N° d'ordre :06/2014-D/INF

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université des Sciences et des Technologies Houari Boumediéne

Faculté d'Electronique et d'informatique

THESE

Présentée pour l'obtention du grade de DOCTEUR EN SCIENCES

EN : Informatique

Spécialité : Intelligence Artificielle

Par : Youcef DJENOURI

Sujet

Fouille Parallèles des Règles d'association Sur GPU

Soutenue publiquement, le 03/12 /2014, devant le jury composé de :

M. Slimane Larabi	Professeur	USTHB	Président
Mme. Zineb Habbas	Professeur	Lorraine	Directrice de Thèse
M. Karim Atif	MCA	USTHB	Examinateur 1
Mlle. Dalila Boughaci	MCA	USTHB	Examinatrice 2
M. Riadh Baba Ali	MCA	USTHB	Examinateur 3
M. Ahmed- Ouamer Rachid	Professeur	UMMTO	Examinateur 4

Dédicace

A ma trés chère mére qui ne cesse de se sacrifier pour nous.

A mon superbe pére qui fait tout pour nous rendre heureux.

A mes frére Djamel et Rachid.

A mes belles soeurs Fella et Farida

A mes adorables anges Zineb, Abdenour, Abderhmane, Nour el houda et Abdellah.

A mon binôme de master Amine.

A mes trés chers amis Hamza, Mohamed, Nourredine et Ryad qui ont partagé avec moi, les moments les plus dûrs et les plus beaux de ce travail.

En particulier, à ma fianciée Asma pour son soutien tout au long de la réalisation de ce travail.

Merci à tous.

Remerciements

Je remercie le bon DIEU le tout puissant de m'avoir donné le courage et la patience durant l'élaboration de ce modeste travail. Je tiens à exprimer mes vifs remerciements à ma directrice Pr. Zineb Habbas pour son soutien tout au long de cette thèse, ses conseils, ses recommandations, sa disponibilité et ses encouragements.

J'adresse ma profonde gratitude à l'ensemble du corps enseignant de l'USTHB ayant contribué à ma formation.

Je remercie les membres de jury d'avoir accepté d'évaluer et juger mon présent travail.

Mes remerciements vont aussi à mes amis et collègues pour leurs encouragements, et à toute personne qui a contribué de prés ou de loin à la préparation de ce travail.

Résumé

La fouille des règles d'association est un processus qui permet de déduire un ensemble de règles reliant les différentes données d'une base de transactions. Avec l'apparition du web, les algorithmes d'extraction des règles d'association sont devenus de plus en plus lents. L'utilisateur ne peux exploiter toutes les règles obtenues par le processus d'extraction des règles d'association, ce qui nécessite une autre fouille des règles d'association afin d'extraire d'autres informations sur les règles permettant d'organiser, résumer, et réduire l'espace des règles d'association. Dans un premier temps, nous allons traiter le problème d'extraction des règles d'association en proposant plusieurs variantes séquentielles et même parallèles (sur GPU) de l'algorithme des essaims d'abeilles pour les règles d'association. Dans un deuxième temps, nous proposons des algorithmes de fouille des règles d'association ; ces algorithmes permettent de réduire l'espace des règles d'association. Pour valider les approches proposées, nous les avons appliqués sur des problèmes réels, tel que le problème de satisfiabilité des clauses (SAT) et la recherche d'informations, les expérimentations montrent l'efficacité de nos algorithmes par rapport aux algorithmes existants.

Mots Clés : Extraction des règles d'association, fouille de données, intelligence par essaims, Technologie GPU.

Table des matières

Table des figures

Introduction générale

Les règles d'association sont trés utilisées dans plusieurs domaines comme la recherche d'informations, la bio-informatique, la médécine, et la gestion des stocks. L'extraction des règles d'association est un processus qui permet de déduire un ensemble de règles reliant les différentes données d'une base de transactions. Plusieurs algorithmes ont été dévellopés pour la résolution du problème d'extraction des règles d'association. Apriori est l'algorithme le plus utilisé, la génération des règles se fait récursivement en commençant par les règles de taille 2, les règles de taille 3 sont générées à partir des règles de taille 2 et ainsi de suite jusqu'au bout d'une itération on ne génère aucune règle. Avec l'apparition du web, les algorithmes d'extraction des règles d'association sont devenus de plus en plus lents.

Le processus d'extraction des règles d'association généralement produit un nombre important des règles, cette bénéfique information parfois devient un obstacle pour l'utilisateur. En effet, il ne peut pas exploiter toutes les règles délivrées, ce qui nécessite une autre fouille des règles d'association afin d'obtenir d'autres informations sur les règles permettant d'organiser, résumer et réduire l'espace des règles d'association.

L'espace des règles d'association est beaucoup plus grand que l'espace des données. Pour celà, les méthodes de fouille classiques proposées dans la litérature sont devenues inutiles. En contre partie, l'intelligence par essaim est une classe des métaheuristiques basées sur population qui a montré une efficacité de résolution de plusieurs problèmes complexes. De plus, avec la téchnologie GPU, ces méthodes sont bien parallélisables tout en diminuant parfaitement le temps de calcul et en gardant en même temps la qualité des solutions.

Notre problématique se résume en deux axes, dans le premier axe, on traite

le problème d'extraction des règles d'association en proposant plusieurs variantes séquentielles et même parallèles (sur GPU) de l'algorithme des essaims d'abeilles pour les règles d'association. Dans le deuxième axe, on projéte notre étude sur les règles obtenues par le processus d'extraction en proposant des algorithmes de fouille des règles d'association. Ces algorithmes permettent dans un premier temps d'organier et résumer l'espace des règles et dans un deuxième temps de réduire cet espace des règles d'association.

Pour valider les approches proposées, nous les avons appliqué sur des problèmes réels. Concernant les algorithmes d'extraction des règles d'association nous les avons testé sur le problème de satisfiabilité (SAT) alors que pour les algorithmes de fouille et réduction des règles d'association, nous les avons testés sur le problème de recherche d'information. Les expérimentations montrent l'efficacité de nos algorithmes par rapport aux algorithmes existants.

Le plan de la thése est comme suit : Dans le premier chapitre nous présenterns des algorithmes d'extraction des règles d'association, dans le deuxième chapitre nous détaillerons les algorithmes de fouille et de réduction des règles. Ensuite, nous présenterons quelques algorithmes d'intélligence par essaims dans le troisiéme chapitre. Nous introduisons notre première contribution ainsi que les résultats obtenus et tous celà dans le quatriéme chapitre. Dans le chapitre suivant, nous détaillerons la deuxième contribution de la thése en devellopant deux algorithmes de fouille des règles et en testant les algorithmes proposés sur des données classiques. Le sixiéme chapitre explique l'application de notre algorithme d'extraction des règles au problème de satisfiabilité (SAT). Le septiéme chapitre expliquera l'application de l'algorithme de réduction des règles au problème de recherche d'informations. A la fin de cette thése, nous dresserons un bilan général de ce présent travail ainsi que ses futures perspectives.

