à plus de 100%. La technologie du GPGPU a permis à de nombreux constructeurs de proposer des langages et des plateformes de développement. Parmi lesquels, nous citerons et utiliserons au terme de ce travail de thèse le langage CUDA.
CUDA, pour Computer Unified Device Architecture Kirk et Wen-Mei [2016], est un langage de programmation qui permet au développement de programme exécutable sur GPU. Introduit par *NVIDIA* en 2007, le langage CUDA présente une extension du langage C, dont les grandes lignes sont les suivantes ;
Un programme CPU/GPU CUDA considère l'existence d'un *host* (CPU) et d'un ou de multiples *devices* représentant le ou les GPUs d'un ordinateur. Le corps de ce programme est scindé en deux grandes parties. Dans la première on retrouve les fonctions séquentielles exécutées sur CPU dont le mot clé y référant est *__host__*. Ces fonctions ne peuvent être appelées que par le CPU. Dans la partie parallèle du programme, on retrouve des fonctions nommées *Kernel* qui sont exécutées sur GPU. Il existe deux types de kernels : celles appelées exclusivement par le GPU et ayant pour mot référant *__device__*, et celles pouvant être appelées par le CPU et/ou le GPU ayant pour mot référant *__global__*.
Chaque kernel du programme doit disposer des variables suivantes dites implicites : $blockIdx$, $blockDim$ et $threadIdx$ indiquant respectivement l'indice, la taille d'un bloc dans une grille et l'indice du thread dans le bloc. Ces variables permettant d'adresser la fonction à un ou plusieurs threads dans un bloc.
La syntaxe d'appel d'un kernel est alors comme suit :
$NomKernel <<< \#blocs, \#threads >>> (paramtres)$
Il est à noter que le CPU et le GPU possèdent des mémoires séparées, et par conséquent l'exécution d'une fonction sur le GPU nécessite un transfert de données du CPU vers le GPU. De plus, les kernels ne retournant aucune valeur (void), les différents résultats de calcul doivent être transmis du GPU vers le CPU.
La gestion de ce flux de données entre CPU et GPU est exécutée par la fonction *cudaMemcpy* dont les 4 paramètres sont : la destination de la donnée, le pointeur ou l'adresse mémoire de la donnée à copier, le

nombre d'octets à copier et le type de transfert à effectuer. Ce dernier est défini par les mots clés *cudaMemcpyHostToDevice* dans le cas de transfert de données du CPU au GPU, et *cudaMemcpyDeviceToHost* dans le cas de récupération de résultats de GPU vers CPU.
A titre d'exemple, lors de la récupération de résultats de calcul parallèle :
$cudaMemcpy(PtrCPU, PtrGPU, sizeof(VariableType), cudaMemcpyDeviceToHost)$
Une allocation préalable de toutes les variables nécessaires sur CPU est GPU est requise. L'allocation de mémoire GPU se fait de la même manière que pour le langage C avec la fonction cudaMalloc en y associant les paramètres relatifs à l'adresse mémoire et à la taille désirée comme suit :
$cudaMalloc((void**)PtrGPU, sizeof(VariableType))$.
La figure 3.5 illustre grossièrement l'appel de fonction dans le GPU ainsi que le fonctionnement et l'organisation des threads dans les différents blocs.

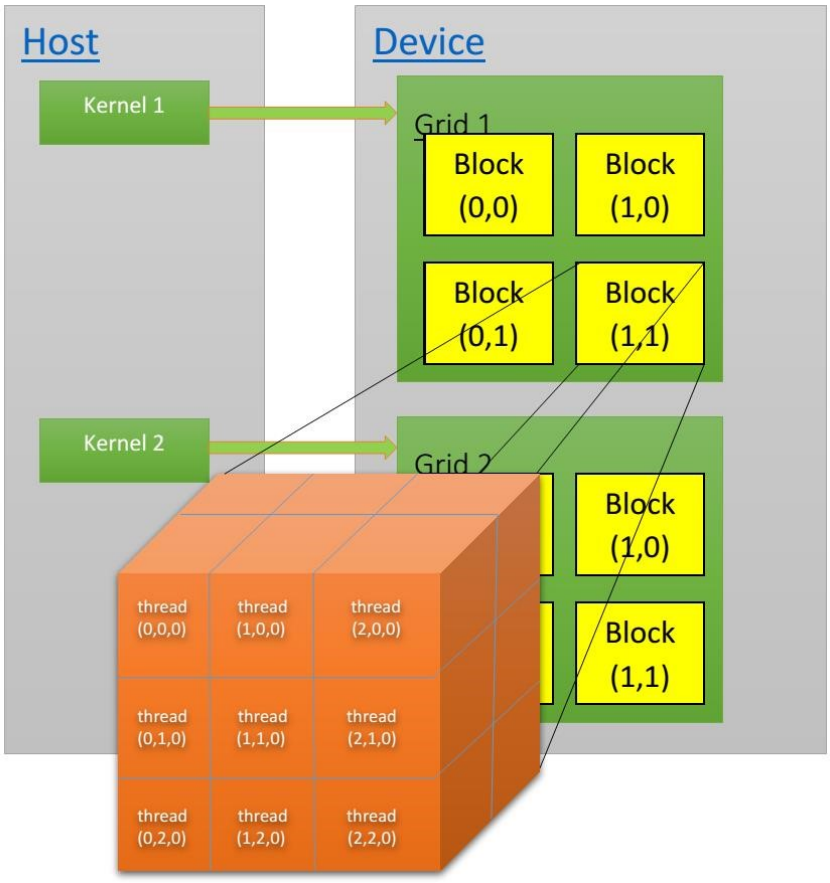


FIGURE 3.5 – *Organisation des threads et appel de fonction Kirk et Wen-Mei [2016]*

Il est à noter que l'architecture CUDA supporte différents types de mémoire. Kirk et Wen-Mei [2016].
En premier lieu, nous trouvons la mémoire globale ainsi que la mémoire constante, de type lecture/écriture qui permettent le transfert de données du CPU au GPU et vice versa.
On retrouve dans chaque bloc, une mémoire partagée par les threads ainsi que des registres propres à chaque thread. Ces mémoires sont à

latence courte et sont à lecture seule lorsque plusieurs threads accèdent à la même adresse mémoire. Les registres sont utilisés pour les variables les plus fréquemment utilisées. Une schématisation et hiérarchisation de ces mémoires est proposée dans la figure 3.6 Kirk et Wen-Mei [2016].

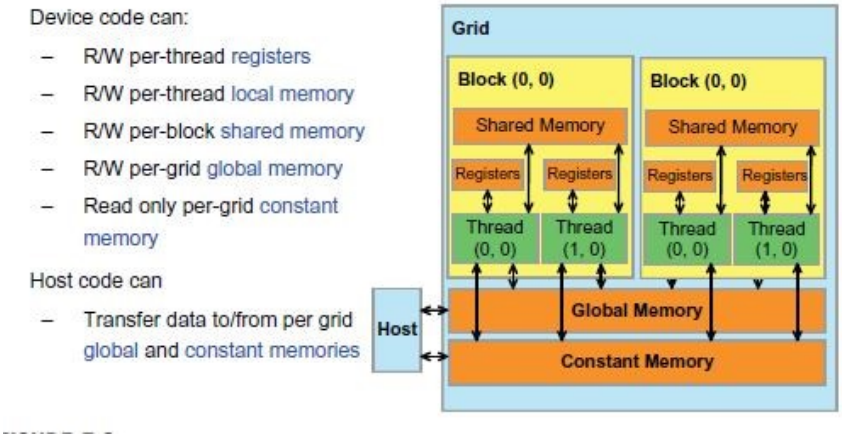


FIGURE 3.6 – *Architecture et organisation des mémoires – CUDA Kirk et Wen-Mei [2016]*

Des mots clés ou extensions CUDA existent pour indiquer au GPU la mémoire dans laquelle une donnée est stockée. Ces mots clés sont *device*, *constant* et *shared* et indiquent respectivement le stockage d'une variable dans la mémoire globale visible par tous les threads, dans la mémoire constante et par conséquent accessible à tous les threads en lecture seule, et enfin le stockage de celle-ci dans la mémoire partagée d'un même bloc et visible qu'aux threads de ce même bloc.

3.5 Conclusion

Dans ce chapitre, nous avons introduit quelques notions de parallélisme et notamment celles offertes par le calcul GPU. En effet, outre l'utilisation de ces cartes dans le monde des jeux et de la réalité virtuelle pour les résolutions graphiques qu'elles offrent, ces dernières sont utilisées pour optimiser le calcul et augmenter les performances d'un ordinateur.
En 2007, *NVIDIA* introduit un langage de programmation CUDA permettant d'implémenter et de développer des programmes exécutables sur GPU. Ce langage se définit comme une extension du langage C auquel des librairies et des fonctions propres au GPU sont ajoutées.
Cette nouvelle technologie permet un gain considérable de temps grâce au parallélisme massif des données. Néanmoins, la gestion des mémoires reste un point crucial. En effet, l'architecture de ces GPU définit différents niveaux de mémoire partagée entre tous les éléments du GPU ou entre les threads du même bloc. La communication entre le CPU et les éléments du GPU se faisant via ces mémoires, une gestion de ce flux doit être la plus optimale possible pour permettre d'avoir de bonnes performances.

En conclusion, nous pouvons affirmer de l'importance et de l'impact que peut avoir l'utilisation du GPU sur un grand nombre de données à condition que les flux de données soient bien gérés.

Etude exploratoire des instances SAT

4

4.1 Introduction

Le clustering est l'une des techniques de datamining les plus utilisées dans le but de déterminer des ensembles de données en fonction de leur similitude. Il existe de nombreuses techniques de datamining pouvant fournir des résultats différents pour le même jeu de données. Déterminer la technique de clustering la plus adéquate à notre ensemble de données et alors primordiale.
Nous proposons, à travers ce chapitre, une étude statistique et exploratoire des instances SAT et particulièrement l'étude de la dispersion des variables sur les clauses.
Cette étude a permis de révéler deux principaux modèles de distribution. Le premier modèle présente un espace où les variables forment des zones de forte densité parsemées de zones moins denses voire vides. La seconde distribution est aléatoire et ne présente aucune région de densité particulière. À chacune de ces distributions est associée une méthode de clustering jugée plus appropriée.

4.2 Modèles de distribution et types de clustering

Choisir le type de clustering le plus approprié pour un jeu de données ou encore une instance de problème est l'une des tâches les plus primordiales à accomplir.
Dans le but de déterminer le clustering le plus adéquat à une instance SAT, l'étude de la distribution des variables sur les clauses est une étape nécessaire.
Rappelons qu'une instance de problème SAT peut être définie par une ou matrice de contingence n*m définie comme suit, n étant le nombre de variables et m le nombre de clauses :

$$\begin{cases} C[i,j] = 0 & \text{Lorsque la variable i est absente de la clause j} \\ C[i,j] = 1 & \text{Lorsque le litéral i est présent dans la clause j} \\ C[i,j] = -1 & \text{Lorsque le litéral } \bar{i} \text{ est présent dans la clause j} \end{cases}$$

La distribution des variables sur les clauses résulte de la projection des clauses sur un plan bidimensionnel de la même manière que sur une

matrice de contingence. L'axe des abscisses représente les variables et l'axe des ordonnées les clauses. Une clause C_i est alors représentée par n points $p_1(x_{i1}, i)$,$p_2(x_{i2}, i)$,...,$p_n(x_{in}, i)$, où chaque point représente une variable de la clause i.

Exemple 4.1 : Soit l'instance SAT suivante (exemple 1.1 chapitre 1) :
$C_1 = x_1, x_2$
$C_2 = \overline{x_1}, x_2, x3$
$C_3 = x_1, x_2, \overline{x_4}$

La figure 4.1 représente la distribution des variables sur les clauses de cet exemple. La clause C_1 est représentée par les points $p_1(1, 1)$ et $p_2(2, 1)$.

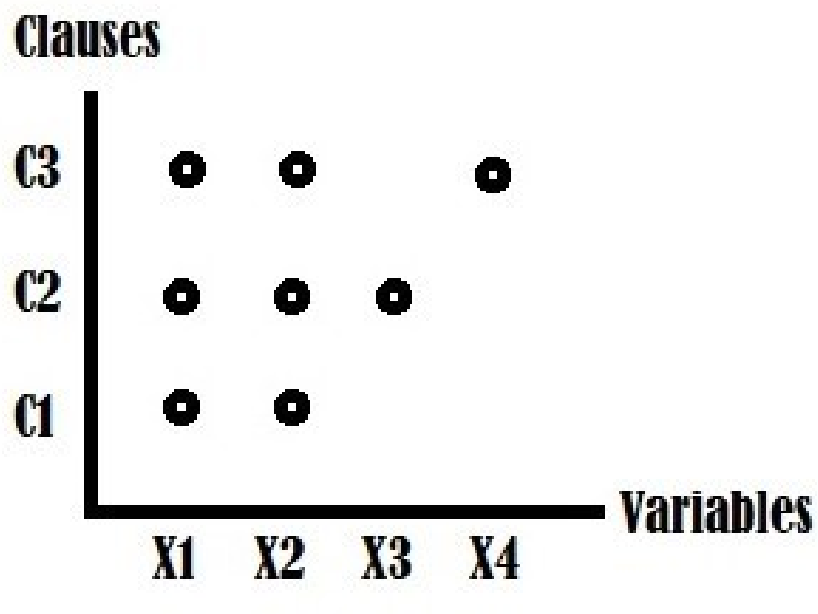
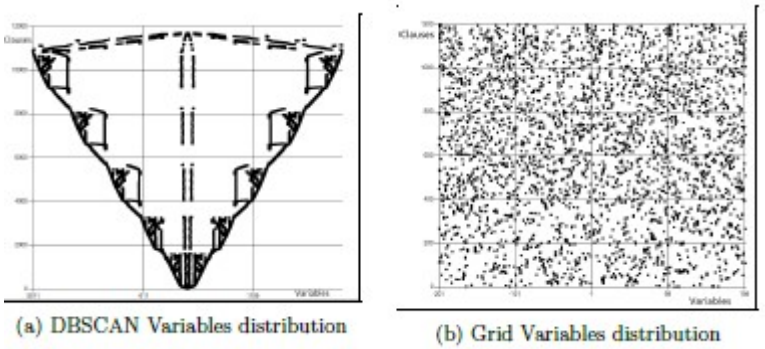


FIGURE 4.1 – *Dispersion des variables sur les clauses (exemple 4.1)*

Après étude de la dispersion des variables sur les clauses de diverses instances de problème, nous retenons et présentons les deux principaux modèles de distributions révélés. Le premier modèle où la distribution des variables permet de distinguer des zones où la densité des variables est plus importante. L'espace est parsemé de régions à faible densité de variables voire des régions à densité nulle.
Dans le second modèle, la dispersion des variables est plus aléatoire et ne présente aucune région de densité particulière. Ce modèle de distribution présente une certaine particularité dans certaines instances où un grand nombre de variables présentent une importante occurrence.
La figure 4.2 introduit ces deux modèles de distribution.

La figure 4.2 illustre les deux principaux modèles de dispersion des variables sur les clauses. À partir de ces modèles, la technique de clustering la plus appropriée à chaque modèle est déterminée. En effet, la première distribution nous fait directement penser au clustering basé sur la densité et plus particulièrement à l'algorithme DBSCAN. Cet algorithme a pour principal objectif de détecter et d'extraire les zones de forte densité ce qui



(a) DBSCAN Variables distribution

(b) Grid Variables distribution

FIGURE 4.2 – *Dispersion des variables sur les clauses (exemple 4.1)*

pourrait offrir de bien meilleurs résultats que d'autres méthodes de clustering pour les instances ayant ce genre de distribution.
Le deuxième type de distribution nous fait penser au clustering basé sur les grilles. En effet, cet espace où les variables sont distribuées par rapport aux clauses peut être vu comme une grille où chaque cellule de celle-ci forme un cluster.
Par ailleurs, certaines instances dont la distribution correspond au second modèle possèdent la particularité d'avoir un nombre important de variables de haute fréquence. Cette particularité augmente la probabilité d'appartenance de ces variables aux même clauses. Les algorithmes d'extraction de modèles fréquents semblent être adaptés à ce type de données.

4.3 Prétraitement des instances de problème SAT

Nous avons vu précédemment que le prétraitement des données est une étape primordiale au clustering ou à tout autre type de traitement sur ces données qui peuvent contenir des inconstances et incohérences. Cependant, les instances SAT représentent en grande partie des problèmes réels traduits en instances SAT dans le but de faciliter leur résolution. Nous ne pouvons donc pas parler d'inconsistance ou d'incohérence. Notre proposition de prétraitement pour les instances SAT consiste alors à satisfaire les deux conditions introduites dans le premier chapitre à savoir ;

- La suppression des tautologies. Ces clauses contiennent à la fois une variable et sa négation et sont toujours satisfaites quelques soit l'instanciation des variables. Leur suppression n'influe pas sur l'assignation des variables et l'intégrité des clauses.
- L'élimination des variables redondantes. Les clauses ne doivent compter qu'une et une seule occurrence d'un même littéral.

Outre ces conditions à satisfaire, et dans le but de réduire la complexité des instances SAT, les clauses incluses dans d'autres clauses sont éliminées. Nous définissons qu'une clause C_i est incluse dans une clause C_j si et seulement si le nombre de variables communes entre C_i et C_j est égal au nombre de variables qui forment la clause C_i. La clause C_j est éliminée dans ce cas, car la satisfaction de la clause C_i induit la satisfaction de C_j.

Exemple 4.2 :
Soient les clauses C_i et C_j suivantes :
$C_i = x_1, \overline{x_2}$
$C_j = x_1, \overline{x_2}, x_3$.
On remarque que quelques soit la valeur que prend la variable x_3, si la clause C_i est satisfaite alors la clause C_j l'est aussi. Dans ce cas, la suppression de la clause C_j permet la réduction de la complexité de l'instance et un gain de temps lors de la résolution.

4.4 Étude de cas : Validation du prétraitement proposé et modèle de distribution

Dans cette section, nous allons déterminer l'efficacité du prétraitement proposé en démontrant que celui-ci permet une réduction de la complexité des instances SAT sans compromettre leurs intégrités.
Trois études de cas sont réalisées dans le but de définir les différents modèles de distributions révélés.

4.4.1 Présentations des benchmarks et instances SAT étudiées

Pour déterminer et prouver l'efficacité et l'efficience des différentes approches proposées dans cette thèse, différentes expérimentations ont été effectuées sur des instances SAT connues et reconnues.
Le tableau ci-après (4.1) expose le nombre de variables et nombre de clauses des différentes instances ainsi que le nombre de clauses supprimées lors du prétraitement proposé.

Benchmark	# variables	# clauses	# tautologies	# clauses incluses
BMC-IBM1	9685	55870	15	1173
BMC-IBM2	2810	11683	16	1106
BMC-IBM7	8710	39774	28	2358
BMC-IBM13	13215	65728	0	2605
AIM-200	200	1200	3	15
FLA-500	500	2205	0	0
Unif-k3	5000	21335	0	0
Unif-k5	230	4857	0	0

Table 4.1 – *Instances de problème SAT étudiées*

d

Les instances BMC sont issues de Biere et al. [1999]encoded BMC., FLA-500 ainsi que Unif-k3 et Unif-k5 proviennent de SAT. et AIM-200 de Random..
Le nombre de tautologies ainsi que celui des clauses incluses indiquent le nombre de clauses éliminées de l'instance.

4.4.2 Validation du prétraitement

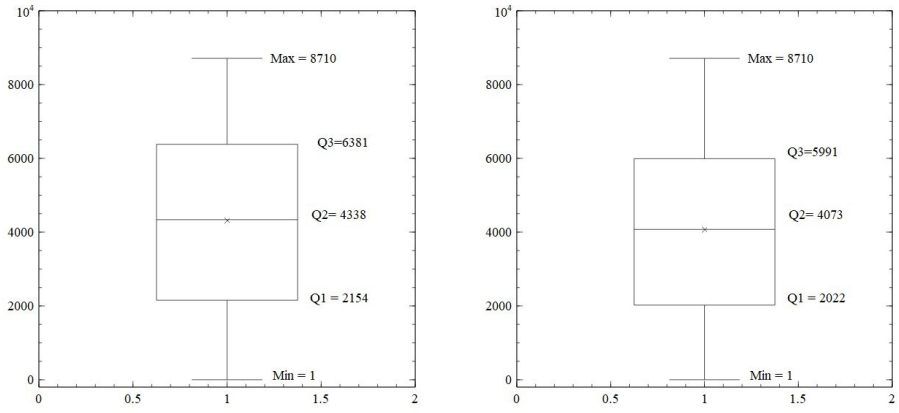
Afin de valider le prétraitement présenté précédemment, nous proposons de vérifier l'intégrité des instances de problèmes obtenues après

prétraitement. Dans un premier temps, une étude statistique comparative entre les instances SAT avant et après prétraitement est effectuée. La seconde phase de validation consiste à comparer la qualité d'une solution générée aléatoirement sur l'instance avant et après prétraitement.
La première étape de cette validation est présentée sur l'instance BMC-IBM7.
Le tableau 4.2 et la figure 4.3 représentent une comparaison des plus importants paramètres statistiques permettant la validation du prétraitement.

Benchmark	Avant nettoyage	Après nettoyage
	IBM7	
# clauses	39774	37388
# variables	8710	8710
Valeur Min	X_1	X_1
Valeur Max	X_{8710}	X_{8710}
# variables manquantes	80	80
Valeur moyenne	X_{4314} $(+X_{4331})$ $(-X_{4298})$	X_{4068} $(+X_{4140})$ $(-X_{3997})$
Quartiles	$X = X_{2154}, X_{4338}, X_{6381}$ $(+)X_{2114}, X_{4337}, X_{6364}$ $(-)X_{2193}, X_{4343}, X_{6402}$	$X = X_{2022}, X_{4073}, X_{5991}$ $(+)X_{2069}, X_{4233}, X_{6086}$ $(-)X_{1992}, X_{3969}, X_{5909}$
Mode	$X = X_{480}, X_{1545}, X_{2845}$ X_{4326}, X_{5807} Accuracy = 352	$X = X_{480}, X_{1545}, X_{2845}$ X_{4326}, X_{5807} Accuracy = 348
Moyenne d'occurrence	11	10
Variables symmétriques	803	865

TABLE 4.2 – *Benchmark IBM7 : Statistiques avant et après nettoyage*

Remarques : Les signes (+) et (-) précédant les paramètres moyenne, mode et autres indiquent ces paramètres selon la polarité des variables. À titre d'exemple $(+X_{4331})$ indique la moyenne des variables sous forme positive seulement.
Le terme variables symétriques désigne les variables dont l'occurrence sous forme positive x_i est identique à celle sous forme négative $\overline{x_i}$.



(a) *Avant prétraitement* (b) *Après prétraitement*

FIGURE 4.3 – *Boite à moustache pour l'instance IBM7 avant et après le nettoyage*

Le tableau 4.2 ainsi que la figure 4.3 exposent les caractéristiques statiques les plus importantes du benchmark IBM7 avant et après prétraitement.

On remarque une symétrie dans la dispersion des variables dans l'instance. En effet, la médiane qui divise les variables en deux ensembles d'occurrences égales est quasi identique à la variable moyenne. Et ce dans les deux instances avant et après prétraitement.
L'IQR qui représente l'un des meilleurs indicateurs de dispersion est de 4250 avant prétraitement et 3696 après, ce qui reste relativement identique avec 2386 clauses supprimées. L'instance étudiée est multimodale avec cinq différentes variables ayant le plus grand effectif.