Chapitre 1

Les algorithmes d'extraction des règles d'associations

1.1 Introduction

La fouille de données est un ensemble de méthodes et de téchniques permettant l'extraction de connaissances à partir d'une grande masse de données. Il existe plusieurs tâches de fouille de données telles que : la classification supervisée et la classification non supervisée, l'extraction des règles d'associations, la régression, la prédiction...ect. Nous nous intéressons à l'extraction des règles d'association et plus particulièrement aux règles booléennes.

L'extraction des règles d'association est le processus de découverte d'un ensemble de règles pertinentes à partir d'une base de transactions. Chaque transaction est un ensemble d'items alors qu'un item est la donnée la plus élémentaire du problème rencontré. A titre d'exemple : un item peut être considéré comme un produit pour le problème Market Basket Analysis [1], comme un terme pour le problème de recherche d'information [3], et comme un génome pour le problème de classification de génotype [2].

Le processus d'extraction des règles consiste à trouver les règles les plus représentatives des données incluses dans la base de transactions.

Les règles d'associations sont un modèle de représentation de connaissances et contiennent deux parties disjointes, chaque partie contient un ensemble d'items

(itemset).

Parmi les différents types de règles, on a les règles sures, une règle sure est une règle qui permet de représenter les implications entre les itemsets, c'est à dire si l'ensemble des items de la partie antécédente de la règle est présent dans une transaction t, les items de la partie conséquente de la règle seront sûrement présents dans t. Les règles sures sont des règles trés utilisées dans le processus d'extraction car elles représentent l'ensemble de transactions sans incertitude.

Il existe une panoplie des algorithmes d'extractions des règles d'association. Il y'a des algorithmes exacts qui donnent l'ensemble de toutes les règles pertinentes. Ces algorithmes consomment énormément de temps et de l'espace mémoire. D'autres algorithmes sont dits approchés car ils donnent juste une partie de règles pertinentes mais avec un temps raisonnable. De plus, des modèles parallèles ont été proposés afin de réduire le temps d'exécution des algorithmes d'extraction des règles d'association.

Ce chapitre présente une synthèse sur les algorithmes d'extraction des règles d'association, ainsi que les différentes classifications qui existent dans l'état de l'art des algorithmes d'extraction des règles d'association. A la fin de ce chapitre, on proposera une nouvelle classification de ces algorithmes.

1.2 Concepts de Base

Le problème d'extraction des règles d'association peut être défini comme suit : Considérons un ensemble de transactions $T = \{t_1, t_2, \ldots, t_m\}$ et un ensemble d'items $I = \{i_1, i_2, \ldots, i_n\}$, une règle d'association $(X \rightarrow Y)$ est composée de deux parties : X c'est la partie antécédente de la règle alors que Y est la partie conséquente. X et Y sont deux itemsets disjoints c'est à dire $X \cap Y = \emptyset$ un itemset est un ensemble d'items inclus dans I, un k-itemset est un itemset de taille k c'est à dire il contient k items.

Le problème d'extraction des règles d'association consiste à extraire un ensemble de règles qui couvrent le maximum de transactions et qui représentent parfaitement l'ensemble de transactions. Cependant, puisque la base de transactions est trés large, l'utilisateur ne cherche pas à trouver toutes les règles mais uniquement une partie de règles pertinentes. Pour celà, deux mesures sont uti-

lisées afin d'évaluer la qualité d'une règle, le support et la confiance de la règle. Le support d'un itemsets ($I' \subseteq I$) est le rapport entre le nombre de transactions contenant I' et le nombre de transactions total. Le support de la règle $X \rightarrow Y$ est le support de $X \cup Y$, la confiance de cette règle est [4] :

$$\frac{support(X \cup Y)}{support(X)}$$

.

En se basant sur ces deux mesures (le support et la confiance), on peut dire que le problème d'extraction consiste à trouver toutes les règles qui ont un support et une confiance supérieurs au minimum de support (Minsup) et au minimum de confiance (Minconf) réspectivement, telque Minsup et Minconf sont deux paramètres choisis par l'utilisateur [4].

Cependant, le support et la confiance n'évaluent pas parfaitement la qualité des règles obtenues par le processus d'extraction des règles. En effet, il y'a des règles évidentes avec un support et une confiance maximals et des règles rares avec un support et une confiance minimals [5].

Pour cette raison, d'autres mesures d'évaluations ont été proposées comme suit [6] :
$Lift(X \rightarrow Y) = \frac{confidence(X \rightarrow Y)}{support(Y)}$.
$Leverage(X \rightarrow Y) = support(X \rightarrow Y) - (support(X) \times support(Y))$.
$Coverage(X \rightarrow Y) = support(X)$.
$Conviction(X \rightarrow Y) = \frac{1-support(Y)}{1-confiance(X \rightarrow Y)}$.
$Information - Gain(X \rightarrow Y) = \log \frac{m \times support(X \rightarrow Y)}{support(X) \times support(Y)}$.

Exemple1.1 Considèrons la base de transactions (voir la Table1.1) qui contient 5 transactions {t_1, t_2, t_3, t_4, t_5} et 5 items {A, B, C, D, E}. Par exemple, pour calculer le support de ce 2-itemset (A,B), on doit compter d'abord le nombre de transactions contenant en même temps les items A et B qui est égal à 2. Le support de (A,B) est donc égal à 2/5. Maintenant, la confiance de la règle ($A \rightarrow B$) est égale au rapport de support($A \rightarrow B$) et de support(A), soit $\frac{2/5}{3/5}$=2/3. Si on considère (Minsup $<$ 2/5 et Minconf $<$ 2/3) alors la règle ($A \rightarrow B$) est acceptée sinon elle est refusée.

t_1	A	B	C
t_2	A	B	
t_3	C	D	
t_4	E	D	
t_5	C	A	

TABLE 1.1 – Exemple de représentation horizontale

Items	t_1	t_2	t_3	t_4	t_5
A	1	1	0	0	1
B	1	1	0	0	0
C	1	0	1	0	1
D	0	0	1	1	0
E	0	0	0	1	0

TABLE 1.2 – Exemple de la représentation verticale

De plus, les autres mesures de la règle $(A \rightarrow B)$ sont caclulées comme suit :
Lift$(A \rightarrow B)$=5/3.
Leverage$(A \rightarrow B)$=4/25.
Coverage$(A \rightarrow B)$=3/5.
Conviction$(A \rightarrow B)$=9/5.
Information-Gain$(A \rightarrow B)$=log 3/5.