La dispersion et la tendance générale des variables sur les deux instances avant et après prétraitement sont similaires indiquant un maintien de l'intégrité des instances.

Afin d'appuyer ce résultat et de prouver l'intégrité de ces instances, une solution aléatoire a été générée pour chaque instance. Le tableau 4.3 illustre la comparaison de la qualité de ces solutions pour les instances avant et après prétraitement.

Benchmark	Initial		Preprocessing		Difference
	# clauses	% SAT	# clauses	% SAT	
BMC-IBM1	55870	77.03	54682	76.97	0.05
BMC-IBM2	11683	78.87	10561	78.82	0.05
BMC-IBM7	39774	77.30	37388	77.22	0.08
BMC-IBM13	65728	78.28	63123	78.26	0.02
AIM-200	1200	86.58	1182	86.63	-0.05

TABLE 4.3 – *Taux de satisfaction avant et après prétraitement*

Remarque : Les instances FLA500, Unif-k3 et Unif-k5 ne sont pas présentées dans ce tableau car leur prétraitement n'a généré aucune élimination de clauses.

Les résultats exhibés par le tableau 4.3 indiquent une forte similitude entre les taux de satisfaction des deux instances avant et après preprocessing. Ce qui nous permet, avec les résultats précédents de conclure que le prétraitement proposé n'engendre pas de perte de l'intégrité des instances.

4.4.3 Étude de cas : Modèle de dispersion 1 : BMC-IBM7

Les figures suivantes introduisent les paramètres statistiques les plus importants pour l'étude de l'instance.

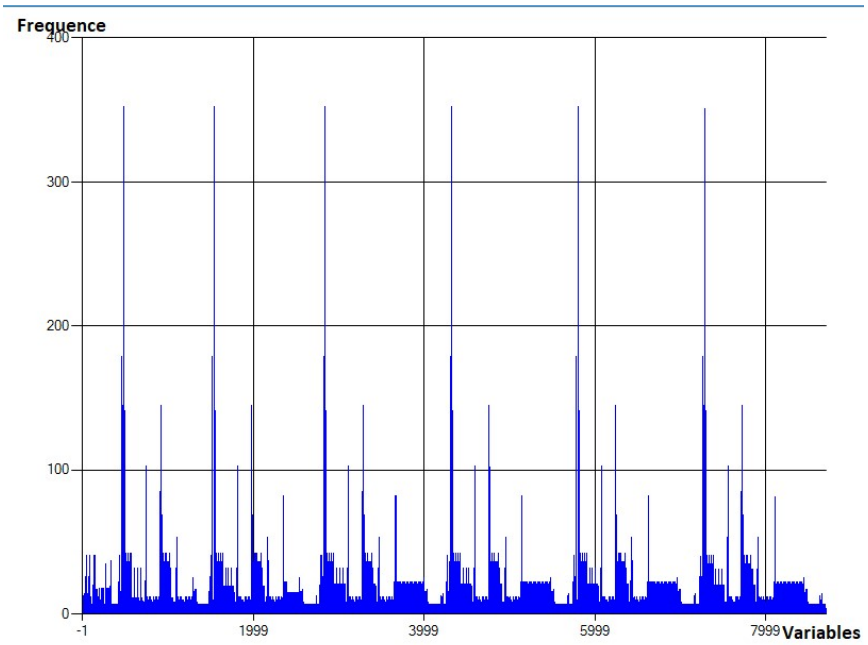


FIGURE 4.4 – *Fréquence d'apparition des variables*

La figure 4.4 présente l'histogramme des fréquences d'apparition des variables dans l'instance IBM7. On note la présence de 52 fréquences différentes pour un ensemble de 8630 variables. Ce qui indique la présence d'un nombre important de variables de même fréquence.
Le nuage de points des fréquences d'apparition des variables (figure 4.5) illustre la fréquence des variables sous les différentes formes positive et négative.

Dans cette figure, les points rouges (respectivement noirs) représentent la fréquence des variables sous forme positive (respectivement négative). On remarque une certaine superposition de ces points, indiquant l'existence de variables dont la fréquence sous forme positive est égale à celle sous forme négative. On parle alors de variables équilibrées. Le nombre de ces variables est de 803.

La figure 4.6 introduit la distribution des variables sur les clauses pour l'instance BMC-IBM7.

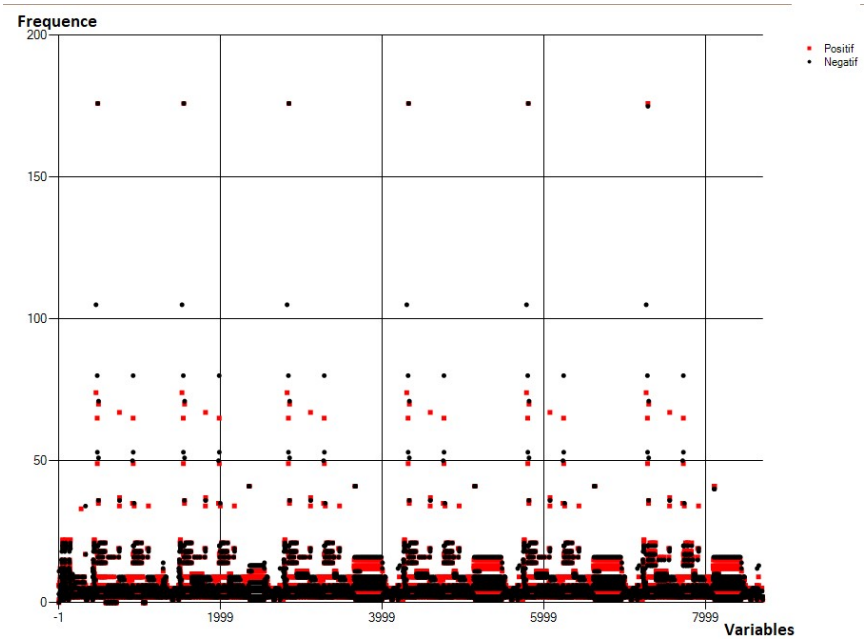


FIGURE 4.5 – *Nuages de points des fréquences d'apparition des variables*

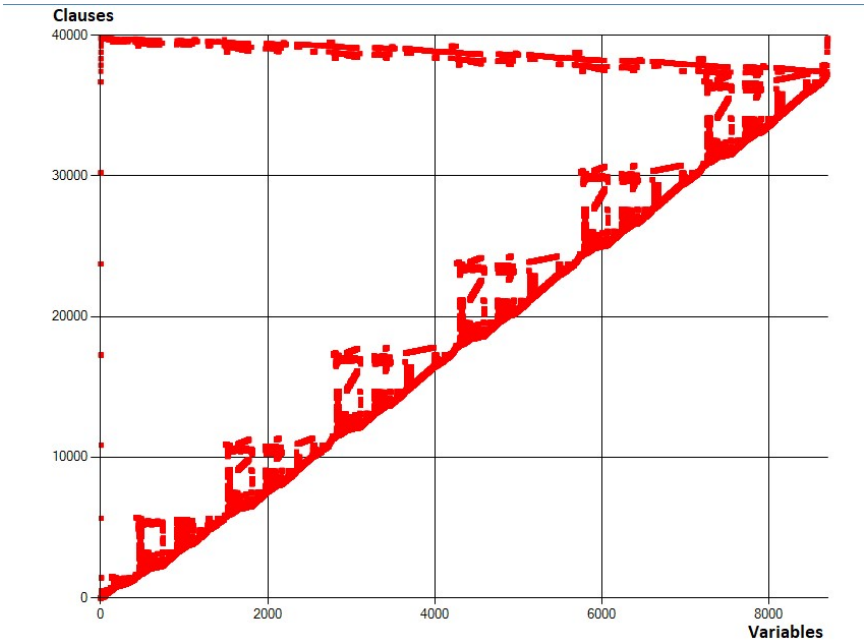


FIGURE 4.6 – *Distribution des variables sur les clauses pour l'instance BMC-IBM7*

La distribution des variables sur les clauses illustrée par la figure 4.6 montre un espace où les variables sont dispersées de façon à former des zones de densité différentes. En effet, ces variables sont regroupées formant des régions de forte densité et des régions de moindre densité ainsi que des zones de densité nulle.
On peut remarquer également qu'il n'y a pratiquement pas de chevauchement entre ces zones. Autrement dit, les variables formant une zone α ne sont pas présentes dans la zone β . Cette configuration permettrait d'avoir une totale disjonction entre les variables de différents clusters et par consé-

quent une bonne réduction de la complexité de l'instance en question.
Le clustering basé sur la densité semble être la technique de choix pour ce type de distribution. Nous proposons dans le chapitre suivant, différentes adaptations de l'algorithme DBSCAN pour les instances suivant ce type de distribution.

4.4.4 Étude de cas : Modèle de dispersion 2 : FLA500

Nous étudions dans ce deuxième cas l'instance FLA500 dont la dispersion correspond au deuxième modèle précédemment introduit. La table 4.4 et figure 4.7 présentent les principaux paramètres statistiques de cette instance.

Benchmark	FLA-500
Nombre de clauses	2205
Nombre de variables	500
Valeur minimale	X_1
Valeur maximale	X_{500}
Nombre de variables manquantes	0
Valeur moyenne	X_{248} $(+X_{244})$ $(-X_{252})$
Quartiles	$X_{123}, X_{250}, X_{373}$ $(+)X_{117}, X_{248}, X_{367}$ $(-)X_{130}, X_{252}, X_{379}$
Mode	X_{487} Occurrence = 16
Moyenne d'occurrence	13
Variables symétriques	#45

TABLE 4.4 – *Elements de Statistiques pour le benchmark FLA-500*

Le tableau 4.4 ainsi que la figure 4.7 exhibent les caractères statistiques les plus importants de l'instance SAT fla500.
Les principaux paramètres de tendance centrale et de dispersion, à savoir la moyenne, la médiane ainsi que l'écart type indiquent un équilibre et une symétrie dans la dispersion des variables. En effet, l'IQR est exactement de 250 pour une instance comprenant 500 variables.
Par ailleurs, l'instance ne présente qu'un seul mode, la variable $X_4$87, avec une occurrence de 26.
Les différentes occurrences du benchmark avec le nombre de variables ayant cette occurrence sont présentées dans le tableau suivant (4.5).

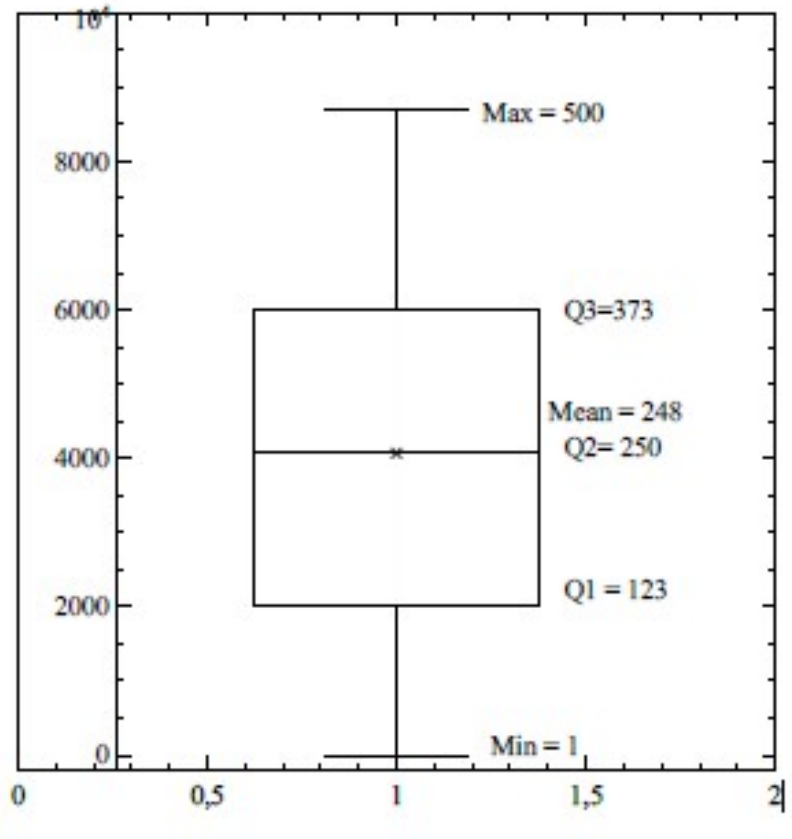


FIGURE 4.7 – *Boite à moustache pour l'intsance FLA-500*

Fréquence	Nombre de variables	Fréquence	Nombre de variables
3	1	16	42
5	3	17	33
6	8	18	20
7	16	19	11
8	20	20	13
9	32	21	6
10	43	22	5
11	41	23	2
12	59	24	2
13	57	25	1
14	44	26	1
15	40		

TABLE 4.5 – *Fréquences d'apparition des variables pour le benchmark FLA-500*

Le tableau 4.5 présente un récapitulatif des fréquences d'apparition des variables. On note l'existence de 23 fréquences différentes.
La répartition de ces variables sur les clauses est exhibée dans la figure 4.8.

La figure 4.5 illustre une dispersion aléatoire des variables sur les clauses. Ces variables sont parsemées dans tout l'espace sans formation de la moindre zone de densité significative. Ce type de dispersion nous

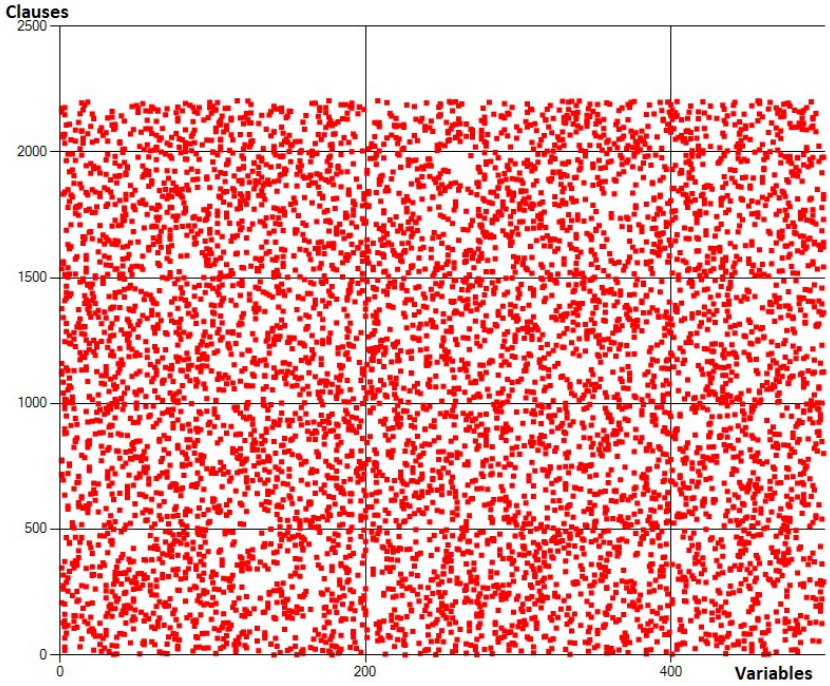


FIGURE 4.8 – *Distribution des variables sur les clauses pour l'instance FLA-500*

ramène au clustering basé sur les grilles.
Une adaptation de l'algorithme STING est proposée pour les instances SAT dont la distribution des variables sur les clauses est aléatoire dans le chapitre 6.

4.4.5 Étude de cas : Modèle de dispersion 2 : AIM-200

Certaines instances dont la distribution des variables sur les clauses correspond au second modèle possèdent une particularité : la majorité de leurs variables ont une fréquence élevée. Parmi ces instances, l'instance Aim-200 que nous allons étudier dans ce qui suit.
Le tableau 4.6 et la figure 4.9 présentent les principaux paramètres statistiques de ce benchmark.

Benchmark	AIM-200
Nombre de clauses	1182
Nombre de variables	200
Valeur minimale	X_1
Valeur maximale	X_{200}
Nombre de variables manquantes	0
Valeur moyenne	X_{100} $(+X_{100})$ $(-X_{100})$
Quartiles	X_{51}, X_{101}, X_{151} $(+)X_{51}, X_{101}, X_{152}$ $(-)X_{51}, X_{101}, X_{151}$
Mode	X_{149} Occurrence = 18
Moyenne d'occurrence	17
Variables symétriques	#155

TABLE 4.6 – *Elements de Statistiques pour le benchmark AIM-200*

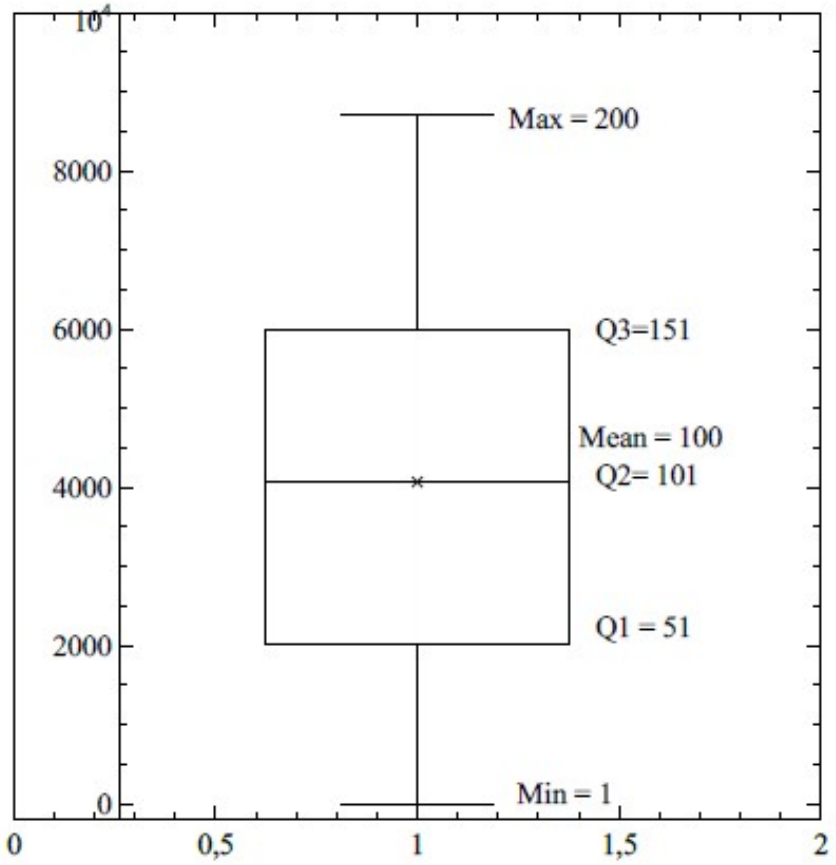


FIGURE 4.9 – *Boite à moustache pour l'instance AIM-200*

Les paramètres exhibés par le tableau 4.6 ainsi que la figure 4.9 montrent une symétrie presque parfaite de la distribution des variables avec une variable moyenne de 100. Cette moyenne est identique pour les variables sous forme positive et négative.
Les variables X_{51}, X_{101}, X_{151} représentent respectivement les quartiles

$Q_1, Q_2 et Q_3$. Ces valeurs indiquent une forte symétrie et un équilibre dans la dispersion des variables.

La particularité de cette instance réside dans le fait que la majorité de ses variables ait la même fréquence. Le tableau 4.7 expose ces différentes fréquences.

Fréquence	15	16	17	18
Nombre de variables	01	08	42	149

TABLE 4.7 – *Fréquence d'apparition des variables du benchmark AIM-200*

Le tableau 4.7 présente les différentes fréquences retrouvées dans l'instance aim200 ainsi que le nombre de variables ayant ces fréquences. On remarque qu'il n'y a que quatre différentes fréquences pour un ensemble de 200 variables. Cette instance est multimodale et possède un nombre de modes dépassant les deux tiers du nombre total de variables (149/200). Par ailleurs, le nombre de variables symétriques est d'environ 155. Ce type de distribution est appelé instance équilibrée dans la littérature.

La dispersion de ces variables sur l'ensemble des clauses est présentée dans la figure 4.10.

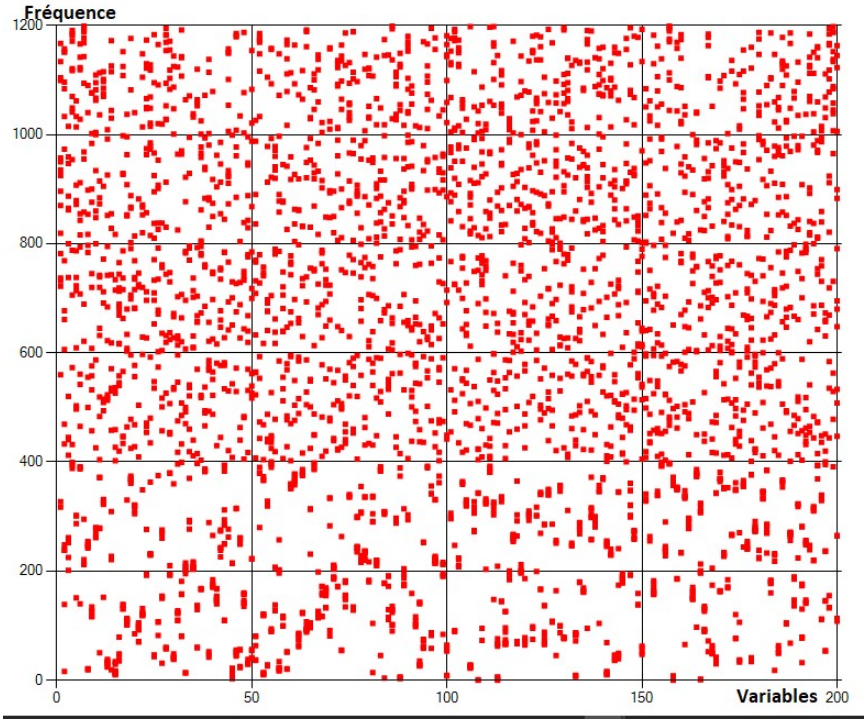


FIGURE 4.10 – *Distribution des variables sur les clauses pour l'instance AIM-200*

La figure 4.10 illustre une distribution des variables sur les clauses identiques à la précédente et correspondant au deuxième modèle introduit. En effet, les variables sont dispersées aléatoirement dans l'espace et

ne forment aucune région de densité particulière. Cependant, une particularité relative à la fréquence des variables de cette instance est à souligner. Plus de deux tiers de ces variables présentent la plus haute fréquence. Cette particularité nous a amené à considérer les algorithmes d'extraction de modèles fréquents comme technique de clustering pour ce type de distribution.
Une approche hybridant l'algorithme Apriori avec le k-means est proposée dans le chapitre 7 pour ces instances.

4.5 Conclusion

Pour le même jeu de données, différentes techniques de clustering peuvent aboutir à différents résultats. Ce qui rend le choix de la méthode de clustering à la fois indispensable et compliqué. Dans ce chapitre, nous avons étudié et exploré la distribution des variables sur les clauses dans le but de déterminer la technique de clustering la plus adéquate à chaque instance.
Trois différentes distributions ont été révélées. La première décrit un espace où les variables forment des régions de haute densité parsemées de régions de faible densité voire des régions vides. À l'opposé, le deuxième modèle de distribution définit un espace où les variables sont dispersées aléatoirement ne formant aucune zone de densité significative. Finalement, la troisième distribution suit le même schéma que la précédente avec la particularité que l'ensemble des variables présentent une fréquence élevée.
À chacune de ces distributions est attribuée une technique de clustering jugée plus appropriée. Une adaptation de chacune de ces techniques est présentée dans les chapitres qui suivent.

5 DBSCAN Bidimensionnelle et Multidimensionnelle pour les instances SAT

5.1 Introduction

Nous avons vu dans le chapitre précédent, les différents modèles de distribution des variables sur les clauses. Cette exploration nous a permis de déterminer la méthode de clustering la plus adéquate à chacune de ces distributions.
Dans ce chapitre, nous proposons une adaptation de l'algorithme DBSCAN pour les instances SAT dont la distribution définit un espace où les variables sont dispersées de manière à former des zones de densité considérable.
Le clustering basé sur la densité semble être la méthode la plus appropriée à ce type de distribution et particulièrement l'algorithme DBSCAN. En effet, l'algorithme DBSCAN est l'un des algorithmes basés sur la densité les plus populaires dont l'objectif principal est d'extraire les zones de forte densité. Par ailleurs, tout élément n'appartenant à aucune zone dense est considéré comme aberrant. Cette partie de l'algorithme DBSCAN n'est pas prise en compte lors de l'adaptation pour les instances SAT. Notre objectif étant de réduire la complexité de ces dernières dans le but de les résoudre.
Nous proposons dans ce chapitre, deux principales adaptations du DBSCAN. Une adaptation bidimensionnelle et une multidimensionnelle. Dans l'approche bidimensionnelle, les variables sont considérées comme point de départ lors de la détermination de régions denses, alors que ce sont les clauses qui sont prises en compte lors de la génération de clusters dans l'approche multidimensionnelle.

5.2 Approche DBSCAN Bidimensionnelle et Expérimentations

L'idée générale de l'adaptation bidimensionnelle du DBSCAN Hireche et Drias [2018] est de considérer la variable comme point de départ dans la génération des clusters.

L'approche proposée est une approche séquentielle de clustering-résolution, qui consiste à résoudre un cluster après sa création. En effet, à chaque fois qu'un cluster est extrait, la résolution de celui-ci est immédiate permettant la propagation de sa solution sur le reste de l'instance, réduisant ainsi le nombre de clauses de celle-ci.
La figure 5.1 illustre ce fonctionnement :

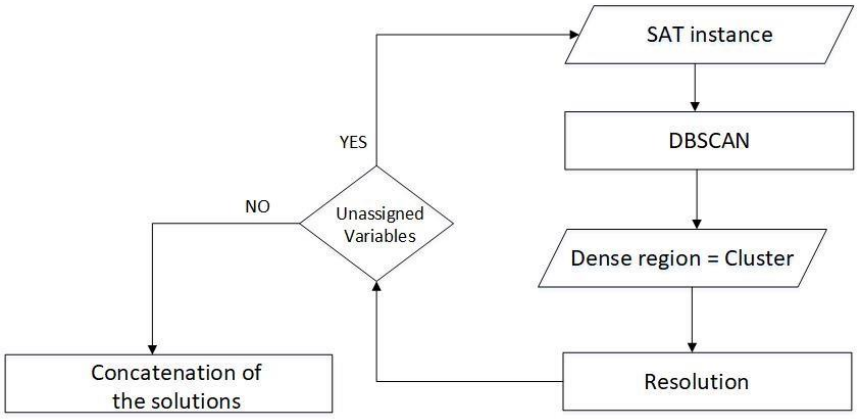


FIGURE 5.1 – *Approche de clustering-résolution DBSCAN*

Notre adaptation du DBSCAN commence par sélectionner la variable la plus fréquente de l'instance comme centre de cluster. Le choix de la variable la plus fréquente est guidé par l'impact que pourrait avoir l'affectation de cette variable sur le nombre de clauses satisfaites et par conséquent sur le temps de résolution de l'instance. A partir de cette variable, une région ou un cluster est déterminé.
Après l'extraction d'une région, cette dernière est résolue en exécutant l'algorithme DPLL ou BSO en fonction du nombre de ses variables représentatives de cluster. En effet, les clusters dont le nombre de variables n'excède pas un certain seuil sont résolus en utilisant le DPLL. L'algorithme BSO est utilisé dans le cas échéant.
La solution de chaque cluster est propagée sur l'ensemble des clauses restantes dans le but de réduire leur nombre et permettre d'avoir une solution globale plus effective.
Ce processus de clustering-résolution est alors poursuivi jusqu'à ce qu'il n'y ait plus aucune variable à instancier. L'algorithme 7 résume le fonctionnement de cette adaptation bidimensionnelle du DBSCAN (distance *Hamming*).

Algorithm 7 DBSCAN bidimensionnel

Données LX : Liste des variables
LCl : Liste des clauses
LN : Liste des voisins
LC : Liste des clusters
Début
Choisir la variable qui se répète le plus dans les clauses (x_{max})
Créer un cluster avec x_{max} comme variable représentative
Retirer x_{max} de la liste LX
Pour toutes les variables de la liste LX
Si le nombre de clauses partagées (distance hamming) est supérieur à un certain seuil
Ajouter la variable aux variables représentative du cluster
Retirer la variable de la liste LX
si LX n'est pas vide alors reprendre le processus
(Les clauses sont assignées au cluster le plus proche)
Fin

Complexité : Le pire cas de cette approche est représenté par le cas où tous les clusters ne possèdent qu'une et une seule variable représentative. Ce qui signifierait qu'il n'existe aucune paire de variables satisfaisant la condition de distance requise. Par conséquent, à chaque itération et pour chaque nouvelle variable, l'algorithme doit parcourir toutes les variables restantes.
Si l'on considère que n est le nombre de variables dans une instance SAT, alors le nombre de calculs de distance est de $n + (n-1) + (n-2) + \cdots + 1$ ce qui est équivalent à $\frac{n(n+1)}{2}$. La complexité au pire cas est alors de l'ordre de $\theta(n^2)$.

Trois différentes approches sont proposées dans la définition et l'extraction d'une région à partir d'une variable donnée.

5.2.1 DBSCAN-Bidimensionnel : rayon Euclidien

Cette approche considère la distance *Euclidienne* dans la détermination de la région d'une variable centrale x_i. Une région dont le point de départ est la variable x_i est définie par toutes les variables dans un rayon r (de $x_i - r$ à $x_i + r$) partageant au moins une clause avec x_i.
La figure 5.2 ainsi que le tableau 5.1 illustrent le taux de satisfaction de l'approche de clustering DBSCAN Bidimensionnelle considérant comme rayon de définition d'une région la distance *Euclidienne*.
Remarque : Les taux de satisfaction des benchmarks $fla-500$ et $aim-200$ étant assez bas par rapport aux autres, nous les avons présentés séparément pour permettre une bonne appréciation du graphique.

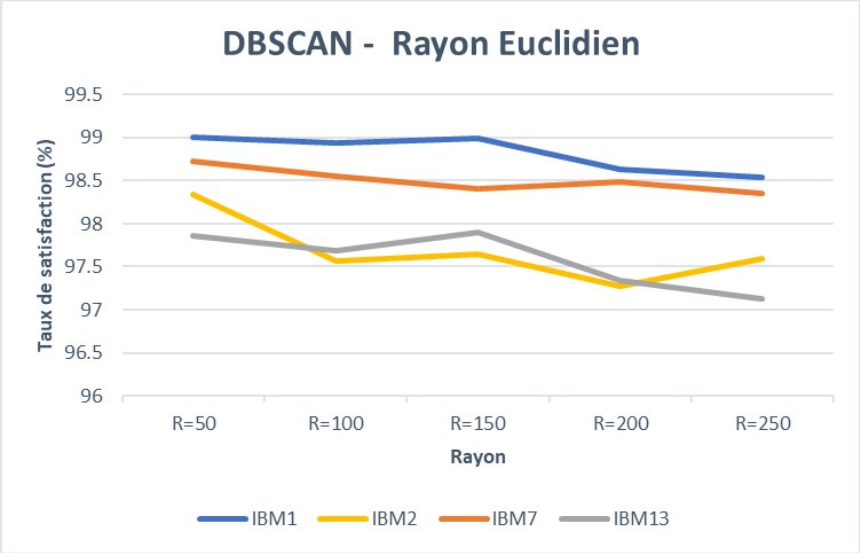


FIGURE 5.2 – *DBSCAN bidimensionnel à rayon Euclidien - Taux de Satisfaction*

Rayon / Benchmark	Taux de satisfaction (%)	
	FLA-500	AIM-200
R=50	96.10	92.22
R=100	96.73	92.47
R=150	96.42	92.47
R=200	95.96	92.72
R=250	96.19	-

TABLE 5.1 – *DBSCAN bidimensionnel à rayon Euclidien - Taux de satisfaction*

La figure et le tableau précédents nous permettent d'observer que lorsque le rayon est augmenté, le taux de satisfaction des instances est diminué. En effet, plus le rayon est augmenté plus le nombre de variables à instancier dans un cluster est augmenté. Cette augmentation accroit la probabilité d'utiliser de l'algorithme BSO au lieu du DPLL et engendre une résolution moins effective.
A titre d'exemple, le nombre maximal de variables retrouvées dans un cluster pour l'instance IBM1 est de 43 pour un rayon=50 alors qu'il est de 118 pour un rayon=250.

Le tableau suivant (5.2) expose les temps d'exécution correspondant à l'approche DBSCAN-bidimensionnel avec distance *Euclidienne* comme paramètre de définition d'une région.

Rayon / Benchmark	Temps d'exécution(s					
	IBM1	IBM2	IBM7	IBM13	FLA-500	AIM-200
R=50	377.82	68.19	331.7	360.39	0.73	13.71
R=100	527.22	112.68	753.41	374.8	102.82	3.09
R=150	566.28	161.43	954.32	644.82	227.13	59.29
R=200	726.99	192.33	959.72	1115.53	62.63	9.55
R=250	834.84	244.55	1006.3	1187.72	13.4	-

TABLE 5.2 – *DBSCAN bidimensionnel à rayon Euclidien - Temps d'exécution*

Les résultats précédents exposent une augmentation du temps d'exécution lors de l'augmentation du rayon définissant une région. Cette augmentation induit une élévation du nombre de variables par cluster et par conséquent celle du temps de résolution.

5.2.2 DBSCAN-Bidimensionnel : rayon basé sur distance de Hamming

Cette approche contrairement à la première considère la distance de *Hamming* dans la détermination des clusters. Dans ce cas, la région dont le point de départ est x_i regroupe toutes les variables avec lesquelles au moins r clauses sont communes avec x_i.

Les figures 5.3 et 5.4 présentent les taux de satisfaction ainsi que les temps d'exécution des différents benchmarks pour l'approche considérant la distance de *Hamming* comme distance de sélection des éléments inclus dans un cluster.

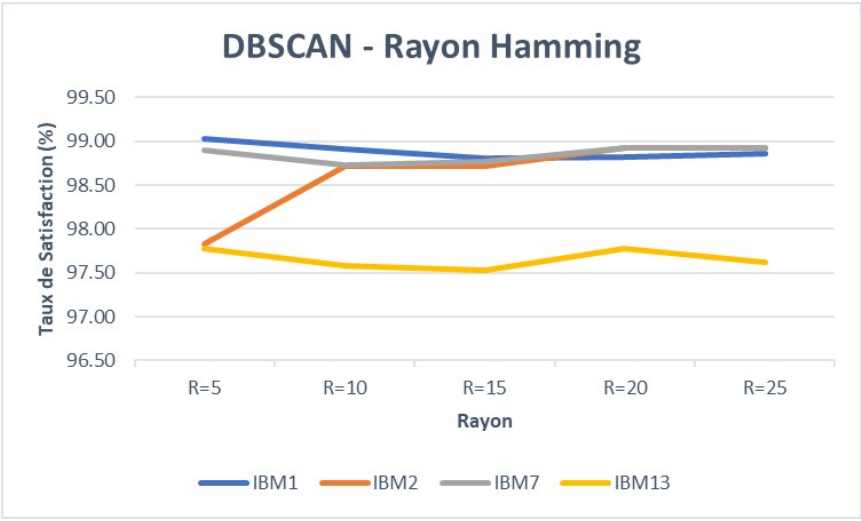


FIGURE 5.3 – *DBSCAN bidimensionnel à rayon Hamming - Taux de Satisfaction*

La figure 5.3 illustre la variation du taux de satisfaction des différentes instances en fonction du rayon définissant la taille du cluster. On note une légère amélioration de la qualité de solution suivie d'une stagnation de celle-ci lors de l'augmentation de ce rayon.
Contrairement à l'approche précédente, l'augmentation du rayon favorise

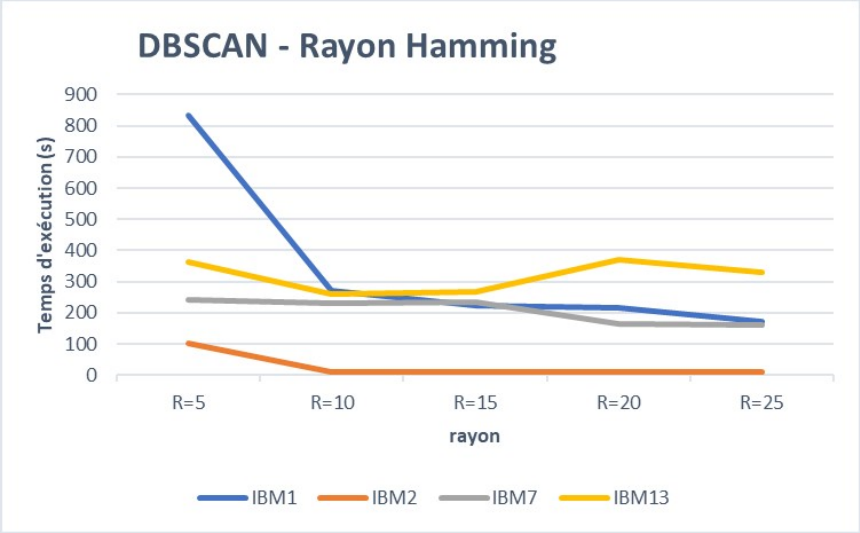


FIGURE 5.4 – *DBSCAN bidimensionnel à rayon Hamming - Temps de résolution*

la diminution du nombre de variables par cluster. Cette approche considère le nombre de clauses partagées entre les variables, condition qui n'est pas facile à satisfaire à partir d'un certain seuil. Ce qui explique la stagnation de la qualité de solution.
Par ailleurs, une diminution du temps d'exécution est notée lors de l'augmentation du rayon. Résultat de la diminution du nombre de variables à instancier par cluster.
Le nombre maximal de variables par cluster de l'instance IBM1 pour un rayon=5 est de 44 alors qu'il est de 5 pour un rayon de 25.
Remarque : Les benchmarks aim-200 et fla-500 ne sont pas représentés, car le nombre maximal de clauses communes entre les variables est de 3, ce qui n'est pas satisfaisant pour vérifier cette approche.

5.2.3 DBSCAN-Bidimensionnel : Régions denses

Les deux précédentes définitions ne considèrent pas une région selon le nombre de variables qu'elle contient, mais par rapport au nombre de clauses partagées entre ces variables. Cette troisième définition considère la densité d'une région lors de sa sélection. Quelques définitions relatives au voisinage, à la densité et au seuil minimal de densité doivent être introduites.

Définition 5.1 : Voisinage
Le voisinage d'une variable x_i correspond à l'ensemble des variables avec lesquelles x_i partage au moins une clause.

Définition 5.2 : densité d'une région
Si l'on considère x_i comme la variable centrale d'une région et S l'ensemble de variables appartenant à son voisinage. Alors la densité de cette région

est représentée par la densité de la variable x_i définie comme suit ;

$$Densit(x_i, S) = \frac{\sum_{j \in S} w_{x_i x_j}}{|S|} \tag{5.1}$$

Où $w_{x_i x_j}$représente le nombre de clauses partagées par les deux variables x_i et x_j.

Exemple 5.1 :

Soit l'ensemble des clauses suivantes :

$C_1 = x_1, x_2, x_3$

$C_2 = x_1, x_3$

$C_3 = x_1, x_4$

$C_4 = x_2, x_4, x_5$

La densité de la région dont la variable de départ est x_1, à savoir l'ensemble de variables $\{x_2, x_3, x_4\}$, est de $D(x_1) = \frac{w_{1,2}+w_{1,3}+w_{1,4}}{3} = \frac{1+2+1}{3} =$ 1.33 .

Définition 5.3 : Région dense Une région est dite dense si et seulement si sa densité est supérieure ou égale au seuil minimal de densité prédéfini. Ce seuil est défini par la densité moyenne de l'ensemble des variables de l'instance et est calculé comme suit ;

$$ThresholdDensity(TD) = \frac{\sum_{i=0}^{m} D(x_i)}{m} \tag{5.2}$$

Avec m le nombre de variables dans l'instance de problème.

Exemple 5.2 :

La région définissant x_1 comme variable centrale de l'exemple 5.1 est considérée comme dense. Sa densité de 1.33 est supérieure au TD qui est de 1.166. $TD = \frac{D(x_1)+D(x_2)+D(x_3)+D(x_4)+D(x_5)}{5} = \frac{4/3+4/4+3/2+3/3+2/2}{5} =$ 1.166.

Dans cette troisième approche, le point de départ pour la création d'un cluster est représenté par la variable qui possède à la fois la plus haute fréquence et dont la densité est supérieure ou égale au TD. Cette région comportera alors l'ensemble des variables partageant au moins une clause avec x_i.

Le seuil de densité utilisé pour valider une région(ou cluster) est dynamique. En effet, à chaque itération, le nombre de clauses dans l'instance diminue ce qui affecte la densité moyenne des variables. Le seuil de densité est alors recalculé après la propagation de la solution d'un cluster.

Le tableau 5.3 indique les taux de satisfaction et les temps d'exécution de l'approche du DBSCAN prenant en considération la densité d'une région dans la création de clusters.

Nom du benchmark	Taux de satisfaction (%)	Temps d'exécution (s)
IBM1	97.36	1852.48
IBM2	96.72	590.33
IBM7	96.70	5750.97
IBM13	97.95	1096.80
FLA500	95.78	25.40
AIM200	92.05	22.22

TABLE 5.3 – *Comparaison entre DBSCAN Bidimensionnel et son amélioration - Temps d'exécution*

Les taux de satisfaction de cette approche sont satisfaisants. Par ailleurs, le recalcul à chaque itération du seuil de densité engendre un temps de calcul important affectant l'efficience de l'approche.

5.2.4 Amélioration du DBSCAN bidimensionnel

Avant d'adopter le bon prétraitement des instances SAT (chapitre 4, section 3), un prétraitement initial des instances SAT avait été proposé. Ce prétraitement consistait à satisfaire les conditions relatives aux tautologies ainsi qu'aux variables redondantes. Cependant, le prétraitement des clauses inclusives favorisait l'élimination de la plus petite clause. Si l'on considère que toutes les variables de la clause C_i sont présentes dans la clause C_j alors la clause C_i était supprimée.
L'adoption et l'utilisation du bon prétraitement représente l'une des deux améliorations apportées à l'approche bidimensionnelle du DBSCAN Hireche et Drias [2019]. La seconde amélioration consiste en l'utilisation d'une représentation verticale des clauses lors de la validation d'une solution (fonction fitness). Cette amélioration permet un gain considérable de temps.

Les tableaux 5.4 et 5.5 introduisent la comparaison entre les temps d'exécution et taux de satisfaction de l'approche DBSCAN avec distance *Euclidienne* et son amélioration.

Temps d'exécution (s)	Rayon	Nom du benchmark			
		IBM1	IBM2	IBM7	IBM13
DBSCAN Bidimensionnel	05	834,94	100,	241,45	364,54
	10	270,64	11,39	229,26	259,13
	15	224,09	11,25	234,23	267,96
	20	217,4	8,85	164,18	371,21
	25	172,59	8,76	160,93	328,37
DBSCAN Bidimensionnel (Amélioration)	05	687,42	6,23	87,25	213,09
	10	930,14	5,6	69,7	208,7
	15	139,04	5,6	67,38	21,98
	20	125,65	5,65	66,19	245,46

TABLE 5.4 – *Comparaison entre DBSCAN Bidimensionnel et son amélioration - Temps d'exécution*

Taux de satisfaction (%)	Rayon	Nom du benchmark			
		IBM1	IBM2	IBM7	IBM13
DBSCAN Bidimensionnel	05	99.02	97.83	98.90	97.78
	10	98.91	98.71	98.73	97.59
	15	98.80	98.71	98.77	97.53
	20	98.82	98.93	98.93	97.78
	25	98.86	98.93	98.93	97.62
DBSCAN Bidimensionnel (Amélioration)	05	99.55	99.55	99.43	98.35
	10	99.46	99.52	99.53	98.31
	15	99.52	99.49	99.45	98.27
	20	99.36	99.51	99.45	98.29

TABLE 5.5 – *Comparaison entre DBSCAN Bidimensionnel et son amélioration - Taux d'exécution*

La comparaison des résultats de l'approche bidimensionnelle du DBSCAN avec son amélioration nous permet d'observer une amélioration des résultats en termes de taux de satisfaction. Cet accroissement de performance est dû à l'efficacité du prétraitement proposée. En effet, la suppression de certaines clauses influe sur la solution globale de l'instance.
On note également une meilleure efficience des résultats, résultat de l'amélioration de la fonction de validation des solutions. A titre d'exemple, le temps d'exécution de l'instance IBM2 en seconde est de 100 pour un rayon de 5 dans la première approche. Il est de 6.23 après amélioration de celle-ci.

5.2.5 Comparaison

Dans le but de déterminer la meilleure approche du DBSCAN bidimensionnelle, une comparaison des différentes approches proposées est effectuée. Le tableau 5.6 illustre les meilleurs taux de satisfaction de chaque approche, alors que le tableau 5.7 présente les temps de résolution relatifs à ces taux.

Méthode / Benchmark	Meilleurs Taux de satisfaction (%)					
	IBM1	IBM2	IBM7	IBM13	FLA-500	AIM-200
Distance Euclidienne	99.01	98.34	98.72	97.9	96.73	92.72
Distance Hamming	99.02	98.93	98.93	97.78	-	-
Densité	97.36	96.72	96.7	97.95	95.78	92.05
DBSCAN améliorée	99.55	99.55	99.53	98.35	-	-

TABLE 5.6 – *Comparaison des différentes adaptations bidimensionnelles du DBSCAN-Taux de satisfaction*

Méthode / Benchmark	Temps de résolution des Meilleurs Taux de satisfaction (s)					
	IBM1	IBM2	IBM7	IBM13	FLA-500	AIM-200
Distance Euclidienne	377.82	68.19	331.7	644.82	102.82	9.55
Distance Hamming	834.94	8.85	160.93	364.54	-	-
Densité	1852.48	590.33	5750.97	1096.8	25.4	22.22
DBSCAN améliorée	687.42	6.23	69.7	213.09		

TABLE 5.7 – *Comparaison des différentes adaptations bidimensionnelles du DBSCAN-Temps d'exécutions*

Après examen de ces résultats, on note une distinction entre les deux premières approches et la troisième. En effet, les approches déterminant un cluster en utilisant la distance *Euclidienne* et la distance de *Hamming* offrent de meilleurs résultats que l'approche définissant des régions denses.
On note par ailleurs que l'approche *Hamming* est plus effective et efficiente que l'approche utilisant la distance *Euclidienne*. Cette approche semble plus adéquate aux instances de problème SAT.
L'amélioration proposée porte ses fruits en permettant une réduction du temps d'exécution et une meilleure qualité des solutions.