Dans la littérature, il y'a trois types de représentations de la base de transactions et qui sont [7] [8] :

– La représentation horizontale : Elle est la plus utilisée dans les algorithmes

Transactions	A	B	C	D	E
t_1	1	1	1	0	0
t_2	1	1	0	0	0
t_3	0	1	0	1	0
t_4	0	0	0	1	1
t_5	1	0	1	0	0

TABLE 1.3 – Exemple de la représentation sous forme de bitmap

d'extraction des règles d'associations, chaque transaction est representée par un ensemble d'items qui sont contenus dans celle ci(voir la Table 1.1).

- La représentation verticale : Dans cette représentation, la base de transactions est represéntée par un ensemble d'items, chaque item est défini par l'ensemble de transactions qu'elles le contiennent. Cette structure est souvent appelée par *tidlists*. Prennons l'exemple de la Table 1.1, sa représentation verticale est illustrée dans la Table 1.2. Par exemple, l'item A appartient à t_1, t_2 et t_5 donc son tidlist est : {1,1,0,0,1}.
- La représentation sous forme de bitmap : Elle est trés utilisée dans le contexte parallèle, c'est une matrice de bits appelée souvent *Bitmap*. Chaque ligne représente une transaction et dans chaque transaction on définit les items qu'elle contient tel que : Bitmap[i][j]=1 si l'item j est inclus dans la transaction i, 0, sinon. La Table 1.3 illustre la représentation sous forme de bitmap de la Table 1.1

1.3 Les algorithmes d'extraction des règles d'association séquentiels

La méthode de résolution naîve pour le problème d'extraction des règles d'association est d'explorer toute la base de transactions ce qui génère 2^n itemset. En effet, le problème d'extraction des règles d'association est de nature NP-Complet. Par conséquent, de nombreux algorithmes exacts ont été proposés dans le cadre de minimiser le temps d'exploitation des itemsets comme Apriori [9], FPgrowth [10], DIC [11] et DHP [12]. Cependant, ces algorithmes sont fastidieux lorsqu'il s'agit de traiter de larges bases de transactions. Pour celà, d'autres algorithmes basés sur les métaheuristiques ont été développés. Dans ce qui suit, on détaillera sur les deux catégories d'algorithmes.

1.3.1 Les algorithmes d'extraction des règles d'association exacts

Apriori est l'algorithme le plus utilisé pour l'extraction des règles d'associations. Il est classé parmi les dix meilleurs algorithmes de fouille des données [13]. Il est principalement composé des deux étapes suivantes :

Génération des itemsets fréquents : Les itemsets fréquents sont déterminés d'une manière récursive de telle sorte que les itemsets de taille "k" sont générés à partir des itemsets de taille (k-1). Ce processus permet la réduction proportionelle de l'espace de recherche des itemsets fréquents. Cependant, on perd beaucoup de temps en terme de calcul des itemsets fréquents. En effet, à chaque fois qu'un itemset est généré, son support doit être calculé. S'il est supérieur à un support minimum (Minsup) alors, l'itemsets sera inséré dans la liste des itemsets fréquents. Ce processus se répètera jusqua où il ni aura plus d'itemsets fréquents à produire.

La génération des règles d'association La génération des règles se fait à partir des itemsets fréquents, déjà trouvés dans la première étape. Pour chaque itemeset fréquent, on génère toutes les règles correspondantes. Par la suite, pour chaque règle, on calcule sa confiance. Si sa confiance est supérieure à MinConf alors la règle est acceptée, sinon elle est rejetée.

Par conséquent, de nombreux algorithmes basés sur Apriori ont été proposés, Park et d'autres [12] développent the Direct Hashing Pruning(DHP) qui est une extension de l'algorithme Apriori utilisant une mémoire supplémentaire afin de calculer à l'avance les 2-itemsets fréquents en cours de la première itération. Aussi, le DHP réduit, progressivement, la taille de la base de transactions en éliminant à chaque fois les itemsets non fréquents.

L'algorithme DIC(Dynamic Itemsets Counting) proposé par S.Brin et d'autres [11], est la généralisation d'Apriori où la base de transactions est partitionnée en plusieurs parties de même taille, telles que chacune d'elles est allouée en mémoire. Initialement, les supports d'items sont calculés pour la première partition. Les items trouvés localement, sont utilisés pour générer les candidats 2-itemsets. Ensuite, la deuxième partition est lue afin de trouver le support de tous les candidats actuels, ce processus est répété pour les autres partitions. Après le traitement de

la dernière partition, le même processus se répétera jusqua où aucun candidat ne peut être généré.

Ashok Savasere et d'autres [14] proposent l'algorithme de partition à deux passes, qui divise, logiquement, la base de transactions en partitions totalement disjointes. Dans la première passe, chaque partition est lue afin de construire les tidlists des items. Ensuite, on génère localement tous les itemsets fréquents en appliquant des intersections entre les tidlists. Dans la deuxième passe, les itemsets fréquents de toutes les partitions se fusionnent afin de former un ensemble global d'itemsets fréquents.

L'algorithme (SEAR) d'Andreas Mueller [15] est identique à Apriori, sauf que SEAR stocke les candidats dans une structure appelée **prefix tree** au lieu de la structure **hash tree**. Dans **prefix tree** (également appelée Trie), chaque arrête est masquée par des items : les préfixes communs sont représentés par des branches d'arbres, et les suffixes uniques sont stockés dans les feuilles. Les tests de Mueller ont montré l'efficacité de la structure **prefix tree** par rapport à la structure **hash tree** lors de calcul des supports des itemsets candidats.

Zaki et d'autres [16] proposent l'algorithm ECLAT qui utlise la représentation verticale des items. Les k-itemsets fréquent sont organisés dans des classes d'équivalence disjointes par les (k-1) préfixes communs, de telle sorte que les (k+1)itemsets candidats peuvent être générés en rejoignant les paires de k-itemsets fréquents dans les mêmes classes. Le support d'un itemset candidat peut, alors, être calculé simplement en s'intéressant juste aux tidlists de sa classe.