5.3 Approche DBSCAN Multidimensionnelle et Expérimentations

L'adaptation multidimensionnelle du DBSCAN Hireche et Drias [2019], contrairement à la première adaptation, considère les clauses comme principal paramètre lors du clustering. Chaque clause est représentée par un seul et unique point dans un espace multidimensionnel. En effet, dans l'approche précédente, la clause $C_i = x_j, x_k, x_l$ est représentée par les points $p_1(j,i), p_2(k,i) et p_3(l,i)$. Cette clause est représentée par le seul et unique point $p(j,k,l)$ dans l'approche multidimensionnelle.
On admet alors que cet espace multidimensionnel comprend autant de dimension que de nombre de variables dans l'instance.
Le schéma général de ce clustering est similaire au précédent. A chaque itération, une clause est sélectionnée aléatoirement pour représenter le centre d'un cluster dont les éléments sont sélectionnés en utilisant la similarité d'intersection (chapitre 2, equation 2.8). Toutes les clauses partageant au moins r variables avec la clause centrale d'un cluster sont affectées à celui-ci.
Chaque fois qu'un cluster est créé, celui-ci est immédiatement résolu par l'algorithme DPLL ou BSO et sa solution est propagée sur l'ensemble des clauses restantes.

Trois différentes approches sont proposées en fonction de la définition d'une région et de la résolution de cette dernière. La figure suivante (5.5) illustre les différentes définitions de région.

Les deux premières approches illustrées par les figures 5.5.a et 5.5.b regroupent dans un même cluster toutes les clauses dont la similarité avec la clause centrale est supérieure ou égale à un seuil prédéfini. Dans le

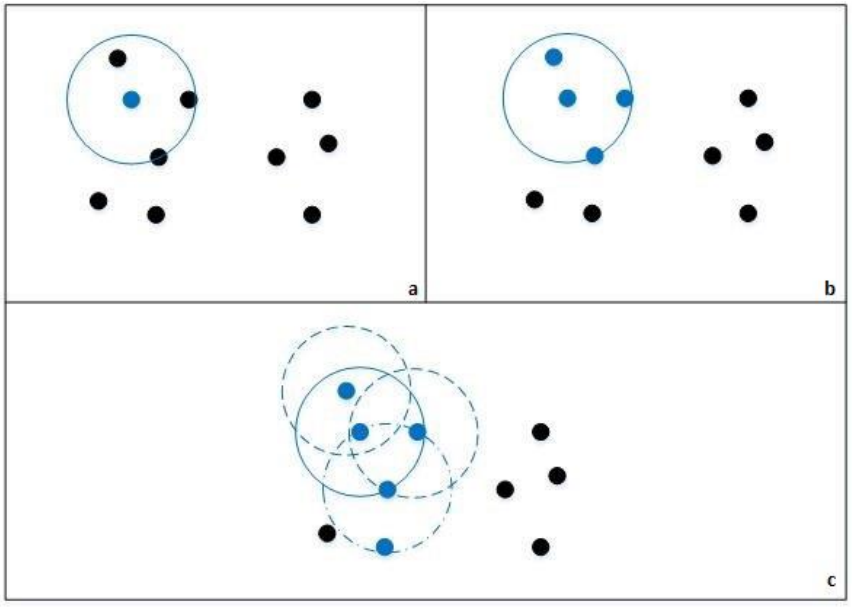


FIGURE 5.5 – *DBSCAN multidimensionnel : différentes définitions de région de densité*

premier cas, les variables représentatives de cluster sont représentées par les variables de la clause centrale. Alors que toutes les variables comprises dans le cluster sont considérées comme représentatives de celui-ci dans la deuxième approche.
Les points en bleu dans la figure représentent les clauses dont les variables sont considérées comme représentatives de cluster.
Dans la dernière approche (figure 5.5.c), un cluster regroupe les clauses similaires à la clause centrale comme pour les deux premières approches. Toutes les clauses similaires aux clauses du cluster sont assignées à ce même cluster. Évidemment, toutes les variables comprises dans le cluster sont considérées comme représentatives de cluster.
Remarque : Cet élargissement du rayon ne s'étend pas à plus de deux rangs. Autrement, toutes les clauses pourraient faire partie du même cluster.

En résumé, le fonctionnement de l'approche DBSCAN multidimensionnel est comme suit (algorithme 8) :

Algorithm 8 DBSCAN multidimensionnelle

Données : LCl : Liste des clauses
LN : Liste des voisins
LC : Liste des clusters
Début
Sélectionner aléatoirement une clause et créer un cluster
Pour chaque clause de LCl
Si le nombre de variables partagées avec la clause initiale (Distance de Hamming) est supérieur au seuil
Ajouter la clause au cluster
Si LCl n'est pas vide, reprendre le processus
Fin

Complexité : Tout comme pour l'approche bidimensionnelle, le pire cas de cet algorithme est défini par le cas où l'algorithme calcule, pour chaque itération, la distance de la clause avec toutes les clauses restantes sans satisfaction du seuil de distance. La complexité est alors de l'ordre de $\theta(m^2)$ pour m clauses.

Afin de valider l'approche de clustering DBSCAN multidimensionnelle, des tests ont été effectués sur les instances SAT précédemment introduites.

La figure 5.6 et tableau 5.8 suivants introduisent les taux de satisfaction et les temps d'exécution de cette approche avec ses différentes sous-classes.

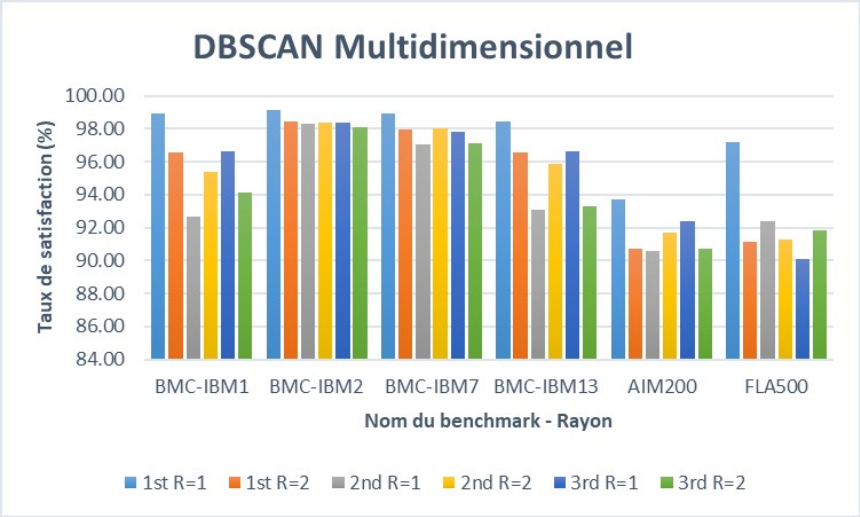


FIGURE 5.6 – *DBSCAN Multidimensionnelle – Taux de satisfaction*

Benchmark	Temps d'exécution (s)					
	1st		2nd		3rd	
	R=1	R=2	R=1	R=2	R=1	R=2
IBM1	84.15	65.6	369.26	171.32	765.39	747.73
IBM2	2.22	2.17	6.49	2.4	41.34	7.24
IBM7	45.26	47.92	257.07	48.38	518.22	182.15
IBM13	430.44	120.23	701.57	337.95	2192.62	262.14
AIM200	0.1	0.06	2.55	0.06	196.39	0.07
FLA500	0.06	0.06	6.005	0.09	355.24	0.1

TABLE 5.8 – *DBSCAN Multidimensionnelle – Temps d'exécution*

En observant les résultats de la figure 5.6 et du tableau 5.8, on remarque que l'augmentation du rayon dans la première approche induit à une diminution du taux de satisfaction conjointement à celle du temps d'exécution. Cette double diminution est due à la diminution du nombre de clauses par cluster qui influe sur la qualité de solution. En effet, plus il y'a de clauses dans un cluster pour valider une solution partielle plus la solution globale est effective.

Dans la seconde approche, contrairement à l'approche précédente, l'augmentation du rayon induit une amélioration du taux de satisfaction des clauses. En effet, l'augmentation rayon induit une diminution du nombre de clauses par cluster. Cette diminution favorise une bonne résolution du cluster, car toutes les variables de celui-ci doivent être instanciées sans que toutes les clauses contenant ses variables soient incluses dans le cluster.
Le temps d'exécution est quant à lui diminué en raison de la baisse du nombre de variables à instancier par cluster.
Enfin, l'augmentation du rayon dans la troisième approche induit une diminution du nombre de clauses et de variables à instancier. Cependant, la définition de région considérée dans cette approche impose l'assignation d'un grand nombre de variables par cluster. Ce qui amoindrit la qualité de solution globale avec l'augmentation du rayon. Néanmoins, le temps d'exécution diminue de manière considérable lors de l'augmentation du rayon. Cette réduction s'explique d'une part par la réduction du nombre de clauses dans le cluster et d'autre part par la réduction du nombre de clauses dans l'instance à chaque propagation de solution partielle.
En comparant ces différents résultats, on peut constater et conclure que la première approche représente la meilleure adaptation multidimensionnelle du DBSCAN.

5.4 Comparaison entre l'approche bidimensionnelle et multidimensionnelle

Une première comparaison du point de vue design de la solution serait de dire que DBSCAN bidimensionnelle est plus adaptée pour une approche de résolution séquentielle. En effet, les variables d'un cluster donné partagent des clauses avec des variables qui peuvent exister dans d'autres clusters. A l'opposé, comme DBSCAN multidimensionnel re-

groupe des clauses dans des clusters, les sous-instances SAT obtenues sont disjointes et peuvent donc être résolues de manière parallèle. C'est cette dernière approche qui sera soumise à un calcul parallèle en utilisant un GPU et qui sera présentée dans le dernier chapitre.
Du point de vue empirique et afin de pouvoir choisir la meilleure adaptation du DBSCAN pour les instances SAT, une comparaison entre les différentes approches proposées dans ce chapitre est réalisée. La figure 5.7, les tableaux 5.9 et 5.10 illustrent les meilleurs résultats pour chaque approche en termes de qualité de solution. Les temps d'exécution de ces méthodes y sont également présentés.

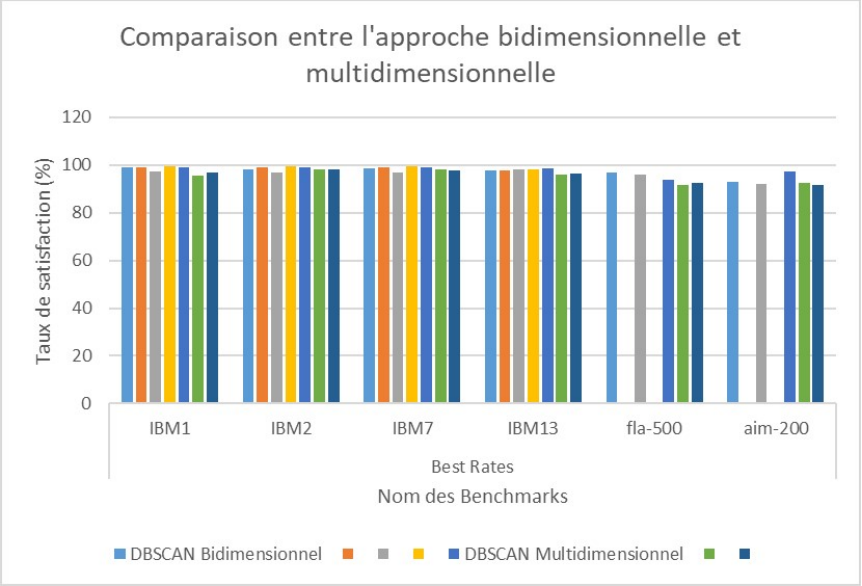


FIGURE 5.7 – *Comparaison entre l'approche bidimensionnelle et multidimensionnelle – Taux de satisfaction*

Temps d'exécution (s)		IBM1	IBM2	IBM7	IBM13
DBSCAN bidimensionnel	Distance Euclidienne	377.82	68.19	331.7	644.82
	Distance Hamming	834.94	8.85	160.93	364.54
	Densité	1852.48	590.33	5750.97	1096.8
	DBSCAN améliorée	687.42	6.23	69.7	213.09
DBSCAN multidimensionnel	Première Approche	84.15	2.22	45.26	430.44
	Deuxième Approche	171.32	2.4	48.38	337.95
	Troisième Approche	765.39	41.34	518.22	2192.62

TABLE 5.9 – *Comparaison entre l'approche bidimensionnelle et multidimensionnelle – Temps d'exécution*

Temps d'exécution (s)		FLA500	AIM200
DBSCAN bidimensionnel	Distance Euclidienne	102.82	9.55
	Distance Hamming	-	-
	Densité	25.4	22.22
	DBSCAN améliorée	-	-
DBSCAN multidimensionnel	Première Approche	0.1	0.06
	Deuxième Approche	0.06	6.005
	Troisième Approche	196.39	0.14

TABLE 5.10 – *Comparaison entre l'approche bidimensionnelle et multidimensionnelle – Temps d'exécution*

L'analyse de ces résultats nous indique que les approches bidimensionnelles définissant un cluster en utilisant la distance de *Hamming* et la distance *Euclidienne* ainsi que la première adaptation du DBSCAN multidimensionnelle présentent les meilleurs taux de satisfactions.
En termes d'efficience, l'approche bidimensionnelle *Hamming* ainsi que la première approche multidimensionnelle présentent les meilleurs temps d'exécution.
Le reste des approches offrent des résultats satisfaisants, mais restent moins efficaces que celles citées précédemment.
Par ailleurs, l'utilisation judicieuse des structures de données lors de la validation des solutions ainsi que l'application d'un prétraitement adéquat a permis d'avoir des solutions plus efficientes.
On remarque également que les instances IBMs présentent de bien meilleurs résultats que le reste des instances. En effet, la distribution des instances IBMs correspond à celle satisfaite par le clustering basé sur la densité.

5.5 Conclusion

Nous avons présenté, dans ce chapitre, différentes adaptations de l'algorithme DBSCAN pour les instances SAT. Cet algorithme de clustering se trouve être la technique la plus adéquate pour les instances dont la distribution des variables sur les clauses définit des zones de forte densité. Deux principales adaptations sont proposées. La première, bidimensionnelle, considère les variables comme principal paramètre dans la création des clusters. Trois différentes définitions de région ont été proposées pour cette première approche considérant la distance *Euclidienne*, la distance de *Hamming* ainsi qu'une définition de la densité propre aux instances SAT.
La seconde adaptation considère les clauses comme principal paramètre au clustering. A chaque itération, une clause est choisie pour être le centre d'une région. Cette région (ou voisinage) comprend les clauses les plus similaires à la clause centrale dans les deux premières approches. Une troisième approche considère, en plus du voisinage de la clause centrale, le voisinage de toutes les clauses appartenant à la région.
Pour ces deux approches, à chaque fois qu'un cluster est créé, celui-ci est résolu en assignant ses variables représentatives par les algorithmes DPLL et BSO en fonction du nombre de ces variables. Des expérimentations ont été conduites dans le but de valider et comparer ces différentes

approches. Les résultats obtenus ont permis de conclure de l'efficacité de ces dernières et particulièrement sur les instances dont la distribution définit des zones de forte densité.

Nous introduirons, dans le prochain chapitre, une adaptation de l'algorithme STING pour les instances dont la distribution est aléatoire.

Approche de clustering basée sur les grilles 6

6.1 Introduction

Nous avons présenté, dans le chapitre précédent, différentes adaptations de l'algorithme DBSCAN pour les instances SAT dont la distribution définit des zones de forte densité. Les résultats obtenus nous ont permis de conclure de l'efficience de cette technique sur ces instances.
Dans ce chapitre, nous traitons les instances dont la dispersion est aléatoire et auxquelles le clustering basé sur les grilles semble être la technique la plus appropriée. Une adaptation de l'algorithme STING est alors proposée dans le but de réduire la complexité de ces instances et faciliter leur résolution.
STING représente l'un des algorithmes basés sur les grilles les plus populaires. Deux principales phases le définissent. La première phase consiste à créer une structure hiérarchique en subdivisant l'ensemble des données. La seconde étape permet alors à déterminer parmi les cellules de cette structure, celles qui répondent aux requêtes du clustering.
Dans notre adaptation de STING, la taille des cellules est prédéfinie permettant un clustering plus efficace.
Cette approche, contrairement à la précédente (DBSCAN) s'articule en deux phases distinctes. En effet, un clustering des instances est réalisé avant de passer à la résolution de chacun de ces clusters indépendamment les uns des autres.

6.2 Clustering basé sur les grilles : STING

Les méthodes de clustering basées sur les grilles sont dites guidées par l'espace et sont totalement indépendantes du nombre d'objets et totalement dépendantes du nombre de cellules de la grille. Ce qui permet un traitement rapide et un gain de temps considérable. En effet, ces algorithmes considèrent une grille sur laquelle les données sont projetées et où chaque cellule définit un cluster. Ces cellules sont définies par un certain nombre de paramètres.
Dans l'algorithme STING (STatistical INformation Grid), les données sont représentées par des paramètres attributs-dépendant tels que les valeurs minimales, maximales et la moyenne, l'écart type, ainsi que la distribution

des données. Et un paramètre non-attribut-dépendant représenté par le nombre d'objets dans la cellule.
Le principe général de l'algorithme est, d'abord, de représenter l'ensemble des données dans une seule et unique cellule racine, qui sera scindée en quatre cellules, conformément aux paramètres définis précédemment. Chacune de ces cellules est divisée en continu, générant plusieurs couches de cellules, jusqu'à la satisfaction d'une certaine condition sur le nombre d'objets dans la couche de feuilles ou toute autre condition. Un intervalle de confiance est alors calculé, déterminant les cellules satisfaisant les requêtes données en entrée et qui représentent les clusters finaux.

Algorithm 9 Algorithme Statistical INformation Grid

Début
Déterminer une couche où commencer.
Pour chaque cellule de cette couche, calculer l'intervalle ou probabilité que cette cellule soit pertinente pour la requête.
A partir de l'intervalle calculé, marquer la cellule comme pertinente ou non pertinente.
Si cette couche est la derniere (bottom), passer à l'étape 6 ; sinon, passer à l'étape 5.
Descendre d'un niveau dans la structure hiérarchique. Passer à l'étape 2 pour les cellules qui forment les cellules pertinentes de la couche de niveau supérieur.
Si la requête est satisfaite, passer à l'étape 8 ; sinon, passer à l'étape 7.
Récupérer les données des cellules appropriées et effectuer un traitement ultérieur. Renvoyer le résultat qui répond à la requête. Passer à l'étape 9.
Trouver les régions des cellules pertinentes. Renvoyer les régions qui répondent à la requête. Passer à l'étape 9.
Stop.
Fin

Complexité :
Les étapes 1, 2 et 3 prennent un temps constant et indépendant de la taille des données. Le temps total est inférieur ou égal au nombre total de cellules de la structure hiérarchique. Le nombre total de cellules étant de $1.33K$, où K est le nombre de cellules dans la couche inférieure. Par conséquent, la complexité globale est $\theta(K)$.
L'algorithme parcourt une fois la base de données pour calculer les paramètres statistiques des cellules. La complexité temporelle de la génération de clusters est $\theta(n)$, où n est le nombre total d'objets. Une fois la structure hiérarchique générée, le temps de traitement de la requête est égal à $\theta(g)$, où g représente le nombre total de cellules de la grille au niveau le plus bas, généralement beaucoup plus petit que n.

6.3 Adaptation du clustering basé sur les grilles pour les instances SAT

L'idée générale de cette approche est de créer des clusters en appliquant l'algorithme STING sur les instances SAT. Une fois créés, une phase de résolution est exécutée pour chacun de ces clusters indépendamment les uns des autres.
Contrairement à l'approche originale de l'algorithme STING qui consiste à construire une structure hiérarchique dans un premier temps et l'explorer dans un second temps. Dans notre adaptation, la taille de chaque cellule de la grille est préalablement définie permettant un accès direct au niveau des feuilles et une réduction du temps d'exécution.
L'approche proposée se déroule alors en deux phases dont la première correspond à la création des clusters. La deuxième étape consiste à résoudre ces différents clusters indépendamment. Selon la détermination de la taille des cellules, trois différentes approches sont proposées :

— Dans la première, appelée approche verticale, la taille d'un cluster est défini par un intervalle de variables, donc à un sous-ensemble des variables. Dans ce cas, une clause peut appartenir à plusieurs clusters différents à condition qu'au moins une des variables qui la composent appartienne à l'intervalle définissant la cellule. Cependant, seules les variables appartenant à l'intervalle de la cellule sont considérées comme variables représentatives de cluster et sont par conséquent les seules à être instanciées.

— Dans la deuxième approche dite horizontale, la taille d'une cellule est définie par le nombre de clauses que peut admettre celle-ci sans aucune condition sur le nombre de variables. Toutes les variables appartenant à ce cluster sont instanciées lors de la résolution. Dans cette approche, une clause ne peut appartenir qu'à un et un seul cluster.

— La dernière proposition appelée approche mixte propose une combinaison entre les deux définitions précédentes. Le nombre de variables ainsi que celui des clauses sont fixés. Le cluster comprend alors un certain nombre de clauses, mais seules les variables définies dans l'intervalle de variables du cluster sont instanciées.

La figure 6.1 illustre les différents découpages de la grille proposés pour l'instance SAT UF20-01 (uniform random 3SAT, SAT'13) Benchmark. [a].

Le rectangle rouge représente le découpage de l'approche verticale où un intervalle de variables par cluster est défini. A l'opposé, le rectangle bleu limitant le nombre de clauses par cluster représente l'approche horizontale. Finalement, le rectangle ou carré noir illustre l'approche mixte et fixe le nombre de variables et de clauses par cellule.

Après création des clusters, la résolution de ces derniers est une étape

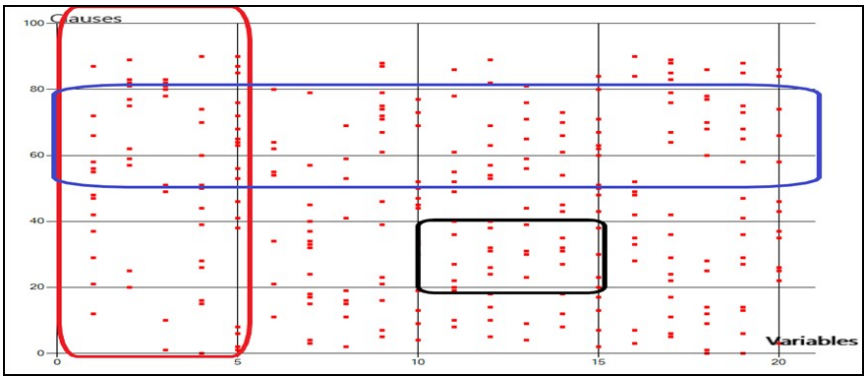


FIGURE 6.1 – *Adaptation de l'algorithme STING pour l'instance UF20-01*

cruciale car l'ordre de résolution peut influencer sur le résultat final et la satisfaction de l'instance globale.
Dans le but d'avoir la meilleure solution possible, nous proposons un tri de ces clusters selon leur densité. Cette densité est définie par la proportion du nombre de clauses par rapport au nombre de variables dans un cluster $D_c = \frac{Nbr_{Clauses}}{Nbr_{Variables}}$.
La raison de ce choix est l'impact que pourrait avoir l'assignation d'un nombre restreint de variables sur un nombre important de clauses. En effet, le tri étant réalisé dans l'ordre décroissant, les clusters ayant la plus haute densité sont résolus en premier.
Ces clusters sont résolus en utilisant l'algorithme DPLL et BSO selon le nombre de variables représentatives de cluster. Les clusters dont le nombre de variables à instancier est inférieur à un certain seuil sont résolus en utilisant le DPLL. L'algorithme BSO est exécuté dans le cas échéant. Il est à noter qu'après résolution d'un cluster, la solution de ce dernier est propagée sur le reste des clusters dans le but d'éliminer les clauses satisfaites par cette solution ainsi que les variables instanciées. Après résolution de tous les clusters, l'ensemble des solutions partielles obtenues sont combinées sans qu'il y'ait chevauchement entre les différentes solutions. Cette condition est satisfaite par le tri de densité effectué ainsi que la propagation des solutions de chaque cluster.

6.4 Expérimentations et Étude comparative

Dans le but de valider l'adaptation de l'algorithme STING présentée dans ce chapitre, plusieurs expérimentations ont été effectuées sur les instances définies précédemment. La machine sur laquelle ces tests ont été effectués est dotée d'un processeur i7 2.40Ghz avec 4GO de RAM.

6.4.1 Détermination des tailles des cellules

Dans notre adaptation de l'algorithme STING, la taille des cellules est définie au préalable. Trois différentes définitions de taille de cellule sont proposées dans ce chapitre. Afin de valider et de comparer ces approches,

un ensemble de tailles de cellules sont définies et représentées dans le tableau 6.1.

Taille de cellule	#variables / cluster	#clauses / cluster	Approche Verticale	Approche Horizontale	Approche Mixte
Smallest		Size / 32 (1708)		21	
Small	25	Size / 16 (3417)	388	11	867
Medium	50	Size / 8 (6835)	194	6	377
Large	100	Size / 2 (27341)	97	2	173

TABLE 6.1 – *Taille de cellules pour l'algorithme STING pour l'instance BMC-IBM1*

NB. NCI (Nombre de Clauses dans l' Instance).

Le tableau 6.1 illustre les différentes tailles de cellules considérées pour la comparaison entre les différentes approches. Le tableau indique également le nombre de clusters obtenus théoriquement pour l'instance BMC-IBM1.
Conventionnellement, les quatre différentes tailles de la plus petite à la plus large seront définies par les noms XS, X, M et L.
Dans l'approche verticale, le nombre de variables par cluster conditionne la taille de la cellule. Les propositions de tailles sont respectivement de 25, 50 et 100 variables par cluster pour les tailles S, M et L.
L'approche horizontale détermine la taille d'une cellule en fixant le nombre de clauses. Quatre différentes proportions du nombre de clauses par clusters sont définies dans cette approche. La plus grande taille L est obtenue en divisant le nombre de clauses par 2. Le reste des tailles sont définies en divisant le nombre de clauses par 8, 16 et 32 pour les clusters de taille M, S et XS respectivement.
L'approche mixte combine les deux précédentes définitions et fixe à la fois le nombre de clauses et de variables par cluster. Trois différentes tailles sont proposées S, M et L. Un cluster de taille S comprend 25 variables et un seizième (116) du nombre total de clauses dans l'instance. Une cellule de taille M est définie par un intervalle de 50 variables et une proportion de 1/8 de l'ensemble des clauses. Enfin, la taille L fixe le nombre de variables par cluster à 100 et le nombre de clauses à la moitié (1/2) de l'ensemble des clauses.

Le nombre de clusters résultants de ce découpage pour l'instance BMC-IBM1 n'est que théorique. En effet, l'application de la procédure de propagation unitaire ainsi que la propagation de littéraux pures (DPLL) avant le clustering permet une réduction du nombre de clauses et par conséquent du nombre de clusters.

6.4.2 STING : Approche verticale

Les figures suivantes (6.2 et 6.3) présentent les taux de satisfaction ainsi que les temps d'exécution de l'approche verticale de l'algorithme STING pour la résolution d'instances SAT.

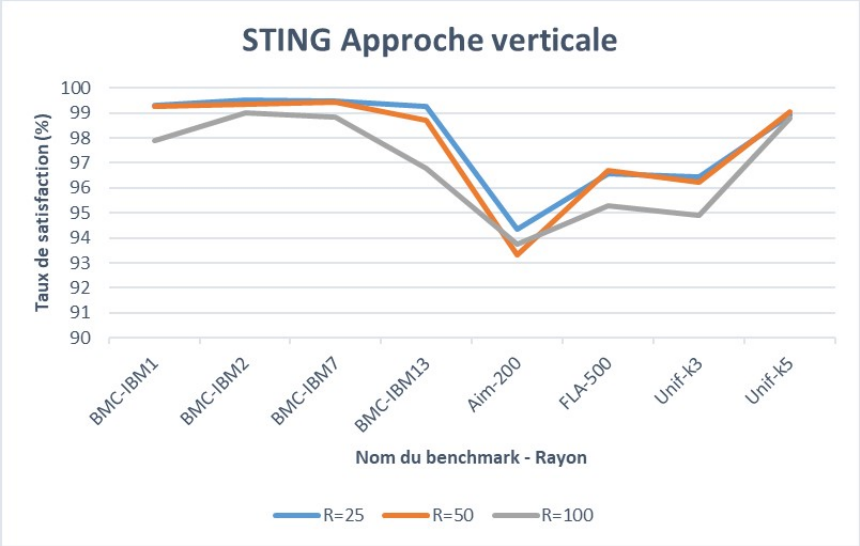


FIGURE 6.2 – *Taux de satisfaction d'instances SAT avec prétraitement STING-Vertical*

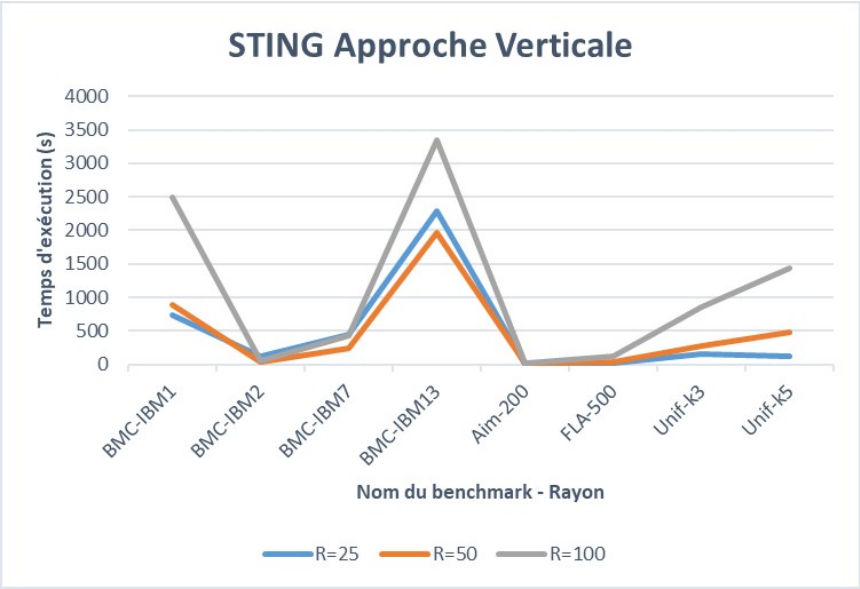


FIGURE 6.3 – *Temps d'exécution d'instances SAT avec prétraitement STING-Vertical*

La figure 6.2 expose les taux de satisfaction des instances après clustering par l'approche verticale de l'algorithme STING où R représente le nombre de variables dans un cluster. On observe une baisse du taux de satisfaction lors de l'augmentation de la taille du cluster. Cette diminution de l'efficacité de la résolution est due à l'augmentation du nombre de variables à instancier par cluster. En effet, les clusters dont le nombre

de variables représentatives ne dépasse pas un certain seuil sont résolus en utilisant l'algorithme DPLL. Ces clusters présentent de meilleures solutions que ceux résolus en exécutant l'algorithme BSO qui est moins effectif que DPLL.
Le temps d'exécution illustré dans la figure 6.3 diminue lorsque l'augmentation de la taille des clusters de la taille *S* à la taille *M*. En effet, on peut deviner que l'algorithme DPLL est exécuté pour la résolution des clusters de taille S alors que les clusters de taille *M* sont résolus en utilisant BSO. Le DPLL étant un algorithme complet, couvre toutes les solutions possibles avant de trouver la bonne solution. Une hausse du temps d'exécution est finalement observée dans la résolution des clusters de taille *L*. Cette augmentation est due à l'augmentation du nombre de variables à instancier.

6.4.3 STING : approche Horizontale

L'approche horizontale de l'algorithme STING considère le nombre de clauses dans la détermination de la taille d'un cluster. La figure 6.4 ainsi que le tableau 6.2 exhibent les résultats de cette approche.

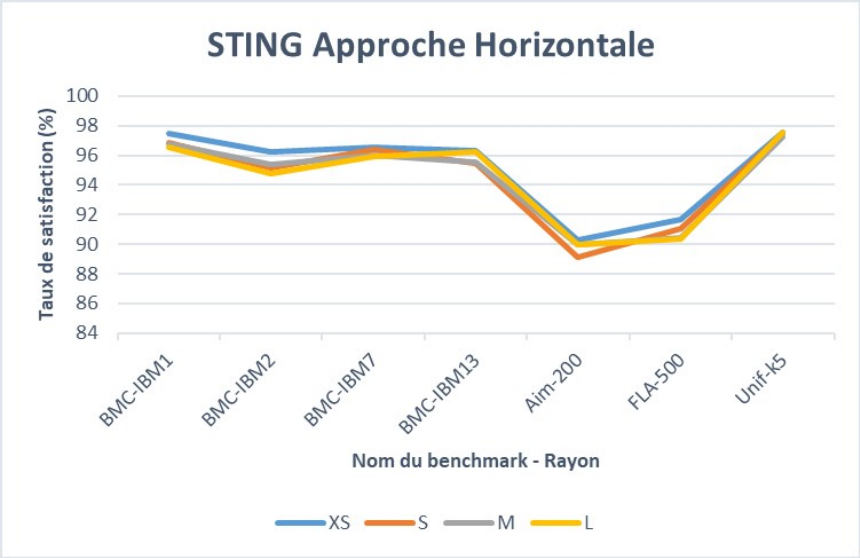


FIGURE 6.4 – *Taux de satisfaction d'instances SAT avec prétraitement STING-Horizontal*

Taille de cellule	Temps d'exécution (s)						
	IBM1	IBM2	IBM7	IBM13	AIM200	FLA500	Unif-K5
Smallest	269.43	15.26	73.06	263.18	1.76	3.58	11.71
Small	207.73	13.1	86.61	313.65	2.9	11.21	41.21
Medium	383.11	11.91	180.68	776.71	4.76	46.9	165.84
Large	2884.35	33.07	1296.43	6459.08	88.67	723.98	2335.96

TABLE 6.2 – *Temps d'exécution d'instances SAT avec prétraitement STING-Horizontal*

Remarque : L'instance Unif-k3 n'est pas représentée car les temps

d'exécution de cette dernière pour la taille *S* dépasse les 3600s.

On observe une diminution de la qualité des solutions lors de l'augmentation de la taille des clusters. En effet, dans cette adaptation, la taille d'un cluster est fixée par le nombre de clauses tandis que la qualité de solution dépend principalement du nombre de variables. Cependant, à partir d'un certain seuil, il existe une corrélation entre le nombre de clauses et celui des variables. Plus le nombre de clauses est important plus le nombre de variables l'est. Et plus ce nombre de variables est important plus la probabilité d'exécution du BSO l'est.

Les temps d'exécution présentés dans le tableau 6.2 montrent une faible variation pour les petites et moyennes tailles de cellules. Par ailleurs, une importante inflation du temps d'exécution est notée pour la taille de cluster la plus large. En effet, le temps d'exécution est proportionnel au nombre de variables à instancier dans le cluster. Le nombre de variables représentatives de cluster intervient directement sur le choix de l'algorithme de résolution.

6.4.4 STING : Approche Mixte

Les figures 6.5 et 6.6 indiquent les résultats en termes de taux de satisfaction et de temps d'exécution de la troisième et dernière adaptation de STING pour la résolution des instances SAT.

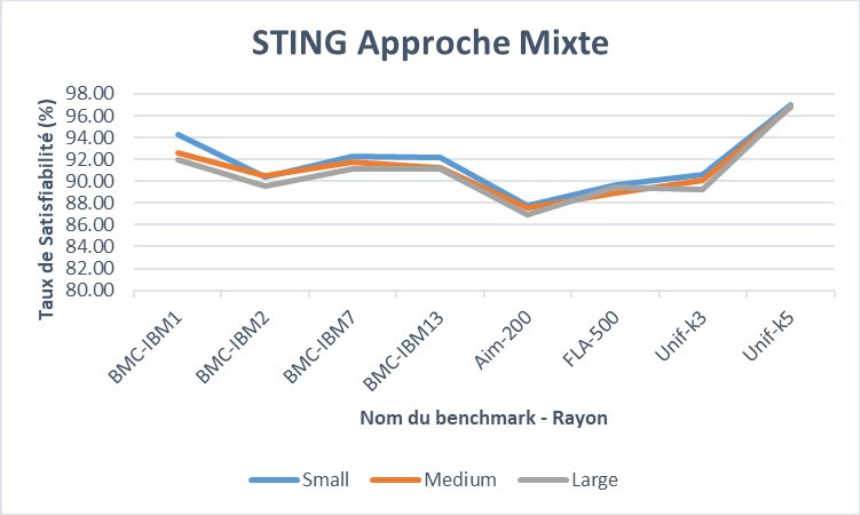


FIGURE 6.5 – *Taux de satisfaction des instances SAT avec prétraitement STING-Mixte*

On observe, à travers la figure 6.5, une décroissance du taux de satisfaction des instances après augmentation de la taille des clusters. En effet, l'augmentation du nombre de variables à instancier par cluster induit à une plus grande probabilité d'utilisation du BSO et par conséquent à une solution moins satisfaisante.

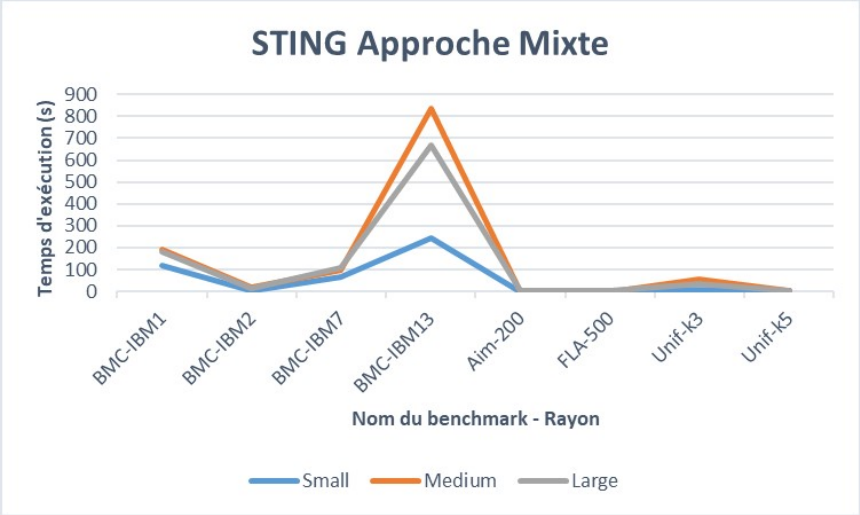


FIGURE 6.6 – *Temps de résolution des instances SAT avec prétraitement STING-Mixte*

La figure 6.6 illustrant les temps d'exécution expose une hausse des temps d'exécution lors de l'augmentation de la taille de cellule de *S* à *M*. Cette augmentation est due à l'accroissement du nombre de variables à instancier par cluster d'une part et celui des clauses lors de la validation des solutions d'autre part. Néanmoins, le temps d'exécution diminue une seconde fois après augmentation de la taille des cellules. Cette réduction du temps d'exécution s'explique par une propagation importante des solutions des clusters résolus. En effet, la propagation d'une solution partielle sur un autre cluster permet une réduction importante du nombre de clauses et de variables. Ce qui permet une réduction du temps d'exécution lors de la résolution de ce dernier.

6.4.5 Comparaison

Afin de comparer les trois différentes approches et définitions de cluster, le tableau 6.3 illustre les meilleurs taux de satisfaction des différentes approches avec leurs temps d'exécution.

Benchmark		STING adaptation		
		Approche verticale	Approche horizontale	Approche mixte
IBM1	Taux SAT (%)	99,28	97,42	94,23
	Temps exe (s)	742,03	383,11	115,85
IBM2	Taux SAT (%)	99,36	96,39	90,37
	Temps exe (s)	35,71	33,07	5,14
IBM7	Taux SAT (%)	99,46	96,71	92,23
	Temps exe (s)	438,61	180,68	65,36
IBM13	Taux SAT (%)	99,27	96,25	92,17
	Temps exe (s)	2293,77	593,47	244,38
AIM200	Taux SAT (%)	94,33	91,28	87,81
	Temps exe (s)	5,87	4,76	0,04
FLA500	Taux SAT (%)	96,68	90,65	89,65
	Temps exe (s)	29,82	3,58	0,11
Unif-k3	Taux SAT (%)	96.46	-	90.59
	Temps exe (s)	146.66	-	7.23
Unif-k5	Taux SAT (%)	99.03	97.44	96.97
	Temps exe (s)	482.19	41.27	0.65

TABLE 6.3 – *Comparaison entre les différentes approches de STING – Taux de satisfaction et temps d'exécution*

On remarque, compte tenu de ces résultats, que l'approche verticale définissant la taille d'une cellule en fonction du nombre maximal de variables que celle-ci peut contenir prodigue de meilleurs résultats que les deux autres en termes de taux de satisfaction. En effet, la fixation du nombre de variables qui est le principal paramètre lors de la résolution d'une instance SAT permet une meilleure instanciation de ces dernières, et ceci en tenant compte de toutes les clauses comportant ces variables, contrairement aux deux autres approches. Par ailleurs, le temps d'exécution de cette approche est bien plus important pour les instances dont le nombre de clauses et de variables est élevé.

6.5 Conclusion

À travers ce chapitre, nous avons proposé une adaptation de l'algorithme STING aux instances SAT dont la distribution est aléatoire et ne définit aucune région de densité considérable.
Contrairement à l'algorithme STING, notre adaptation propose de déterminer la taille des cellules avant de procéder au clustering. Cette étape permet un gain considérable de temps d'exécution.
Trois principales définitions de taille de cellule ont été proposées. L'approche verticale définit la taille d'une cellule par l'intervalle de variables que la cellule peut contenir. Cet intervalle représente alors les variables à instancier lors de la résolution du cluster. L'approche horizontale considère le nombre de clauses dans la définition de la taille d'un cluster. Dans ce cas, toutes les variables contenues dans le cluster sont désignées comme variables représentatives de celui-ci. Enfin, l'approche mixte combine les deux précédentes et fixe le nombre de variables ainsi que celui des clauses que peut contenir une cellule.

Des expérimentations ont été réalisées sur un ensemble d'instances connues. Les résultats de ces expérimentations ont conclu de l'efficacité de cette approche et notamment de l'approche verticale.

Approche de clustering basée sur les règles d'association 7

7.1 Introduction

Nous nous intéressons dans ce chapitre aux algorithmes d'extraction de modèles fréquents. Ces algorithmes représentent l'un des outils de datamining les plus utilisés dans l'extraction de connaissance et de corrélation entre les données et particulièrement dans la génération de règles d'association. L'un des algorithmes les plus basiques et populaires à la fois est l'algorithme Apriori. Il permet l'extraction d'ensembles d'items fréquents de tailles différentes.
Cependant, le fonctionnement de cet algorithme engendre un temps de calcul important. Dans le but de réduire ce temps de calcul, nous proposons dans un premier temps une amélioration à ce dernier basée sur le paradigme « *diviser pour régner* ». L'algorithme Méta-Apriori comprend alors trois phases. Dans la première, la base de données est scindée en deux parties voire plus si nécessaire. Une version améliorée de l'algorithme Apriori est alors exécutée pour chacun de ces sous-groupes. Finalement, les résultats sont combinés de façon à répondre à la requête initiale.
L'algorithme Méta-Apriori est ensuite utilisé comme algorithme de clustering pour les instances SAT dans le but de réduire la complexité de ces dernières et permettre une résolution plus efficace.
Une deuxième approche combinant l'algorithme Apriori et k-means est proposée dans ce chapitre. Cette hybridation se présente en deux phases. L'algorithme Apriori est d'abord utilisé dans le but d'extraire les variables les plus répétées ensemble. Cet ensemble est par la suite utilisé comme centres de clusters initiaux de l'algorithme k-means.

7.2 L'algorithme Apriori

Avant d'introduire l'algorithme Apriori, la définition de certaines notions est nécessaire à la compréhension et l'utilisation de cet algorithme, à savoir le **Support** et la **Confidence**.
Soit l'ensemble de transactions T : $t_1, t_2, \ldots, t_n$ où chaque transaction est formée d'un ou plusieurs items parmi l'ensemble $Ii_1, i_2, \ldots, i_m$. On notera par X ou Y, tout sous-ensemble de I comprenant un nombre défini

d'items.
Le Support $S(X,Y)$ désigne la probabilité d'apparition des sous-ensembles X et Y dans la même transaction $S(X,Y) = Proba(X \cap Y)$.
La Confidence C(X,Y) désigne quant à elle la probabilité conditionnelle d'avoir Y dans une transaction si X est présent dans celle-ci $C(X,Y) = Proba(Y|X)$.
Il est à noter que $C(X,Y) = \frac{S(X,Y)}{S(X)}$.
L'algorithme Apriori, consiste à déterminer et extraire à partir d'un ensemble de transactions, les ensembles d'items fréquents. Un itemset ou ensemble d'items est considéré comme fréquent si et seulement si son support est supérieur ou égal au support minimum fixé $MinSup$. En pratique, cela revient à calculer la fréquence d'apparition de cet itemset dans l'ensemble des transactions.
L'algorithme 10 illustre le fonctionnement de l'algorithme Apriori.

Algorithm 10 Algorithme Apriori

Données : TB : base de transactions
MinSup : minimum support
FPB : base des modèles fréquents
Début
Procedure Lecture(TB)
Parcourir toute la TB
Extraire les itemsets de taille 1 ainsi que leur fréquence
Extraire les itemsets de taille 2 ainsi que leur fréquence

Procedure Validation
Parcourir toute la TB
Calculer l'occurrence de chaque candidat
Si cette occurrence est supérieure à MinSup alors item est fréquent (ajouter à FPB)
Retourner FPB

Method
Lecture(TB)
Validation des itemsets de taille 1
Validation des itemsets de taille 2
Pour les tailles i supérieures à 2
Création des itemsets candidats de taille i : Jointure entre les itemsets de taille i-1
Validation des candidats
Passer à la taille supérieure jusqu'à ce qu'il n'y ait plus de plus grande taille d'itemset possible
Fin

Le fonctionnement de l'algorithme Apriori peut être résumé par les deux étapes suivantes : l'extraction des itemsets candidats et la validation des itemsets fréquents.