Tous les algorithmes expliqués précédemment, nécessitent plusieurs accées à la base de transactions et génère un nombre important des itemsets candidats. Afin de contourner ces inconvénients, deux algorithmes (FP-Growth et FI-Growth) ont été développés qui compressent la base de transactions en mémoire centrale tout en permettant d'éviter la génération des itemsets candidats. La différence principale de ces deux algorithmes est la structure utilisée lors de la compression de la base de transactions.

Jian Han et d'autres [10] ont proposé l'algorithme FP-growth, qui utitlise une structure appelée **FP-tree** pour la compression de la base de transactions. L'algorithme est divisé en deux phases.

(1) La construction de la structure **FP-tree** : FP-tree est un arbre dont chaque noeud contient un item avec sa fréquence. Initialement, l'arbre contient seulement la tête avec la valeur null. Pour chaque transaction t, on doit parcourir l'arbre afin d'insérer les items de t dans le bon endroit de l'arbre. En effet, si l'item courant existe dans le chemin actuel alors on incrémente sa fréquence par 1. Sinon, on crée un nouveau noeud pour cet item dans le chemin actuel avec une fréquence de valeur 1.

(2)Extraction des itemsets fréquents : On produit les itemsets fréquents directement à partir de la structure **FP-tree**, en utilisant la stratégie **profondeur d'abord**.

Similaire au FP-growth, FI-growth qui est développé par Amphawan et d'autres [17], représente l'ensemble de données dans une structure appelée **FI-tree**. Il fournit aussi une opération **combinaison** qui permet de fusionner deux sous-arbres de même tête de telle sorte qu'on aditionne les fréquences de la tête et on fusionne les branches des sous-arbres. FI-growth se compose de deux phases : la construction de la structure **FI-tree** et l'extraction des itemsets fréquent à partir de **FI-tree**. La construction de FI-tree est réalisée comme dans FP-growth. Tandis qu'il existe principalement trois phases dans l'extraction des itemsets fréquents qui sont : La ramification, La recherche des branches restantes et La suppression.

Apriori et FP-growth sont les deux principaux algorithmes exacts d'extraction des règles d'associations. Les Figures 1.1 et 1.2 expliquent réspectivement en détail le déroulement de ces deux algorithmes sur l'exemple précédent présenté dans la Table 1.1 avec un $Minsup = 2/5$.

Malgré que cette catégorie d'algorithmes réduit propotionellement la complexité de problème d'extraction des règles d'association, a titre d'exemple, la complexité d'Apriori en pire des cas est $O(n^2 \times m)$, en augmentant le nombre d'items et de transactions, ces algorithmes deviennent lourds et inutiles. En effet, si on prend l'exemple d'Apriori, on aura besoin d'environ 30 heures afin de fouiller une base de transactions de 10000 d'items et de 1 million de transactions [7].