— Extraction des itemsets candidats. Pour les items de taille un (01), un scan de la base de transactions est effectué et tous les items sont

considérés comme candidat. Pour les items de taille $k \geq 2$, une jointure entre les itemsets fréquents de taille k-1 est réalisée.
— La phase de validation des itemsets fréquents consiste alors à scanner la base de transaction afin de calculer le support de chaque itemset candidat. Par ailleurs, la validation de tous les sous itemsets des itemsets candidats de taille $k > 2$ est nécessaire. En effet, pour qu'un itemset de taille $k \geq 2$ soit considéré comme fréquent, la propriété suivante « Tous les sous-items d'un itemset fréquent doivent également être fréquents » doit être vérifiée.

Les complexités temporale et spatiale de cet algorithme représentent les inconvénients majeurs à son utilisation. En effet, la nécessité d'effectuer de multiples scans de la base de transactions pour valider un ensemble d'itemsets fréquents engendre un temps de calcul considérable. Par ailleurs, le stockage de la base de transactions ainsi que celui des listes de candidats et d'itemsets fréquents engendre rapidement un dépassement de capacité mémoire pour les transactions volumineuses.
Afin de contourner ces limitations, nous proposons une version améliorée de l'algorithme Apriori dans la section suivante.

7.3 Méta-Apriori : Un nouvel algorithme basé sur Apriori et Expérimentations

Afin de contourner et de réduire les inconvénients relatifs au temps d'exécution et à l'espace mémoire utilisé par l'algorithme Apriori, une première amélioration est proposée dans cette section. Cette amélioration consiste à utiliser une structure de données adaptée ainsi qu'une limitation du nombre de scans effectués par l'algorithme.
Dans cette amélioration, une structure de données verticale est adoptée. En effet, pour chaque itemset candidat, l'algorithme Apriori parcourt l'ensemble de la base de transactions pour calculer le support de ce dernier. Une représentation judicieuse de cette base peut permettre une réduction considérable du temps d'exécution de l'algorithme. Cette structure a pour entrée l'indice d'un item et contient les transactions auxquelles l'item appartient. Cette configuration permet de réduire de manière significative le nombre d'accès à la base de transactions. La figure 7.1 illustre le passage de la représentation horizontale initiale de l'algorithme et la représentation proposée dans cette amélioration.

Cette structure est appliquée à la table de transactions ainsi qu'à l'ensemble des itemsets. On associe alors chaque item à tous les itemsets fréquents dont il fait partie.
Une limitation du nombre de parcours de la base de transactions est également proposée. Ainsi, au lieu de parcourir toutes les transactions pour déterminer le support d'un itemset, les limitations suivantes sont proposées :
— Valider la fréquence d'un itemset candidat en ne comptabilisant que les itemsets de l'item le moins fréquent dans la base de transactions. En effet, dans la structure proposée précédemment, l'accès

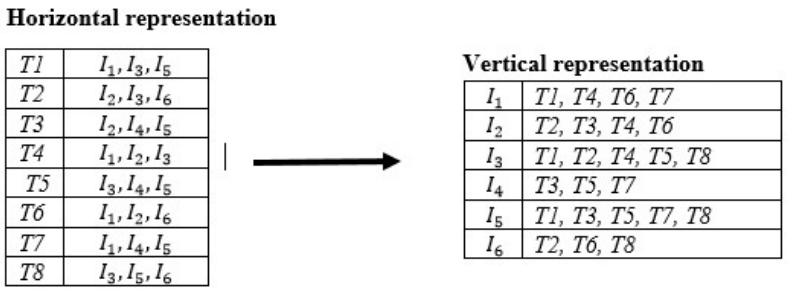


FIGURE 7.1 – *Adoption d'une représentation verticale*

à l'ensemble des itemsets contenant un item donné est direct à travers l'indice de cet item. Un itemset est dit fréquent si sa fréquence excède ou est égale au $MinSup$. Il est alors suffisant de vérifier un nombre minimum de transactions.

Exemple 7.1 :

Si l'on considère l'itemset de taille k=3 suivant $\{I_1, I_2, I_3\}$. Supposons que la fréquence d'apparition des items I_1, I_2 et I_3 soient respectivement de 15,19 et 7. Il est suffisant de vérifier la fréquence de l'itemset à travers l'indice de l'item I_3.

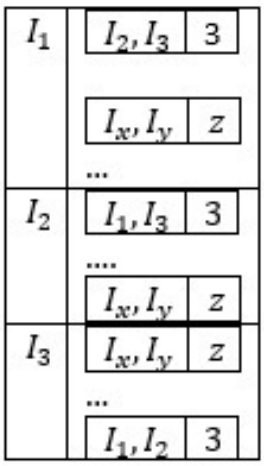


FIGURE 7.2 – *Structure de représentation des itemsets (exemple 7.1)*

— Valider un itemset candidat dès que son support atteint $MinSup$. En effet, si le support minimal à la validation d'un itemset est de 10, il n'est pas nécessaire d'aller au-delà pour valider un itemset.
— Éliminer de la base de transactions les items dont la fréquence d'apparition est inférieure au $MinSup$. Ces items ne peuvent pas être contenus dans des itemsets fréquents s'ils ne sont pas fréquents.
— Éliminer toutes les transactions dont le nombre d'items est inférieur à k. k étant le nombre d'items de l'itemset courant. Si par exemple une transaction ne contient que 3 items, ce serait une perte de temps de la considérer pour valider un itemset candidat de taille 4 voire plus.

Toutes ces améliorations sont intégrées dans l'algorithme Méta-Apriori

Benhamouda et al. [2016]. Cet algorithme répond au paradigme « *diviser pour régner* » et comprend trois principales phases. La première phase consiste à diviser l'ensemble des transactions en deux parties voire plus si nécessaire. L'algorithme Apriori avec toutes les améliorations citées précédemment est exécuté dans la seconde étape et permet d'extraire les itemsets fréquents de chacun des sous-ensembles de transactions. Enfin, les différents itemsets fréquents sont combinés.

Étape de partitionnement :

Cette première phase permet de diviser la base de transactions en deux voire plusieurs parties dans le but de réduire sa complexité et de permettre une meilleure efficience de l'algorithme. Deux méthodes de partitionnement sont proposées.

Le partitionnement aléatoire

Nous proposons pour ce partitionnement d'assigner successivement les transactions aux différents sous-groupes. La figure 7.3 illustre le fonctionnement de ce partitionnement sur un ensemble de transactions. Ces transactions sont divisées en deux groupes.

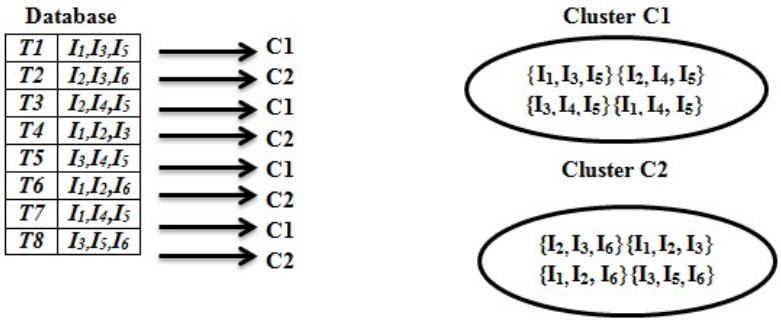


FIGURE 7.3 – *Partitionnement aléatoire*

Le partitionnement basé sur la fréquence

Dans le partitionnement basé sur la fréquence, la dispersion des items dans les sous-bases de transactions est le principal paramètre à la division de la base. L'objectif est d'avoir la même fréquence d'items dans chacun des différents sous-groupes de transactions.
La figure 7.4 illustre ce partitionnement.

Si l'on considère la subdivision en deux groupes, les deux premières transactions sont d'abord assignées aux deux sous-groupes de transactions. Puis, pour chaque transaction, la fréquence d'apparition dans un cluster définie par la somme des fréquences des différents items de la

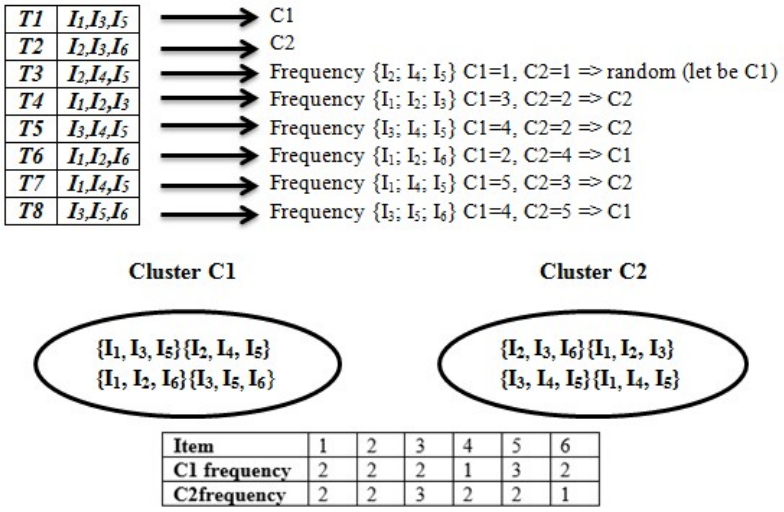


FIGURE 7.4 – *Partitionnement basé sur la fréquence*

transaction est calculée pour chaque cluster. Par exemple, la fréquence de la transaction $T_7\{I_1, I_4, I_5\}$ dans C_1 est de 5 car le nombre d'apparitions des items I_1, I_4 et I_5 sont de respectivement 2,1 et 2. La transaction est assignée au cluster dont la fréquence d'apparition de celle-ci est moins importante.

Les deux étapes d'exécution de l'algorithme Apriori avec les améliorations proposées ainsi que la fusion des itemsets résultants sont exécutées.
Remarque : Dans certains cas et particulièrement dans le cas du partitionnement basé sur la fréquence, les deux ensembles d'itemsets fréquents peuvent être identiques. La fusion est alors inutile.

La figure 7.5 illustre une comparaison du fonctionnement de l'algorithme Méta-Apriori avec l'algorithme Apriori.

Dans le but de prouver l'efficacité de l'algorithme Méta-Apriori, des tests ont été effectués sur une base de transaction de Repository. ainsi que les instances SAT IBM2 et IBM7.
Le tableau 7.1 introduit la description de ces bases de transactions en termes de nombre d'items et de transactions.
Remarque : Les clauses et les variables des instances SAT sont considérées comme transactions et items.

Nom du benchmark	Nombre d'items	Nombre de transactions
IBM2	2810	11266
IBM7	8710	39374
T10I4D100K(Fimi)	999	100000

TABLE 7.1 – *Table de transactions*

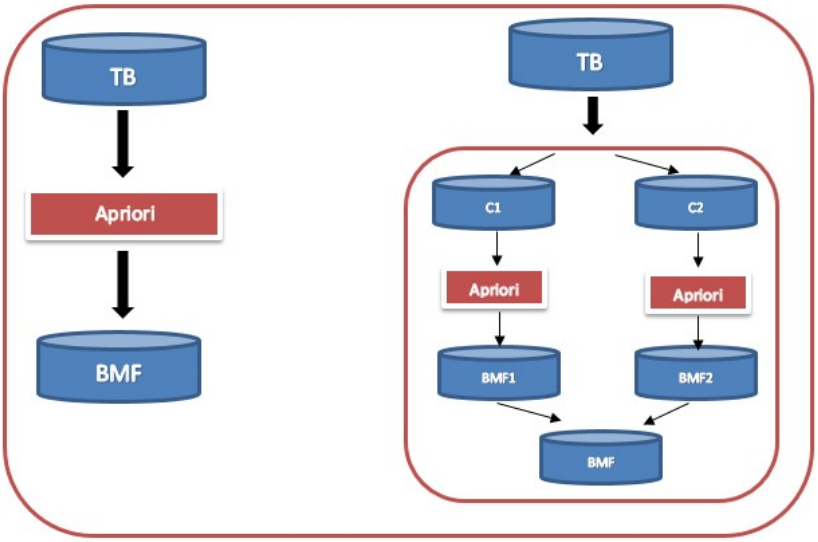


FIGURE 7.5 – *Apriori vs Méta-Apriori*

7.3.1 Apriori vs Improved Apriori

Afin d'approuver les améliorations apportées à l'algorithme Apriori, le tableau 7.2 exhibe les résultats en termes de nombre d'itemsets et de temps d'exécution des algorithmes Apriori et son amélioration.

Benchmark	Apriori		Improved Apriori	
	Nombre d'itemsets	Temps d'exécution (s)	Nombre d'itemsets	Temps d'exécution (s)
Ibm2	3824	1,803305	3824	0,3300005
Ibm7	11510	55,05679	11510	9,7610137
Fimi	27521	4340,222	27521	2008,427241

TABLE 7.2 – *Apriori vs Improved Apriori*

A travers ces résultats, on note une importante réduction du temps d'exécution lors de l'utilisation de la version améliorée de l'algorithme Apriori. Les ensembles d'itemsets fréquents sont identiques pour les deux algorithmes prouvant l'efficacité des améliorations apportées.

7.3.2 Méta-Apriori

Le tableau ci-après (7.3) introduit les résultats en termes de nombre d'itemsets fréquents et de temps d'exécution des deux approches de l'algorithme Méta-Apriori, à savoir l'approche dont le partitionnement est aléatoire et celle dont le partitionnement est basé sur la fréquence.

Benchmark	Méta-Apriori Aléatoire		Méta-Apriori Fréquence	
	Nombre d'itemsets	Temps d'exécution (s)	Nombre d'itemsets	Temps d'exécution (s)
Ibm2	3824	0,2053877	3824	0,202613
Ibm7	11510	5,8213687	11510	8,076405
Fimi	27521	1980,064836	27521	1995,84

TABLE 7.3 – *Méta-Apriori*

On remarque que les résultats obtenus par les deux approches en termes de nombre d'itemsets fréquents sont identiques et cohérents avec les résultats obtenus en exécutant l'algorithme Apriori. Le temps d'exécution est quasi identique dans les deux approches.
L'efficacité de l'algorithme Méta-Apriori étant prouvée par le fait que tous les ensembles itemsets fréquents soient identiques. Une comparaison des temps d'exécution de l'algorithme Méta-Apriori, l'algorithme Apriori et de l'amélioration proposée à ce dernier est illustrée par la figure 7.6.

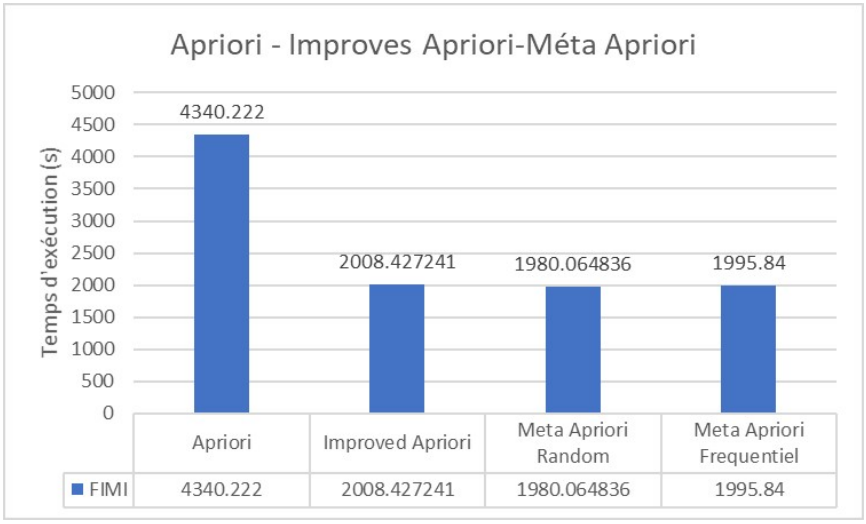


FIGURE 7.6 – *Comparaison des temps d'exécution : Apriori – Improved Apriori – Méta Apriori*

La figure 7.6, exhibe la comparaison des temps d'exécution des différents algorithmes proposés dans cette section appliqués sur la base de transaction Fimi. Une nette amélioration du temps d'exécution est observée prouvant l'efficience des améliorations apportées à l'algorithme Apriori et particulièrement l'algorithme Méta-Apriori.

7.4 Application de Méta Apriori pour les instances SAT

Dans le but de réduire la complexité des instances SAT et de permettre une meilleure résolution de ces dernières, nous proposons l'utilisation de l'algorithme Méta-Apriori comme prétraitement avant la résolution de ces instances Hirèche et al. [2017].

Après comparaison des deux différentes approches de l'algorithme Méta-Apriori, à savoir l'approche de partitionnement aléatoire et celle du partitionnement basé sur la fréquence. Il est à noter que le partitionnement basé sur la fréquence permet d'avoir des ensembles de transactions quasi identiques à la base de transactions initiale, en termes de fréquence des items. Dans ce qui suit, nous allons utiliser l'algorithme Méta-Apriori avec partitionnement basé sur la fréquence.
L'idée générale dans l'utilisation de l'algorithme Méta-Apriori pour les instances SAT, est d'exécuter ce dernier dans le but d'extraire les variables les plus répétées ensemble. Ces variables sont alors considérées comme variables représentatives de clusters. Chaque clause est alors assignée au cluster dont le centre est le plus proche en utilisant la distance de *Hamming*.
Chaque cluster est résolu indépendamment en exécutant l'algorithme DPLL ou BSO en fonction du nombre de variables à assigner. Si ce nombre ne dépasse pas un certain seuil, l'algorithme DPLL est exécuté. Dans le cas échéant, la résolution est assurée par l'algorithme BSO.
Deux différentes adaptations de l'algorithme Méta-Apriori pour les instances SAT sont proposées. La figure 7.7 décrit ces propositions.

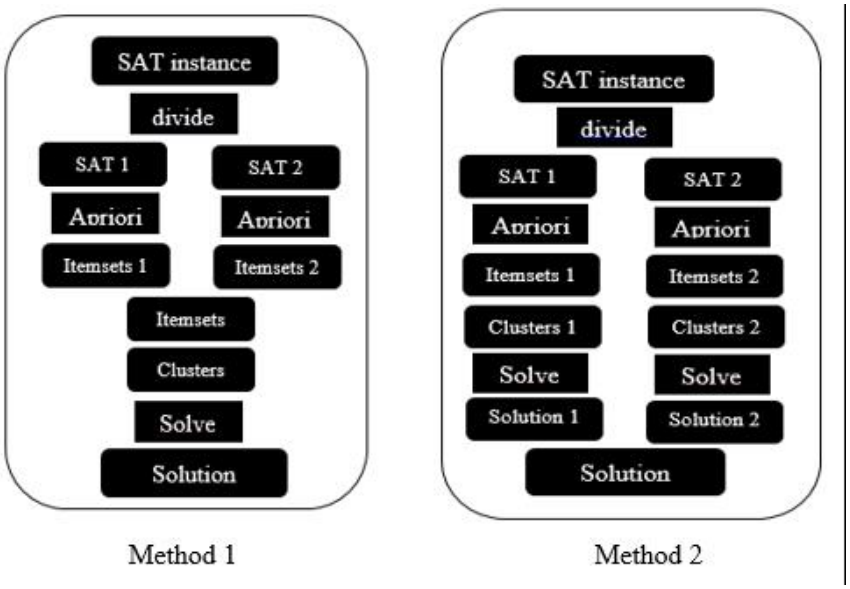


FIGURE 7.7 – *Méta-Apriori pour la résolution d'instances SAT*

La différence entre ces deux approches réside dans l'étape de fusion. Dans la première approche, les deux sous-ensembles de variables les plus répétées ensemble sont fusionnés en un seul. Chaque ensemble de ces variables représente alors les variables représentatives de clusters qui seront assignées dans la phase de résolution.
A l'opposé, la seconde approche ne considère pas la fusion entre les ensembles de variables les plus répétées ensemble. Chaque sous-ensemble de variables permet alors au clustering de la sous-instance SAT subdivisée initialement. Une fois les deux sous-instances résolues, une fusion des solutions de ces dernières est réalisée.

Dans le but d'évaluer l'utilisation de l'algorithme Méta-Apriori, des tests ont été effectués sur les benchmarks suivants Benchmark. [b] (tableau 7.4).

Nom du Benchmark	Nombre de clauses	Nombre de variables
Benchmark 1	99	8691
Benchmark 2	230	9975
Benchmark 3	440	9291
Benchmark 4	240	10409
Benchmark 5	260	11276

TABLE 7.4 – *Description des benchmarks de test*

Le tableau 7.4, présente une description des différents benchmarks utilisés pour démontrer l'efficacité de l'approche Méta-Apriori dans la résolution du problème SAT.

Benchmark Name	Method	Taux de satisfaction (%)	Temps d'exécution (s)
Benchmark 1	Method 1	99.61	37.01
	Method 2	99.64	41.35
Benchmark 2	Method 1	99.10	1.63
	Method 2	98.84	6.88
Benchmark 3	Method 1	97.71	2.20
	Method 2	98.30	0.66
Benchmark 4	Method 1	99.05	20.10
	Method 2	99.22	23.76
Benchmark 5	Method 1	99.20	23.29
	Method 2	99.27	27.66

TABLE 7.5 – *Méta-Apriori pour instances SAT – Taux de satisfaction et temps d'exécution*

Le tableau 7.5 exhibe les taux de satisfaction ainsi que les temps d'exécution des instances SAT résolues après exécution de l'algorithme Méta-Apriori comme prétraitement.
On observe que dans la majorité des cas, l'approche dont le partitionnement est basé sur la fréquence (method 2) prodigue de meilleurs résultats en termes de taux de satisfaction que celle dont le partitionnement est réalisé aléatoirement (method 1). Cependant, l'approche basée sur le partitionnement aléatoire est plus efficiente.
Certaines instances parmi celles utilisées sont résolues dans la littérature. Néanmoins, le temps de résolution de ces dernières est assez important. A titre d'exemple, le temps de résolution du benchmark 3 est de l'ordre de 2000 secondes alors que ce dernier est satisfait 98On peut conclure de l'efficacité et efficience de l'adaptation de l'algorithme Méta-Apriori pour les instances SAT.

7.5 Apriori-k-means pour les instances SAT

Dans cette section, une hybridation de l'algorithme Apriori avec le k-means Hireche et Drias [2019] est proposée pour les instances dont la distribution des variables sur les clauses est aléatoire et qui présentent la particularité que leurs variables aient une importante fréquence. Cette particularité nous fait penser à l'algorithme Apriori car ces variables fréquentes peuvent être associées de manière significative.
Cette hybridation des algorithmes Apriori et k-means se déroule en deux phases. L'algorithme Apriori est exécuté à chaque itération dans le but de produire les variables les plus répétées ensemble. Ces variables forment alors les *centroïds* initiaux de l'algorithme k-means qui est à son tour exécuté.
La figure 7.8 illustre le fonctionnement de cet algorithme.

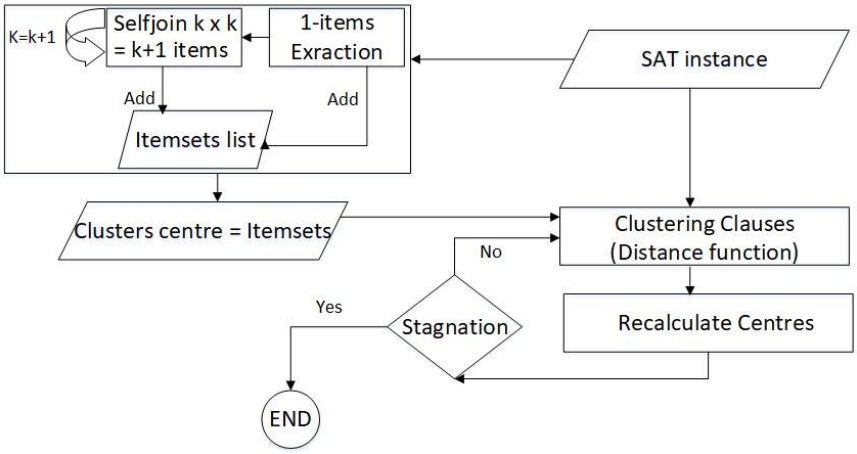


Figure 7.8 – *L'algorithme Apriori-KM*

Remarque : L'algorithme Apriori-k-means est réitéré autant de fois que nécessaire et jusqu'à ce que toutes les clauses appartiennent à un cluster. Le **Support Minmum (MinSup)** est dynamique et décroissant à chaque itération.
Une adaptation de l'algorithme k-means est nécessaire. En effet, les clauses n'ayant pas toutes la même taille dans un cluster, il est difficile de calculer les nouveaux centres de gravité des clusters. Nous proposons alors la fonction décrite comme suit :

- La taille du nouveau centre de cluster est calculée et représente la taille moyenne des clauses du cluster.
- Toutes les variables incluses dans un cluster sont triées par ordre croissant, en répétant chaque variable autant de fois que sa fréquence dans le cluster.
- Ces variables sont partitionnées en R parties distinctes, où R est la taille du nouveau *centroïd*.
- La moyenne de chacune de ces parties est calculée.
- Ces différentes moyennes sont fusionnées pour former le nouveau *centroïd*.

Exemple 7.1 :

Soit le cluster suivant :

$$\begin{cases} Centrod: & C = x_1, x_2 \\ Clauses & \begin{cases} C_1 = x_1, x_2, x_3, x_4 \\ C_2 = x_1, x_2, x_5 \\ C_3 = x_2, x_4, x_5 \\ C_4 = x_2, x_4 \end{cases} \end{cases}$$

La taille du nouveau *centroïd* est : $Size_(Newcentroid) = \frac{\sum_{x \in Clauses} Size_{C_x}}{|Clauses|} = \frac{4+3+3+2}{4} = 3$.
Les variables du cluster sont divisées en trois (03) sous-ensembles : $R_1 = \{x_1, x_1, x_2, x_2\}; R_2 = \{x_2, x_2, x_3, x_4\}; R_3 = \{x_4, x_4, x_5, x_5\}$.
La moyenne de chacun de ces sous-ensembles est calculée et considérée dans le nouveau *centroïd* ; $NewCentoid : C_New = x_1, x_2, x_4$.

Comme pour les approches précédentes, la résolution de ces clusters est effectuée en exécutant les algorithmes DPLL et BSO en fonction du nombre de variables à instancier par cluster. Dans cette approche, seules les variables des *centroïds* sont instanciées.

Dans le but de démontrer l'efficacité de l'algorithme Apriori-KM, une comparaison avec les algorithmes k-means et *CLARANS* est proposée.
La figure 7.9 ainsi que le tableau 7.6 indiquent les taux de satisfaction et temps d'exécution de ces différentes approches.

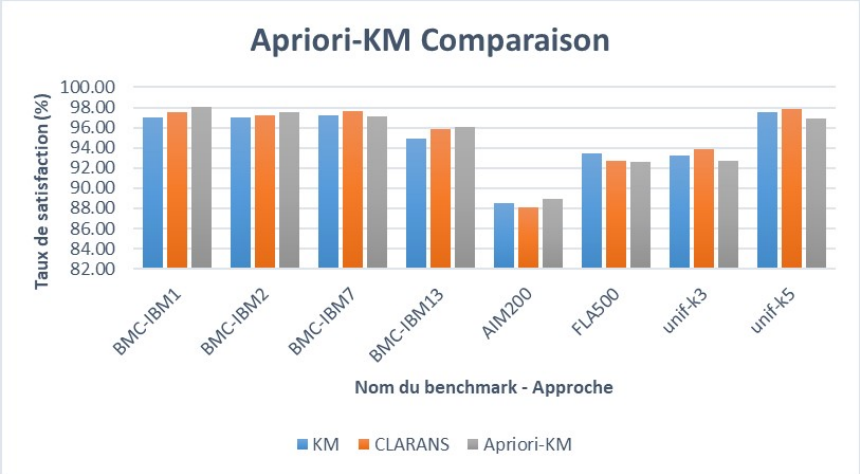


FIGURE 7.9 – *Apriori-KM, k-means et CLARANS – Taux de Satisfaction*

Benchmark	Temps d'exécution (s)		
	KM	CLARANS	Apriori-KM
IBM1	189.78	1244.57	277.91
IBM2	3.05	367.76	7.21
IBM7	57.29	2537.5	2421
IBM13	722.02	5294.22	179.51
AIM200	0.45	15.7	0.3
FLA500	0.23	47.98	0.28

TABLE 7.6 – *Apriori-KM, k-means et CLARANS – Temps d'exécution*

On observe à travers les résultats précédents que l'algorithme *CLARANS* offre de meilleurs résultats que le k-means en termes de satisfaction de clauses. Néanmoins, il reste moins efficient que le k-means avec un temps d'exécution considérable.
L'algorithme Apriori-k-means proposé fournit de meilleurs résultats que l'algorithme k-means voire même *CLARANS*. Cependant, il reste moins efficient que le k-means dans la majorité des cas. Cette élévation du temps d'exécution est principalement due à la dépendance de l'algorithme au nombre de variables et l'interconnexion entre celles-ci lors de l'extraction des variables les plus répétées ensemble dans la phase Apriori.

7.6 Conclusion

Tout au long de ce chapitre, nous avons proposé des approches de clustering pour les instances SAT basées sur l'extraction de modèles fréquents.
Une amélioration de l'algorithme Apriori a d'abord été proposée. Cette proposition considère une optimisation des structures de données utilisées par l'algorithme ainsi que des limitations sur le nombre d'analyses de la base de transactions. Ces différentes améliorations ont par la suite été intégrées dans un nouvel algorithme basé sur le paradigme « *diviser pour régner* » que nous avons nommé Méta-Apriori. Cet algorithme se définit en trois étapes. La première étape a pour but de diviser la base des transactions en deux voire plusieurs parties. La version améliorée de l'algorithme Apriori est alors exécutée sur chacune de ces parties. Enfin, les ensembles d'itemsets résultants sont combinés.
L'algorithme Méta-Apriori a été appliqué sur les instances SAT. Les ensembles des variables les plus fréquentes ont été utilisés comme variables représentatives de clusters. Chacun de ces clusters a par la suite été résolu en utilisant l'algorithme DPLL ou BSO selon le nombre de variables à instancier. Les expérimentations réalisées sur des instances SAT connues ont permis de démontrer l'efficacité de cette approche.
Une deuxième approche utilisant l'algorithme Apriori est d'associer ce dernier au k-means afin de tirer les bénéfices du clustering basé sur le partitionnement et d'en améliorer l'efficience. L'algorithme Apriori est alors utilisé comme prétraitement à l'algorithme k-means en extrayant les ensembles de variables les plus fréquemment répétées ensemble. Ces variables représentent alors les *centroïds* initiaux du k-means.
Cette approche a fait ses preuves face à l'algorithme k-means ainsi que *CLARANS* en termes de taux de satisfactions des instances. Par ailleurs, l'efficience de l'approche est relative au nombre de variables ainsi qu'aux interconnexions entre ces dernières.
Pour conclure, il est prouvé que l'exploitation des techniques et algorithmes d'extraction de modèles fréquents et l'utilisation de ces derniers comme prétraitement avant la résolution d'instances SAT permet d'avoir de bien meilleurs résultats.

Calcul GPU pour la résolution de SAT

8

8.1 Introduction

Dans ce chapitre, nous proposons d'exploiter la technologie offerte par le calcul parallèle -GPU- dans le but d'améliorer les performances des différentes approches précédemment proposées.
Ce travail s'articule alors en deux étapes dont la première consiste à générer les sous-instances SAT en exécutant les différentes adaptations des algorithmes DBSCAN et STING. Les clusters résultants et dont le nombre de variables ne permet pas leurs résolutions en utilisant un algorithme complet tel que le DPLL sont résolus en exécutant une version parallèle du BSO.
Le but de l'utilisation du GPU est d'améliorer les temps de résolutions des instances et de permettre une efficience des solutions.

8.2 Algorithme BSO parallèle

Bien que les métaheuristiques et les algorithmes de résolutions incomplets permettent une réduction du temps de résolution considérable en comparaison avec les algorithmes de résolutions complets tels que le DPLL. Le nombre important de variables à instancier combiné au nombre d'itérations nécessaires pour la génération d'une solution satisfaisante induit à une perte de l'efficience de l'algorithme de résolution.
Dans le but de réduire ce temps de résolution, nous nous sommes intéressés à la technologie proposée par les cartes graphiques programmables NVIDIA qui permettent le calcul parallèle.

Ce parallélisme n'est appliqué que pour l'algorithme BSO. L'algorithme DPLL étant exécuté que pour les clusters dont le nombre de variables à instancier ne dépasse pas un certain seuil, permet d'avoir des solutions exactes avec des temps de résolutions raisonnables et par conséquent ne nécessite pas de calcul parallèle.

Parallèle BSO :

Pour rappel, l'algorithme BSO est composé des principales étapes suivantes :

— Création d'une solution de référence aléatoire par une abeille initiale.
— Génération d'une zone de recherche de n solutions correspondant aux n abeilles, chaque solution est affectée à une abeille.
— Exécution d'une recherche locale SLS par chacune des abeilles avec retour de la meilleure solution de chaque abeille dans la table danse.
— Sélection de la meilleure solution de la table danse comme solution de référence et reprise du processus de recherche.

Dans le passage de la version séquentielle à la version parallèle du BSO, les boucles ainsi que la génération des solutions se font en parallèle à l'aide des différents threads du GPU. Le schéma général de l'algorithme BSO parallèle est présenté dans la figure 8.1 :

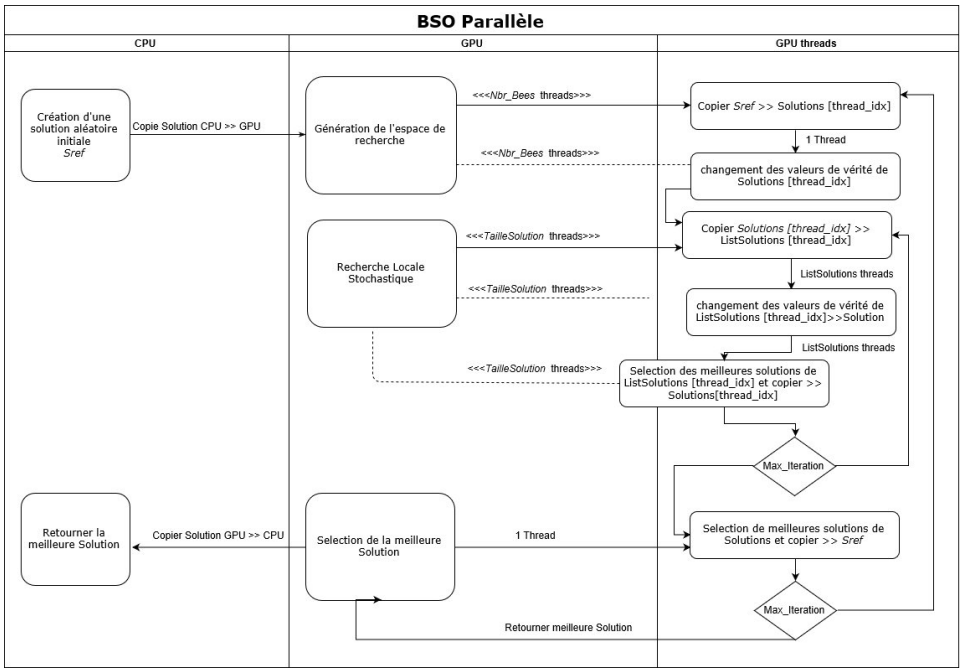


FIGURE 8.1 – *Algorithme BSO parallèle*

Génération de l'espace de recherche

Dans la version séquentielle de BSO, la génération de cet espace de recherche de n solutions à partir d'une sol*Sref*, se fait en effectuant n itérations dans lesquelles on copie la solution et on change la valeur de vérité de certaines variables selon un intervalle donné. La complexité de cette opération est alors de l'ordre de $\theta(n)$.
Dans la version parallèle du BSO, la complexité de cette opération est de l'ordre de $\theta(1)$. En effet, le calcul GPU permet d'effectuer simultanémentla même opération sur plusieurs données différentes simultanément.

Dans la génération de l'espace de recherche, chaque thread prend en charge une seule solution. L'algorithme 11 présente le fonctionnement de la fonction parallèle de génération de l'espace de recherche.

Algorithm 11 Procédure parallèle de génération de l'espace de recherche

Données : Sref : Solution initiale
n : Nombre de solutions à générer
SearchArea : Liste de solutions
Début
Kernel $copySolutionToSolutions <<< 1, n >>> (Sref, SearchArea)$
Kernel $SearchAreaGeneration <<< 1, n >>> (SearchArea)$
Fin

Kernel copySolutionToSolutions (Sref,SearchArea)
Données : Sref : Solution initiale
SearchArea : Liste de solutions
Début
Initialiser l'index de la première case mémoire du thread utilisé sur GPU

Copier la solution Sref à cette adresse mémoire (thread)
Fin

Kernel SearchAreaGeneration (SearchArea)
Données : SearchArea : Liste de solutions
SearchArea : Liste de solutions
Début
Initialiser l'index de la première case mémoire du thread utilisé sur GPU (indice)
Tant que l'indice est inférieur à la taille de la solution
  Changer la valeur de la variable à l'adresse Indice
  Incrémenter l'adresse (indice)
Fin

Les fonctions copySolutionToSolution et SearchAreaGeneration permettent respectivement de faire une copie d'une solution et de changer la valeur de vérité d'une ou plusieurs variables d'une même solution selon un certain intervalle.
La syntaxe « $NomKernel <<< x, y >>> (Paramtres)$ » permet d'indiquer le nombre de blocs ainsi que le nombre de threads par blocs à exécuter la même fonction sur les différentes données. Dans les fonctions précédentes, le nombre de threads est de n, correspondant au nombre de solutions de l'espace de recherche.
Chaque thread prend alors en compte une et une seule solution et permet une réduction considérable de la complexité de l'algorithme.

Recherche locale

Le schéma général de la recherche locale, effectuée par chaque abeille, est de prendre en considérable une solution et d'en créer son voisinage. Le voisinage de cette solution comprend n solutions où n est égale à la taille de cette solution, et où chacune de ces solutions ne diffère que d'une seule valeur de vérité avec la solution initiale. La meilleure solution de ce voisinage est alors considérée comme solution initiale et le processus reprend jusqu'à une certaine itération.
La recherche locale parallèle permet la création des différentes solutions du voisinage simultanément en exécutant n threads où chaque thread copie la solution initiale et change la valeur de vérité d'une et une seule variable. Une fonction d'évaluation est exécutée à chaque création de nouvelle solution permettant de sélectionner la meilleure solution par la suite.

Algorithm 12 Procédure parallèle de recherche locale

Données : Sref : Solution initiale
n : Nombre d'abeilles
Neighbourhood : Liste de liste de solutions
Début
Kernel : Créer une liste de solutions identiques à partir de la solution initiale
Kernel *RechercheLocale* $<<<1, n>>>$ (*Neighbourhood*)
Kernel Choisir la solution la mieux évaluée (utiliser l'adresse utilisée par le thread directement)
Fin

Kernel RechercheLocale (Neighbourhood)
Données : Neighbourhood[i] : Liste de solutions
Début
Initialiser l'index de la première case mémoire du thread utilisé sur GPU

Changer la valeur de la variable dont l'index dans la solution correspond à l'indice mémoire de la solution (thread)
Evaluer cette solution
Fin

Cette modélisation et parallélisme de l'algorithme BSO sur CPU/GPU permet une réduction du temps de résolution des différents clusters ce qui représente un des principaux buts de ce travail.

8.3 Application de l'algorithme BSO parallèle dans la résolution des clusters SAT

Deux différents modèles de résolution ont été proposés en fonction du type de clustering précédent la résolution. Dans le premier modèle, une résolution parallèle des clusters est réalisée après création des clusters. Contrairement à la deuxième approche, séquentielle où chaque sous-

instance est résolue de manière parallèle après sa création et avant la génération d'un nouveau cluster.

8.3.1 Approche de résolution parallèle Parallèle-BSO

Cette approche s'articule en trois phases distinctes. Dans la première étape, les différents clusters sont créés en utilisant l'algorithme DBSCAN multidimensionnel ou l'algorithme STING. Le premier permet la création de sous instances SAT disjointes car il ne considère pas la distance entre les variables, mais entre les clauses lors de l'extraction des différents clusters. Le clustering par l'algorithme STING permet la création de groupes de clauses totalement disjoints où chaque grille représente une sous-instance. Après création des différents clusters, une phase de résolution est exécutée. Les clusters dont le nombre de variables à instancier ne dépasse pas un certain seuil sont résolus de façon séquentielle (CPU) par le DPLL. Les instances avec un nombre de variables plus important sont résolues simultanément (CPU/GPU) par l'algorithme parallèle-BSO.
Une troisième étape de combinaison des différentes solutions est alors exécutée.
L'approche de clustering STRING verticale permet la création de clusters en délimitant le nombre de variables par clusters, dans ce cas la combinaison des solutions se fait de manière intuitive en concaténant les différentes solutions. Cependant, la résolution parallèle des clusters obtenus par les autres techniques de clustering peut engendrer un chevauchement des différentes solutions partielles. Dans ce cas, à chaque fois qu'une variable possède deux valeurs différentes, la valeur qui maximise le nombre de clauses satisfaites est assignée à la solution globale.
La figure 8.2 résume le schéma de résolution de cette approche.

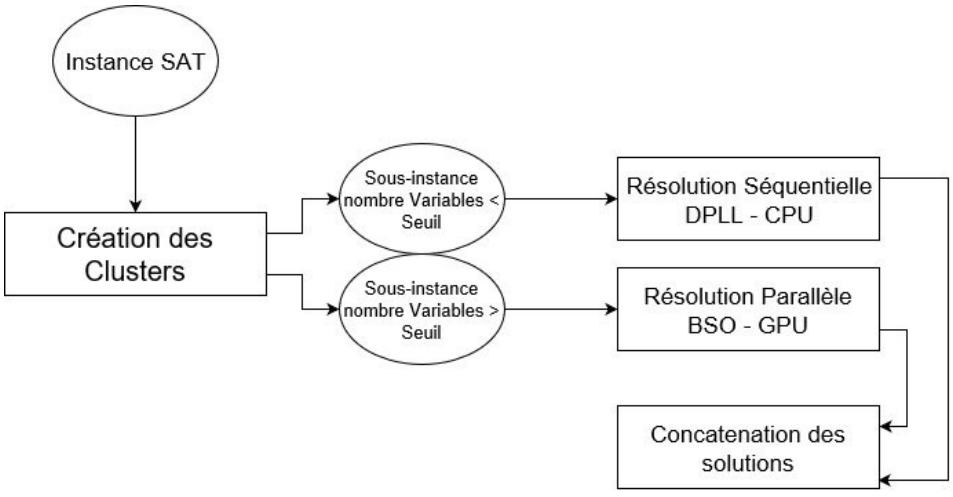


FIGURE 8.2 – *Approche de résolution parallèle*

8.3.2 Approche de résolution séquentielle Parallèle-BSO

Contrairement à la première approche, dans cette approche, les différentes sous-étapes sont combinées en une et même étape où à chaque fois

qu'un cluster est créé, ce dernier est résolu soit par l'algorithme DPLL ou par l'algorithme Parallèle-BSO. Cette approche est utilisée lors du clustering DBSCAN bidimensionnel.
Dans l'approche de clustering DBSCAN bidimensionnel, les variables représentent le principal paramètre pris en compte lors de la génération de clusters. La propagation d'une solution partielle sur le reste des clusters avant leurs résolutions permet alors l'obtention de meilleures solutions en termes de nombre de clauses satisfaites.
La figure 8.3 illustre le fonctionnement de l'approche de résolution séquentielle utilisant l'algorithme parallèle-BSO.

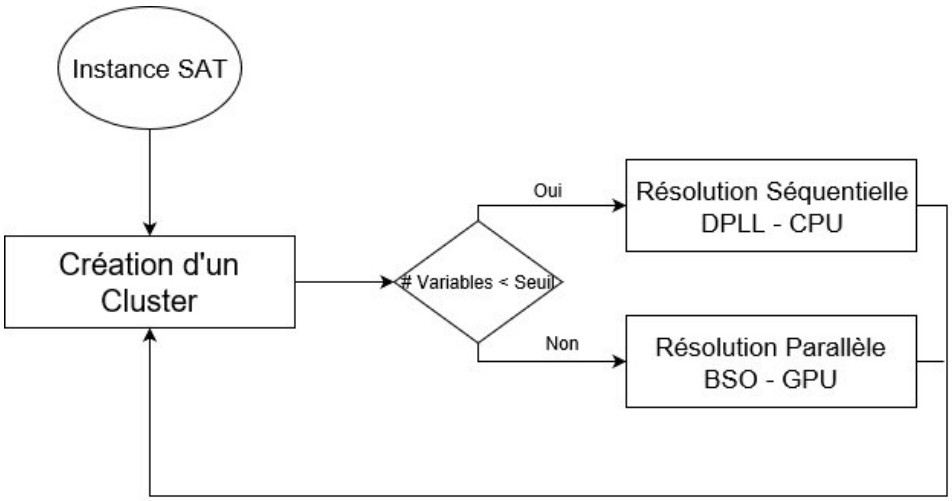


Figure 8.3 – *Approche de résolution séquentielle*

8.4 Expérimentations et comparaison

8.4.1 Approche Séquentielle

Cette approche de résolution suit le même schéma que l'approche définie dans le chapitre 5, à la différence que les clusters dont le nombre de variables dépasse un certain seuil sont résolus en utilisant l'algorithme BSO parallèle. Cette approche permet un gain de temps considérable. Le tableau 8.1 ainsi que la figure 8.4 illustrent les résultats de l'approche DBSCAN bidimensionnelle avec rayon *euclidien*.

Rayon Euclidien						
Nom	Clauses	Rayon	SAT(%)	Time(s)	DPLL	BSO
Ibm1	54682	R=50	97.44	890.65	538	20
		R=100	95.65	117.81	392	29
		R=150	95.68	126.67	381	25
		R=200	95.58	133.15	362	31
		R=250	95.75	173.03	361	28
Ibm2	10561	R=50	98.90	6.2	83	1
		R=100	99.01	4.51	74	1
		R=150	99.02	4.36	71	1
		R=200	98.48	5.13	69	1
		R=250	98.92	21.63	60	1
Ibm7	37388	R=50	99.15	54.84	225	4
		R=100	98.50	95.09	170	8
		R=150	98.47	63.29	172	7
		R=200	98.40	80.2	170	8
		R=250	97.94	89.48	142	10
Ibm13	63123	R=50	98.42	2743.11	809	7
		R=100	96.77	266.84	609	29
		R=150	95.89	1382.65	621	39
		R=200	95.09	414.53	584	40
		R=250	95.02	2254.82	540	47
Aim200	1182	R=50	91.37	12.61	13	0
		R=100	92.05	3.19	9	1
		R=150	91.12	43.53	7	2
		R=200	90.61	6.76	4	2
		R=250	91.03	6.48	5	2
Fla500	2205	R=50	96.96	0.55	59	0
		R=100	97.32	103.96	42	0
		R=150	96.96	237.56	35	0
		R=200	94.47	23.96	32	2
		R=250	94.06	14.17	20	4

TABLE 8.1 – *Approche DBSCAN Bidimensionnelle à rayon Euclidien*

Les résultats du tableau 8.1 indique une corrélation entre l'augmentation du rayon des clusters avec l'accroissement du nombre de clusters résolus par l'algorithme BSO-parallèle.
La comparaison de ces résultats avec l'approche définie dans le chapitre 5 indique une stabilité dans les taux de satisfiabilité des instances. En effet, le processus de résolution reste le même hormis la génération aléatoire de la solution initiale.
Par ailleurs, un gain de temps considérable est à noter, et ce grâce au passage du calcul CPU au calcul CPU/GPU. La figure 8.4 illustre une comparaison entre les temps d'exécution des deux approches pour les trois instances IBM1, IBM2, IBM7.