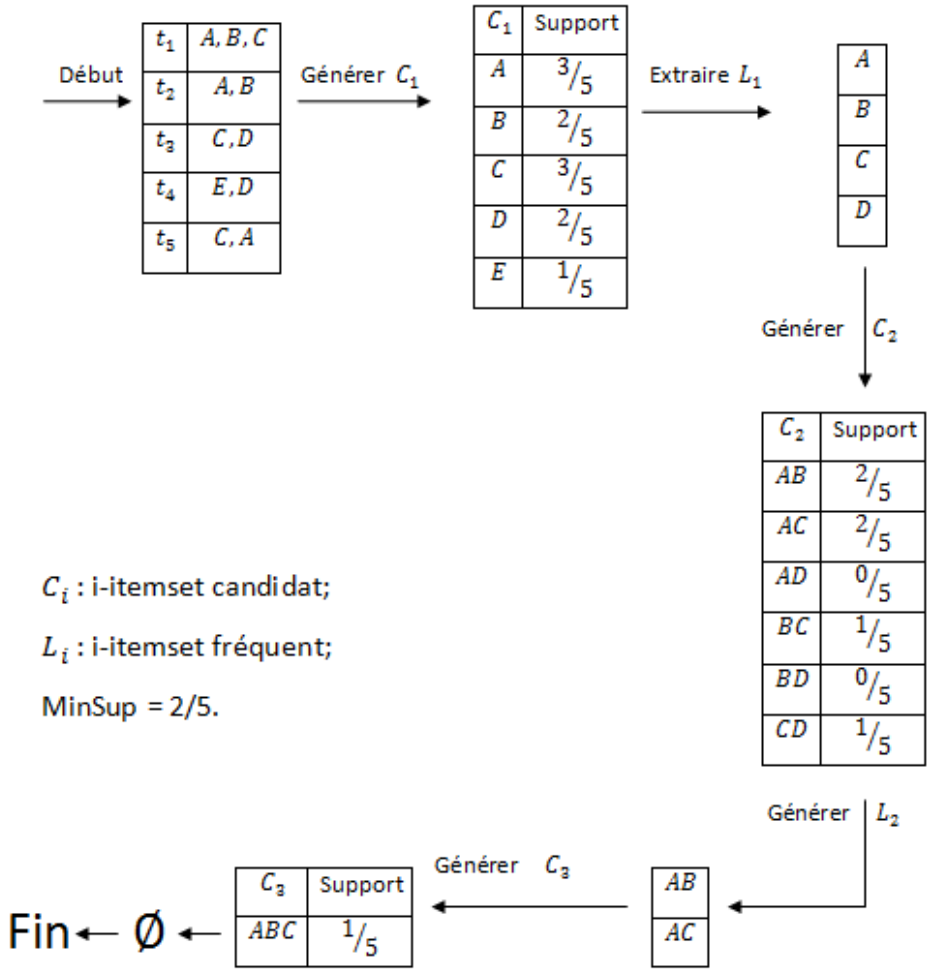


FIGURE 1.1 – *L'illustration de l'algorithme Apriori*

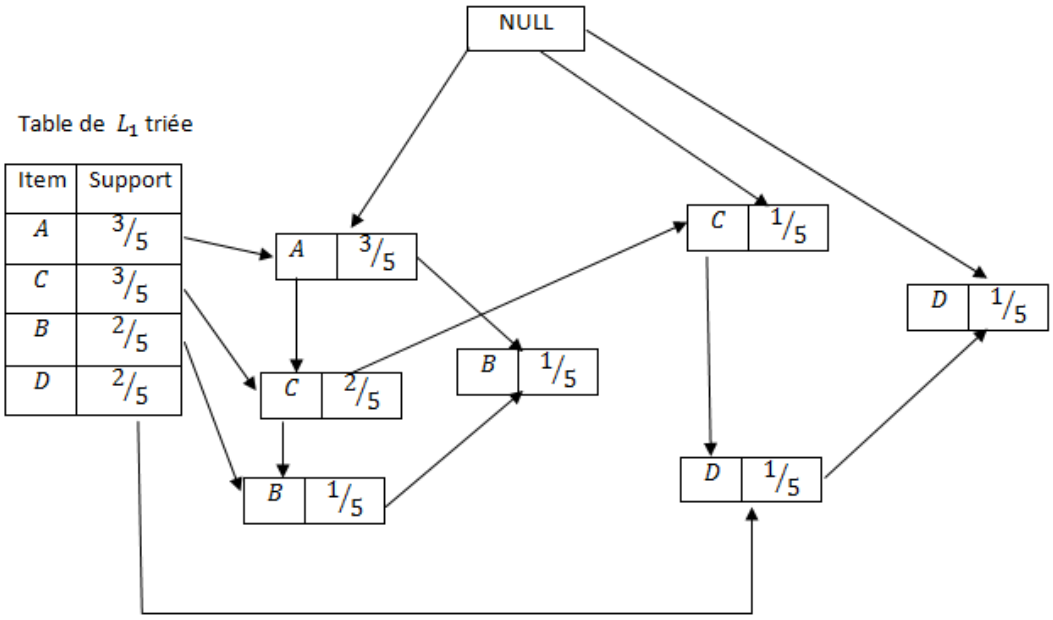


FIGURE 1.2 – *L'illustration de l'algorithme FPGrowth*

1.3.2 Les algorithmes d'extraction des règles d'association approchés

Dans la littérature, différentes métaheuristiques basées population ont été proposées pour le problème d'extraction des règles d'association.

De nombreux de ces travaux sont basées sur les algorithmes génétiques. Les deux premièrs algorithmes génétiques, pour l'extraction de règles d'association, appelés Genar et GAR sont proposés réspectivement dans [18] et [19] par Mata et d'autres. Ils utilisent une mauvaise représentation de l'individu ce qui provoque un calcul important lors de l'étape de l'évaluation.

Par la suite, plusieurs algorithmes génétiques utilisant une bonne représentation de l'individu sont proposés. Yan et d'autres [20] ont développé un algorithme appelé ARMGA, chaque règle est representée par une solution. Aussi, il utilise un séparateur afin de distinguer entre la partie antécédente et la partie conséquente de la règle. Lors de la production d'une nouvelle population, ARMGA utilise un simple croisement et un simple mutation.

Dans [21], un algorithme AGA est développé pour l'extraction des règles d'as-

sociations. Les deux principales différences entre ARMGA et AGA sont dans les stratégies de croisement et de mutation. En effet, le taux de croisement et de mutation est calculé en fonction de la moyenne de l'évaluation de la population courante. Cette stratégie permet la convergence rapide de la population à l'optimum global.

PQGMA est proposé par Liu pour les règles d'association dans [22]. L'extraction des règles est principalement réalisée par un algorithme génétique, tandis que la mutation et le croisement sont faits par le recuit simulé. Cependant, cet algorithme favorise la diversification à l'intensification ce qui conduit à ne pas exploiter des grandes parties de l'espace de solutions. Dans [23], un autre GA est proposé pour l'extraction des règles d'association. En utilisant un taux de mutation adaptatif, cet algorithme fournit une variation de population importante. Toutefois, la probabilité de mutation pour tous les chromosomes est calculée à chaque itération, ce qui augmente le temps d'exécution.

Un travail intéressant permettant une analyse détaillée sur la performance des algorithmes génétiques pour le problème d'extraction de règles d'association est proposé dans [24].

Romero et d'autres [25] ont développé l'algorithme G3PARM, basé sur la programmation génétique, ils utilisent le G3P (Grammar Guided Genetic Programming) afin d'éviter la génération des individus inadmissibles.

Kuo et d'autres [26] proposent l'algorithme PSOARM basé sur les essaims de particules. L'espace de voisinage est déterminé en déplaçant deux positions **avant** et **arrière** de chaque particule. Cet algorithme donne des meilleurs résulats par rapport à AGA mais la recherche basée sur les positions **avant** et **arrière** génère un grand nombre de voisinages qui favorise l'intensification à la diversification. Par conséquent, cet algorithme tombe toujours sur l'optimum local que l'optimum global.

Un travail intéressant qui utilise les colonies de fourmies pour le problème d'extraction des règles d'association est proposé dans [27] en développant un algorithme appelé ACO_R. Il applique le même processus que l'algorithme classique de ACO sauf qu'il utilise une fonction de gausse lors de la représentation de la solution afin de traiter des variables continues. ACO_R consomme un temps important lors de l'évaluation juste d'une seule solution.

Inspiré par la nouvelle métaheuristique développée en 2009 qui est basée sur le principe physique de gravité et de la loi des mouvements, Fariba et d'autres [28] proposent l'algorihme ARMBGSA. Chaque règle est modélisée par une masse, toutes les masses s'attirent en utilisant la loi des mouvements. Ensuite, à chaque itération k les masses les plus lourdes sont séléctionnées afin de produire de nouvelles masses en leur appliquant une force. L'algorithme ARMBGSA génère peu de règles par rapport aux autres algorithmes évolutionnaires.

1.4 Les algorithmes parallèles d'extraction des règles d'association

De nos jours, la taille des données augmente d'une façon massive au rang du téraoctets. Les algorithmes d'extraction des règles d'association séquentiels même approchés ne suffisent pas pour fouiller cette quantité de données. Par conséquent, plusieurs algorithmes parallèles ont été dévellopés afin de gérer cette masse d'informations et de données [29], [30].

1.4.1 les algorithmes parallèles exacts

En 1999, Zaki [31] a fait une étude détaillée sur les algorithmes parallèles d'extraction des règles d'association. En outre, il les a classés par rapport à la plateforme utilisée (mémoire distribuée ou partagée). Depuis 1999 et avec l'apparition de la programmation GPU en 2007, plusieurs algorithmes parallèles ont été développés. Dans cette section, certains de ces algorithmes sont présentés, analysés et classés selon la plateforme utilisée (mémoire distribuée, partagée et la téchnologie GPU).

Les algorithmes basés sur mémoire distribuée

Le systéme basé sur mémoire distribuée est un systéme où chaque processeur a une mémoire privée. La communication entre processeurs est réalisée par échange de messages [31]. Un meilleur algorithme parallèle, sur le système de mémoire

distribuée, consiste à trouver une bonne décomposition des données et à minimiser la communication entre processeurs [29], [30]

Dans [32], Agrawel et Shafer développent deux algorithmes parallèles basés sur Apriori. Dans l'algorithme CD (Count Distribution), la base de transactions est divisée en plusieurs parties où chaque processeur est chargé d'une partie de la base. Chaque processeur génère, comme Apriori, complètement les itemsets candidats et calcule leurs supports en utilisant sa propre partie. Ensuite, tous les processeurs calculent les supports globaux de tous les candidats par échange de message afin d'extraire les itemsets fréquents. Le deuxième algorithme est appelé DD (Data Distribution), les candidats sont partitionnés sur tous les processeurs de telle sorte que chaque processeur calcule les supports des candidats qui lui sont assignés, en utilisant toute la base de transactions. De plus, l'affectation des candidats aux processeurs est réalisée avec la stratégie round robin.

IDD (Intelligent Data Distribution) est développé par Ein, George et Kumar [33], c'est une amélioration de l'algorithme DD. Il réduit le coût de communication entre processeurs causé par l'algorithme DD. En effet, dans l'algorithme IDD, la communication se fait en utilisant la stratégie par anneau. Chaque processeur envoi le message seulement à ses voisins gauche et droit, et ce n'est pas à tous les processeurs en même temps. Ce processus répete récursivement jusqu'à ce que le message envoyé revient au processeur source. La détermination des voisins gauche et droit est faite par l'identificateur de processeur. En utilisant la téchnique par anneau, les messages sont diffusés tout en minimisant le coût de communication ainsi que le nombre de ports occupés. En outre, IDD permet de résoudre le problème de redondance des itemsets candidats en faisant une partition intelligente de chaque itemset C_k à tous les processeurs. En effet, IDD partitionne C_k sur les processeurs de telle façon que chaque processeur obtienne les itemsets commençant seulement avec un sous-ensemble de tous les items possibles. Par conséquent, il parcoure la base des transactions en s'intéressant uniquement aux transactions contenant un élément de cet sous-ensemble.

Un autre algorithme basé sur Apriori, développé par pan et sun [34], est appelé DDMS (Data Distribution Master Slaves) en utilisant une architecture maître-esclaves entre les nuds. Initialement, le processeur maître trouve les 1-itemsets fréquents en parcourant la base de transactions, aprés il génère et divise les 2-

itemsets candidats en n parties égales, puis envoie ces parties à tous les autres processeurs (les esclaves). En revanche, chaque processeur esclave parcourt toute la base de transactions afin de calculer le support de ses itemsets-candidats locaux et envoie ses propres itemsets fréquents locaux au processeur maître. Par la suite, le processeur maître génère des candidats itemsets pour la prochaine itération à partir de tous les itemsets fréquents de tous les processeurs. Ce processus doit être répété jusqu'à ce que les candidats itemsets soient vides. DDMS optimise parfaitement le coût de communication produit par l'algorithme DD.

Un algorithme PDM [35] est la version parallèle de l'algorithme DHP, comme dans l'algorithme CD, la base de transactions est divisée sur l'ensemble de processeurs et dans chaque étape k, tous les candidats C_k sont considérés par tous les processeurs. PDM utilise une méthode appelée **scrutation** pour obtenir les 2-itemsets fréquents, et dans chaque passe, il supprime les itemsets non fréquents dans la base de transactions.

Shen et Li proposent un nouvel algorithme (MSFPTree) basé sur l'algorithme FPTree [36] en utilisant une architecture maître-esclaves et MPI pour passer le message. Dans un premier parcours de la base de transactions, un processeur maître trouve les 1-itemsets fréquents et les envoie aux processeurs esclaves. Chacun de ces processeurs parcourt ses bases de transactions locales afin de trouver ses propres itemsets fréquents. Enfin, le processeur maître fussionne les itemsets fréquents de tous les processeurs locaux afin d'obtenir tous les itemsets fréquents. Chaque processeur bénéficie d'une structure appelée **conditional pattern** tout en optimisant le coût du calcul des itemsets fréquents.

Les algorithmes basés sur mémoire partagée

Un systéme de mémoire partagée est un systéme où tous les processeurs accèdent à une mémoire commune. Un meilleur algorithme parallèle sur le système de mémoire partagée permet la réduction de faux partages (l'accès concurents de plusieurs processeurs aux variables communes) et aussi la minimisation de synchronisation entre processeurs [30].

Zaki et d'autres ont développé deux approches CCPD (Common Candidates Partitioned Database) et PCCD (Partitioned Candidate Common Database) [37].

Ces deux algorithmes sont les versions des algorithmes CD et DD dans le système de mémoire partagée [32]. CCPD utilise une structure appellée **common hash tree** qui est commmune à tous les processeurs afin de générer les itemsets candidats. Tandis que la base de transactions est divisée logiquement entre les processeurs. La structure **hash tree** est construite en parallèle, alors, chaque processeur traverse sa base de transactions locale et calcule le support de chaque itemsets candidats. Dans PCCD, chaque prosseceur construit sa propre structure appelée **local hash tree**, cependant, la base de transactions est traversée par tous les processeurs afin de générer les itemsets candidats. Chaque processeur calcule les supports des itemsets candidats qui sont uniquement inclus dans sa structure **local hash tree**.

Osmar et d'autres [38] développent une version parallèle de l'algorithme FP-growth appelée MLFPT (Multiple Local Frequent Pattern tree). Il est principalement constitué de deux étapes : construction parallèle des arbres locaux des itemsets (un arbre pour chaque processeur) et l'extraction des itemsets fréquents à partir des arbres des itemsets. La construction des arbres locaux des itemsets est faite en deux phases, chaque phase requiert un parcours complet de la base de transactions. Dans le premier parcours, l'ensemble de transactions est divisé entre les processeurs, chacun d'eux énumère localement les items apparaissant dans ses propres transactions. Ensuite, les supports des candidats sont calculés en parallèle où tous les processeurs prennent en charge un nombre égal d'items. Finalement, les items non fréquents sont éliminés et les items fréquents restants sont triés selon l'ordre de leur supports. Dans le second parcours, chaque processeur récupére le même nombre de transactions afin de construire son propre arbre local des itemsets comme dans [10]. Avant l'étape d'extraction des itemsets fréquents et pour bien équilibrer la charge de travail entre les processeurs, les items sont triés suivant l'ordre de ses supports globaux et sont assignés consécutivement aux processeurs. Ensuite, les itemsets fréquents locaux ne sont obtenus que dans FP-growth. A la fin une fusion des itemesets fréquents locaux est faite pour déterminer les itemsets fréquents globaux.