Remarque : Dans l'approche DBSCAN utilisant le rayon *Hamming*, la condition dans la création de cluster est portée sur le nombre de clauses communes aux variables, ce qui entraine la génération de clusters avec un nombre minime de variables à instancier. Ces clusters pouvent être résolu

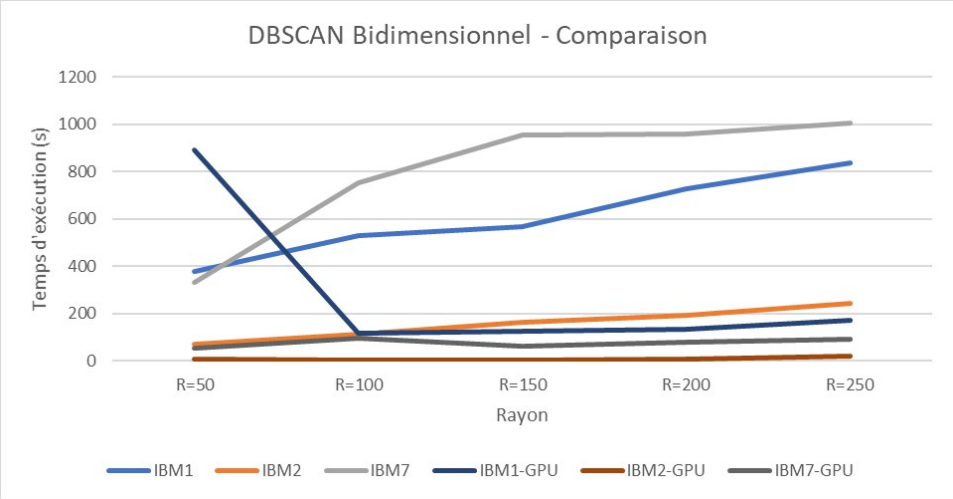


FIGURE 8.4 – *Comparaison entre l'approche séquentielle CPU et CPU/GPU - Temps d'exécution*

par DPLL sans avoir recours à l'algorithme BSO ou BSO-parallèle.

8.4.2 Approche Parallèle

DBSCAN Multidimensionnelle

Le tableau 8.2 illustre les temps de résolution ainsi que les taux de satisfaction des différentes instances après clustering par DBSCAN multi-dimensionnelle et résolution par l'algorithme BSO parallèle.

Approche DBSCAN Multidimensionnelle / résolution parallèle						
R=1	ibm1	ibm2	ibm7	ibm13	aim200	fla500
Temps de clustering (s)	6.28	0.28	3.08	10.33	0.034	0.029
# clusters dpll	902	301	799	1337	5	14
Temps de résolution (s)	711.64	113.004	744.07	1401.09	0.0009	2.46
#clusters bso	173	70	241	332	3	12
Temps de résolution (s)	7.43	9.2	12.05	25.62	1.45	2.13
Taux de satisfaction(%)	86	87.98	92.52	92.17	89.26	92.97
R=2						
Temps de clustering (s)	210.38	3.06	87.19	267.64	0.106	0.39
# clusters dpll	36249	5704	21259	35950	777	2027
Temps de résolution (s)	436.08	25.99	675.008	477.03	0.019	0.01
#clusters bso	154	13	64	129	0	0
Temps de résolution (s)	6.51	3.36	6.76	18.2	0	0
Taux de satisfaction(%)	95.79	95.17	94.69	95.81	87.31	88.44

TABLE 8.2 – *Approche de résolution BSO parallèle DBSCAN – Taux de satisfaction et temps de résolution*

Le tableau 8.2 montre une baisse de la qualité des solutions par rapport à l'approche multidimensionnelle vue en chapitre 5. Ce qui est dû au fait qu'à chaque résolution d'un cluster, la solution de ce dernier soit

propagée sur l'ensemble des instances, dans l'approche précédente. Alors que dans le cas présent, tous les clusters sont résolus de façon indépendante les uns des autres.
Cette différence de méthode de résolution induit également une augmentation du temps de résolution dû aux différents clusters résolus par l'algorithme DPLL. En effet, lors de la propagation d'une solution partielle avant la création d'un nouveau cluster, le nombre de variables à instancier diminue, réduisant le temps de résolution de ce dernier.

STING

Le tableau 8.3 exhibe les résultats des expérimentations de la résolution parallèle des clusters obtenus après clustering par l'algorithme STING vertical.

STING Vertical / résolution parallèle						
Taille =S	ibm1	ibm2	ibm7	ibm13	aim200	fla500
Temps de clustering (s)	0.14	0.033	0.051	0.16	0.0065	0.0086
# clusters dpll	1	3	9	1	0	0
Temps de résolution (s)	0.013	5.73	347.17	2.65	0	0
#clusters bso	387	110	340	528	8	20
Temps de résolution (s)	35.79	4.96	12.79	9.8	1.003	3.74
Taux de satisfaction(%)	96.49	94.73	94.02	95.54	88.32	93.6
Taille=M						
Temps de clustering (s)	0.096	0.037	0.121	0.13	0.026	0.01
# clusters dpll	0	1	1	1	0	0
Temps de résolution (s) 0	0.049	0.094	2.65	0	0	
#clusters bso	194	56	174	264	4	10
Temps de résolution (s)	48.22	10.2	20.77	16.77	2.05	6.84
Taux de satisfaction(%)	95.61	93.49	93.14	93.71	88.49	93.46
Taille=L						
Temps de clustering (s)	0.25	0.049	0.11	0.27	0.021	0.012
# clusters dpll	0	1	1	1	0	0
Temps de résolution (s)	0	0.02	0.038	3.32	0	0
#clusters bso	97	28	87	132	2	5
Temps de résolution (s)	50.69	12.56	32.41	24.77	3.92	10.36
Taux de satisfaction(%)	92.33	90.55	90.84	90.74	91.87	91.47

TABLE 8.3 – *Approche de résolution BSO parallèle STING – Taux de satisfaction et temps de résolution*

En comparant les résultats obtenus dans le tableau 8.3 avec ceux de l'approche initialement proposée du STING vertical (chapitre 6), on note une réduction considérable du temps de résolution. En effet, la résolution d'un cluster avec l'algorithme BSO-parallèle induit une réduction du temps de résolution de ce dernier. Dans le cas présent, la résolution de tous les clusters se fait de manière simultanée et en utilisant l'algorithme BSO parallèle induisant une réduction significative du temps de résolution qui est l'un des buts recherchés lors du passage au calcul GPU.

Néanmoins, une baisse de la qualité des solutions est notée et cela est dû à la propagation des solutions partielles sur le reste des clusters non encore résolus dans l'approche initiale.

8.5 Conclusion

L'architecture fortement parallèle des GPU ainsi que les différents outils de programmation GPU tel quel CUDA permettent un important calcul parallèle. Dans le cadre de ce projet, le calcul GPU pour la résolution de problème complexe permet d'obtenir des solutions plus efficaces et efficientes.
Une version parallèle de l'algorithme BSO a été proposée dans ce chapitre dans le but de permettre une résolution plus efficiente des instances SAT. Deux différentes approches ont été proposées pour la résolution des sous-instances SAT. Une première approche parallèle où les sous-instances dont le nombre de variables dépasse un certain seuil sont résolues simultanément permettant un gain considérable de temps de résolution. La seconde approche séquentielle où les clusters sont résolus au fur et à mesure de leur création permettant une meilleure résolution.

Les différentes expérimentations réalisées ont montré l'intérêt et l'impact de l'utilisation du calcul GPU dans la résolution de problème et particulièrement sur l'efficience des solutions. En effet, un gain de temps considérable a été noté sur toutes les expérimentations réalisées.
Cependant, toutes les approches exécutées sur GPU n'ont pas été aussi satisfaisantes que celles exécutées sur CPU en termes de taux de satisfiabilité. Cela est dû à l'approche de résolution-clustering proposée sur CPU permettant la propagation d'une solution de cluster sur le reste des clusters non encore résolus.

En résumé, nous pouvons conclure que l'approche de résolution basée sur l'exploitation des techniques de datamining permet d'avoir des solutions de bonne qualité tant sur le plan du taux de satisfiabilité que celui du temps de résolution. Le calcul GPU associé de façon judicieuse tel que dans le DBSCAN bidimensionnel permet un gain de temps de résolution considérable avec une qualité de résolution satisfaisante démontrant l'intérêt et l'impact de ces méthodes.

Conclusion générale

Tout au long de ce travail de thèse, nous avons exploré l'intelligence et la technologie offertes par le datamining et en particulier les techniques de clustering en tant que prétraitement pour les instances SAT.
Plusieurs approches de clustering ont été proposées au long de cette thèse. La structure principale de ces approches se résume en deux parties. La première étape consiste en l'étude exploratoire des instances SAT et particulièrement l'étude de la dispersion des variables sur les clauses. Cette étude a pour objectif de déterminer la technique de clustering la plus adéquate à chaque instance en fonction de sa distribution. Après détermination du meilleur algorithme de clustering pour une instance donnée, celui-ci est appliqué permettant la division de l'instance en plusieurs sous-instances de moindre complexité. Ces sous-instances sont résolus indépendamment les unes des autres dans la seconde phase où les algorithmes DPLL et BSO sont exécutés en fonction du nombre de variables à instancier par cluster.
L'étude de la distribution des variables sur les clauses a permis de distinguer deux principaux modèles. Dans la première distribution, les variables sont dispersées définissant des régions de haute densité clairsemées de régions de faible densité ou de régions vides. A l'opposé, le second modèle de distribution présente un espace où les variables sont dispersées aléatoirement sans créer aucune région de densité considérable. Une particularité relative à la fréquence des variables est observée dans certaines instances où la majorité des variables présente une fréquence élevée.
Nous avons associé à chacune de ces distributions une technique de clustering jugée plus appropriée. En effet, les algorithmes de clustering basés sur la densité semblent convenir parfaitement aux instances présentant la première distribution. Alors que pour la seconde distribution, le clustering basé sur les grilles est considéré comme plus approprié. Toutefois, pour les instances présentant une particularité dont l'ensemble de leurs variables a une fréquence considérable, les algorithmes d'extraction de modèles fréquents semblent être les plus appropriés.
Des adaptations de plusieurs algorithmes de clustering ont été proposées au long de ce travail.
Deux principales adaptations de l'algorithme DBSCAN ont été proposées pour répondre aux instances dont la distribution correspond au premier modèle défini. Dans l'approche bidimensionnelle, les variables représentent le principal paramètre du clustering. Un cluster est alors défini par une variable centrale x_i dont la région est définie par l'ensemble des variables avoisinantes. Trois différentes définitions de voisinage ont été suggérées. Les deux premières considèrent la distance Euclidienne et la distance de Hamming alors que la dernière approche propose une définition de la densité propre aux instances SAT.

Dans la seconde adaptation du DBSCAN, les clauses représentent le principal paramètre de clustering. L'algorithme commence par sélectionner une clause aléatoirement pour représenter le centre du cluster. Une région autour de cette clause est alors déterminée. Trois différentes approches sont proposées en fonction du rayon de distance proposé ainsi que de la résolution des différents clusters.
Une adaptation de l'algorithme STING a été proposée pour les instances de distribution aléatoire. Dans cette approche, la taille des cellules est préalablement fixée permettant un meilleur découpage de la grille et un meilleur clustering. Trois différentes tailles permettant de fixer le nombre de variables et de clauses par cluster ont été définies.

Un nouvel algorithme, Méta-Apriori, basé sur l'algorithme Apriori est proposé et présente une amélioration de ce dernier. L'algorithme comporte trois étapes dont la première consiste à subdiviser l'instances en deux voire plusieurs sous-ensembles aléatoirement ou en fonction de la fréquence des variables. La seconde phase consiste en l'exécution d'une version améliorée de l'algorithme Apriori. Enfin, une fusion des différents itemsets résultants est réalisée. Cet algorithme est exécuté sur des instances SAT permettant l'extraction des variables les plus fréquemment répétées ensemble. Ces variables sont alors considérées comme variables représentatives de clusters.
Enfin, une hybridation de l'algorithme Apriori avec l'algorithme k-means est proposée. Les ensembles de variables fréquemment répétées ensemble sont utilisés comme centroïds initiaux du k-means.
L'objectif de l'ensemble de ces contributions est de réduire la complexité des instances SAT en créant des sous-instances pouvant être résolus séparément et en un temps moindre.
Pour une plus grande efficience de ces approches, un parallélisme CPU/GPU est réalisé. Une modélisation parallèle de l'algorithme BSO est d'abord proposée permettant un gain de temps considérable quant à la puissance de calcul parallèle. Cet algorithme est alors exécuté lors de la résolution des différents clusters obtenus. Une première approche consiste à exécuter l'algorithme BSO-parallèle sur un cluster à la fois dans le but de propager la solution de ce dernier sur le reste des clusters non encore résolus, alors que la seconde approche permet la résolution simultanée de tous les clusters.
Dans le but de valider l'ensemble de ces contributions, des expérimentations ont été réalisées sur des instances SAT connues et reconnues. Les résultats de ces dernières attestent de l'efficacité de ces dernières particulièrement en termes d'efficience.
Un premier travail futur est d'exécuter l'algorithme BSO-parallèle sur l'ensemble des approches proposées dans le présent travail et de les comparer avec ceux déjà effectués. Il serait aussi intéressant de réaliser le clustering ainsi que la résolution sur CPU/GPU. Enfin, l'intégration de ces approches à un solver performant pourrait avoir un impact sur le temps de résolution de ce dernier.

Mes contributions scienfifiques

Articles :

— HIRECHE, Celia et DRIAS, Habiba. Multidimensional appropriate clustering and DBSCAN for SAT solving. Data Technologies and Applications, 2019, vol. 53, no 1, p. 85-107.

— HIRÈCHE, Célia, DRIAS, Habiba, et BENHAMOUDA, Neyla Cherifa. Frequent Patterns Mining for the Satisfiability Problem. Polibits, 2017, vol. 55, p. 59-63.

Conférences :

— HIRECHE, Celia et DRIAS, Habiba. Density based clustering for satisfiability solving. In : World Conference on Information Systems and Technologies. Springer, Cham, 2018. p. 899-908.

— BENHAMOUDA, Neyla Cherifa, DRIAS, Habiba, et HIRÈCHE, Célia. Meta-apriori : a new algorithm for frequent pattern detection. In : Asian Conference on Intelligent Information and Database Systems. Springer, Berlin, Heidelberg, 2016. p. 277-285.

— DRIAS, Habiba, HIRÈCHE, Célia, et DOUIB, Ameur. Datamining techniques and swarm intelligence for problem solving : application to SAT. In : 2013 World Congress on Nature and Biologically Inspired Computing. IEEE, 2013. p. 200-206.

— DRIAS, Habiba, DOUIB, Ameur, et HIRÈCHE, Célia. Swarm intelligence with clustering for solving SAT. In : International Conference on Intelligent Data Engineering and Automated Learning. Springer, Berlin, Heidelberg, 2013. p. 585-593.

Bibliographie

Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, et Prabhakar Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998.

Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. Dans *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

Manish Arora. The architecture and evolution of cpu-gpu systems for general purpose computing. *By University of California, San Diago*, 27, 2012.

David Arthur et Sergei Vassilvitskii. k-means++ : The advantages of careful seeding. Dans *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

Benchmark. https ://www.cs.ubc.ca/ hoos/satlib/benchmarks/sat/rnd3sat/uf20-91.tar.gz. a.

Benchmark. http ://www.satcompetition.org/edacc/sc14/. b.

Neyla Cherifa Benhamouda, Habiba Drias, et Célia Hirèche. Meta-apriori : a new algorithm for frequent pattern detection. Dans *Asian Conference on Intelligent Information and Database Systems*, pages 277–285. Springer, 2016.

James C Bezdek, James Keller, Raghu Krisnapuram, et Nikhil Pal. *Fuzzy models and algorithms for pattern recognition and image processing*, volume 4. Springer Science & Business Media, 1999.

Armin Biere, Alessandro Cimatti, Edmund Clarke, et Yunshan Zhu. Symbolic model checking without bdds. Dans *International conference on tools and algorithms for the construction and analysis of systems*, pages 193–207. Springer, 1999.

Armin Biere, Marijn Heule, et Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

Shaowei Cai, Chuan Luo, et Kaile Su. Ccanr : A configuration checking based local search solver for non-random satisfiability. Dans *International Conference on Theory and Applications of Satisfiability Testing*, pages 1–8. Springer, 2015.

Seung-Seok Choi, Sung-Hyuk Cha, et Charles C Tappert. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1) :43–48, 2010.

Stephen A Cook. The complexity of theorem-proving procedures. Dans *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

Nvidia Corporation. Nvidia cuda c programming guide version 4.0, 2012.

Ian Davidson et SS Ravi. Agglomerative hierarchical clustering with constraints : Theoretical and empirical results. Dans *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 59–70. Springer, 2005.

Lawrence Davis. Handbook of genetic algorithms. 1991.

Martin Davis, George Logemann, et Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) : 394–397, 1962.

Martin Davis et Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3) :201–215, 1960.

Marco Dorigo et Mauro Birattari. *Ant colony optimization*. Springer, 2010.

Habiba Drias, Ameur Douib, et Célia Hirèche. Swarm intelligence with clustering for solving sat. Dans *International Conference on Intelligent Data Engineering and Automated Learning*, pages 585–593. Springer, 2013.

Habiba Drias, Souhila Sadeg, et Safa Yahi. Cooperative bees swarm for solving the maximum weighted satisfiability problem. Dans *International Work-Conference on Artificial Neural Networks*, pages 318–325. Springer, 2005.

SAT encoded BMC. http ://www.satcompetition.org/2013/downloads.shtml.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. Dans *Kdd*, volume 96, pages 226–231, 1996.

Randy Fernando. Gpgpu : general general-purpose purpose computation on gpus. *NVIDIA Developer Technology Group*, 2004.

J. W. Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, University of Pennsylvania, 1995.

Marhajan Y. Malik S. (2004). Zchaff sat solver. In Group Princeton University. Fu, Z. http ://www. princeton. edu/ chaff/zchaff. html.

Michael R Garey et David S Johnson. Computers and intractability : A guide to the theory of npcompleteness (series of books in the mathematical sciences), ed. *Computers and Intractability*, 340, 1979.

Fred W Glover et Gary A Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

Youssef Hamadi, Said Jabbour, et Lakhdar Sais. Manysat : a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6 :245–262, 2008.

Jiawei Han, Jian Pei, et Micheline Kamber. *Data mining : concepts and techniques*. Elsevier, 2011.

Ramin Hedayatzadeh, Foad Akhavan Salmassi, Manijeh Keshtgari, Reza Akbari, et Koorush Ziarati. Termite colony optimization : A novel approach for optimizing continuous problems. Dans *2010 18th Iranian Conference on Electrical Engineering*, pages 553–558. IEEE, 2010.

Celia Hireche et Habiba Drias. Density based clustering for satisfiability solving. Dans *World Conference on Information Systems and Technologies*, pages 899–908. Springer, 2018.

Celia Hireche et Habiba Drias. Multidimensional appropriate clustering and dbscan for sat solving. *Data Technologies and Applications*, 53(1) :85–107, 2019.

Célia Hirèche, Habiba Drias, et Neyla Cherifa Benhamouda. Frequent patterns mining for the satisfiability problem. *Polibits*, 55 :59–63, 2017.

Holger H Hoos et al. An adaptive noise mechanism for walksat. Dans *AAAI/IAAI*, pages 655–660, 2002.

Holger H Hoos et Thomas Stützle. *Stochastic local search : Foundations and applications*. Elsevier, 2004.

Robert G Jeroslow et Jinchang Wang. Solving propositional satisfiability problems. *Annals of mathematics and Artificial Intelligence*, 1 (1-4) :167–187, 1990.

Dervis Karaboga et Beyza Gorkemli. Solving traveling salesman problem by using combinatorial artificial bee colony algorithms. *International Journal on Artificial Intelligence Tools*, 28(01) :1950004, 2019.

George Karypis, Eui-Hong Sam Han, et Vipin Kumar. Chameleon : Hierarchical clustering using dynamic modeling. *Computer*, (8) : 68–75, 1999.

Leonard Kaufman et Peter J Rousseeuw. *Finding groups in data : an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

Ashiqur R KhudaBukhsh, Lin Xu, Holger H Hoos, et Kevin Leyton-Brown. Satenstein : Automatically building local search sat solvers from components. *Artificial Intelligence*, 232 :20–42, 2016.

David B Kirk et W Hwu Wen-Mei. *Programming massively parallel processors : a hands-on approach*. Morgan kaufmann, 2016.

Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2) :129–137, 1982.

Joao P Marques-Silva et Karem A Sakallah. Grasp : A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5) :506–521, 1999.

Madigan C. F. Zhao Y. Zhang L. Malik S. Moskewicz, M. W. Chaff : Engineering an efficient sat solver. Dans *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

Raymond T Ng et Jiawei Han. E cient and e ective clustering methods for spatial data mining. Dans *Proceedings of VLDB*, pages 144–155, 1994.

NVIDIA. https ://www.nvidia.com/fr-fr/.

A. G. Random. https ://baldur.iti.kit.edu/sat-competition-2016/index.php ?cat=benchmarks.

Frequent Itemset Mining Dataset Repository. http ://fimi.ua.ac.be/data/.

R. SAT. https ://baldur.iti.kit.edu/sat-competition-2016/index.php ?cat=benchmark.

Bart Selman, Henry A Kautz, Bram Cohen, et al. Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability*, 26 :521–532, 1993.

Bart Selman, Hector J Levesque, David G Mitchell, et al. A new method for solving hard satisfiability problems. Dans *AAAI*, volume 92, pages 440–446. Citeseer, 1992.

JM Silva et K Sakallah. Grasp-a new search algorithm for satisfiability. iccad, 1996.

Tufféry Stéphane. *Data mining et statistique décisionnelle : l'intelligence des données*. Editions Technip, 2012.

Architecture Turing. https ://www.nvidia.com/fr-fr/geforce/turing/.

Karl Von Frisch et Leigh E Chadwick. *The dance language and orientation of bees*, volume 1. Belknap Press of Harvard University Press Cambridge, MA, 1967.

Wei Wang, Jiong Yang, Richard Muntz, et al. Sting : A statistical information grid approach to spatial data mining. Dans *VLDB*, volume 97, pages 186–195, 1997.

Tian Zhang, Raghu Ramakrishnan, et Miron Livny. Birch : an efficient data clustering method for very large databases. Dans *ACM Sigmod Record*, volume 25, pages 103–114. ACM, 1996.