Yang et d'autres [39] développent un nouveau algorithme parallèle, basé sur FP-growth, appelé PBFIMiner. Il utilise une matrice appelée **Object-Bits** qui est sous forme de bits pour représenter les transactions et accélérer par la suite le

calcul des supports des itemsets. Chaque entrée dans **Object-Bits** représente une seule transaction par une chaine binaire, dont sa longueur est égale aux nombres des 1-itemsets fréquents. La valeur d'un itemsets fréquent $\mathbf{x}$ dans $Object - Bits[i]$ est égale à 1 si la transaction t_i est incluse dans $\mathbf{x}$, 0 sinon. PBFIMiner s'effectue en deux étapes comme l'algorithme MLFPT (construction des arbres des itemsets en parallèle et l'extraction d'itemsets fréquents en parallèle). Premièrement, la base de transactions est divisée sur l'ensemble des processeurs, chaque processeur p_i parcours sa base locale (D_i) pour trouver les 1-itemsets fréquents. Ensuite, il construit son arbre local des itemsets en utilisant directement la structure **Object-Bits** et ce n'est pas sa base de transactions D_i. Après cette première étape, chaque processeur applique le même processus d'extraction des itemsets fréquents que dans l'algorithme MLFPT.

Les algorithmes basés sur La téchnologie GPU

Comme on a vu dans les sections précédentes, le problème d'extraction des règles d'association a été parallelisé de plusieurs façons, les auteurs prennent les avantages de la platforme utilisée. Cependant, tous ces algorithmes ont été implementés sur des platformes trés coûteuses (clusters, des supers calculateurs...)

Depuis 2007, La téchnologie GPU a été utilisée dans plusierus domaines d'applications grâce à sa rapidité et sa disponibilité, (moins coûteuse par rapport à d'autres architectures parallèles). pour celà, la communauté des chercheurs d'extraction des règles d'association ont beaucoup investis sur cette téchnologie.
Dans ce qui suit, on présentera l'architecture de la téchnologie GPU ainsi que quelques algorithmes parallèles d'extraction des règles d'association sur GPU.

GPU est une simple architecture composée principalement de deux parties :

- La partie CPU : Elle comporte une mémoire centrale et un seul processeur. elle se communique avec le GPU à l'aide des bus de communication.
- La partie GPU : c'est un ensemble de blocs, chaque bloc contient un ensemble de threads qui s'exécutent en parallèle. Chaque bloc de threads possède une mémoire partagée qui est accéssisible par tous les threads. On possède aussi une mémoire globale de grande taille qui est accessisible par

l'ensemble de blocs du GPU.

De nombreux algorithmes d'extraction des règles d'association ont été implémentés sur la téchnologie GPU.

Wenbin et d'autres ont proposé deux autres versions parallèles de l'algorithme Apriori dites PBI(Pure Bitmap Implementation) et TBI (Trie Bitmap Implementation) réspectivement [8]. Dans PBI, les transactions et les itemsets sont représentés par deux matrices sous forme de **Bitmap**. Les itemsets est un bitmap (n * m) où n est le nombre de k itemsets et m est le nomnbre d'items, par ailleurs un bit (i, j) = 1 si l'itemset i contient l'item j, 0 sinon. De même, la base de transactions est un bitmap (n * m) où n est le nombre d'itemsets et m est le nombre de transactions, ici un bit (i, j) = 1 si la transaction j contient l'itemset i, 0 sinon. La génération des itemsets candidats est effectuée en deux étapes. chaque processeur génère ses propres candidats en faisant une opération **or** entre chaque bit des deux (k-1)-itemsets fréquents. Chaque processeur p_i effectue en parallèle l'opération **and** entre les bits de ses transactions (deux à deux) afin de trouver les k-itemsets fréquents. Alors que dans TBI, une structure appelée **trie** est adoptée pour représenter les itemsets, c'est un arbre dont les 1-itemsets sont insérés à la profondeur 1 ensuite, les 2-itemsets sont insérés à la profondeur 2 et ainsi de suite. Pour générer des k-itemsets à profondeur k, tous les (k-1)-itemsets se joignent deux à deux en faisant l'opération **or** que dans PBI.

Dans [40], un algorithme appelé GPU-FPM est proposé, il est basé sur l'algorithme Apriori-Like Apriori-Like. Comme Apriori-Like, il utilise une représentation verticale des données en raison de taille limitée de la mémoire des blocs de threads en GPU. GPU-FPM donne un temps meilleur de 10 à 15 fois par rapport à la version séquentielle de Apriori-Like.

Syed et d'autres dans [41] ont proposé une version parallèle de l'algorithme Apriori sur GPU. Dans la première étape, la génération d'itemsets se fait dans les blocs de GPU, où chaque bloc de threads calcule le support d'un ensemble d'itemsets. Dans la deuxième étape, les itemsets générés sont renvoyés à la CPU afin de produire les règles correspondantes à chaque itemset et de déterminer la confiance de chaque règle. Le principal inconvénient de cet algorithme est le coût considérable des communications CPU/GPU. Le temps d'exécution de cet algorithme est 20 fois meilleur par rapport à la version séquentielle.

Dans [42], les auteurs ont proposé une version parallèle de l'algorithme DIC, dans lequel les deux opérations les plus fréquentes de DIC (intersection des items et le calcul des supports) sont parallélisées. Dans cet algorithme, deux stratégies ont été proposées : **transaction wise approach** appelée "tw" et **candidates wise approach** appelée "cw". Dans "tw", tous les blocs GPU travaillent sur le même candidat et chaque thread est en charge d'une partie des transactions. Alternativement, dans "cw", de nombreux candidats sont traités par la GPU en même temps. Un ensemble de candidats est affecté à chaque bloc de threads. Lorsque le nombre de transactions est tellement grand, la stratégie "wc" est meilleure que "wt". Toutefois, lorsque la longueur des candidats dépasse la taille de la mémoire partagée de chaque bloc, la stratégie "wt" est la plus appropriée.

Un autre algorithme Cuda-Apriori est proposé dans [44]. Tout d'abord, l'ensemble de transactions est réparti entre différents threads. Ensuite, k-itemsets candidats sont générés et attribués à la mémoire globale de GPU. Chaque thread calcule le suppport d'un seul candidat en utilisant uniquement la partie des transactions affectée à son bloc. A chaque itération, une synchronisation entre les blocs de GPU est faite pour calculer le support global de chaque candidat.

Dans [45], GPApriori est proposé dont le but d'accélérer le calcul des supports des itemsets candidats. La génération des candidats et l'extraction des itemsets fréquents se font en CPU alors que pour le calcul des supports des itemsets candidats se fait en GPU où chaque bloc de threads est occupé par un seul candidat. Pour une petite base de transactions, cet algorithme est 100 fois meilleur que Apriori. Néanmoins, l'amélioration est réduite lorsque le nombre de transactions augmente à cause de la divergence des threads.

Dans [46] un autre travail d'extraction des itemsets fréquents maximaux sur GPU est rapporté. Ce travail est basé sur les principales propriétés de la GPU qui sont le parallélisme et l'indépendance des données. Une structure d'arbre est utilisée pour stocker les itemsets fréquents où chaque noeud contient un itemset, son support et une bitmap de chaque itemset. Une bitmap est une structure booléenne qui représente toutes les transactions contenant un itemset donné. Cet algorithme fonctionne bien sur des grandes bases de transactions. Néanmoins, il nécessite un espace mémoire élevé et un nombre important de pointeurs qui sont difficiles à gérer sur l'architecture GPU.

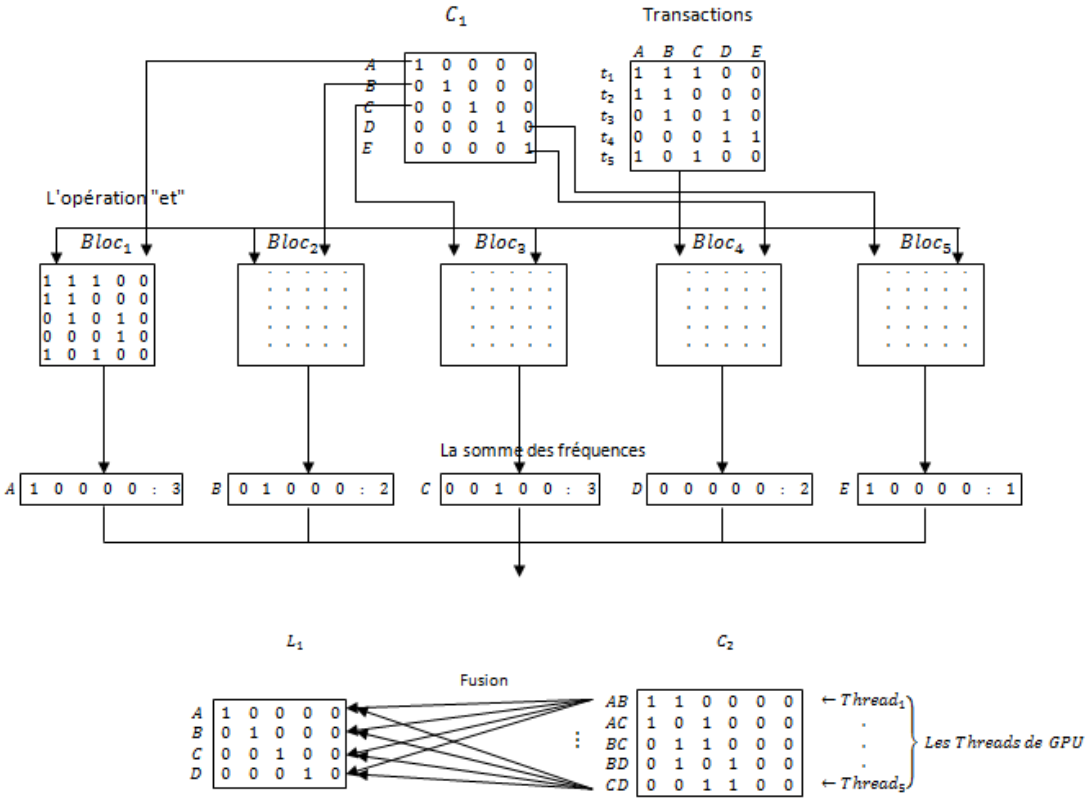


FIGURE 1.3 – *Exemple de l'algorithme PBI*

Jiayi et d'autres [47] implémentent deux packages pour exécuter Apriori sur GPU. Le premier appelé **Mempack**, il utilise une structure de données compacte pour transformer les itemset générés de la CPU au GPU. En outre, il fournit une méthode puissante pour le calcul des supports sur GPU. Le deuxième appelé **CLP**, il réduit le nombre d'accés mémoire de chaque thread alloué sur GPU. En utilisant ces deux packages, l'amélioration de Apriori sur GPU ne dépasse pas 15 fois que la version séquentielle de Apriori en raison de la complexité des structures de données utilisées. La Figure 1.3 explique le déroulement de l'algorithme PBI jusqu'à obtenir les candidats 2-itemsets sur l'ensemble de transactions présenté dans la Table 1.1.

1.4.2 Les algorithmes parallèles approchés

Avec la quantité gigantesque de données qu'on dispose actuellement dans le web (les réseaux sociaux, les services web...), les algorithmes parallèles exacts déja presentés ne traitent pas ces données en temps réel. Pour celà, plusierus algorithmes d'extraction des règles d'association approchés parallèles ont été dévellopés.

Dans [49], Melab et d'autres proposent un algorithme parallèle basé sur les algorithmes génétiques appelé PGARM pour l'extraction des règles d'association, il est basé sur le model Maître/esclaves. premièrement, le processeur maître affecte des sous-populations aux autres processeurs esclaves en utilisant la stratégie de round robin. Puis, chaque esclave applique l'algorithme génétique classique sur sa propre sous-population et retourne au processeur maître les règles obtenues. Cet algorithme comporte principalement deux inconvénients : d'une part, toutes les transactions sont dupliquées en chaque processeur ce qui nécessite un espace de mémoire important. D'autres Part, des règles inadmissibles peuvent être générées.

Dans [50], deux versions basées sur le modéle d'islande [48] d'un nouveau algorithme appelé PMOGA ont été proposées. Dans la première version, chaque processeur exécute indépendament l'algorithme génétique dont lequel il crée sa propre population, applique les opérations de mutation, croisement et séléction afin de reproduire une nouvelle population. Ce processus est répété pour un nombre déterminé de générations. A la fin, un processeur particulier collecte les meilleures règles locales de chaque processeur pour créer les meilleures règles globales. En outre, dans la deuxième version, chaque processeur envoi les meilleures règles obtenues aux processeurs voisins et ça se fait pour chaque génération. En utilisant cet algorithme, chaque processeur permet de rafiner sa recherche en se basant sur d'autres informations supplémentaires sur l'espace de solutions. Cependant, il y'a une possibilté où deux processeurs cherchent dans la même région ce qui produit une certaine redondance.

Une autre version des algorithmes génétiques appelée IPMOGA, est proposée par Mishra et d'autres dans [51]. L'espace de solutions est divisé en differentes régions dont lequel chaque région est explorée par un seul processeur afin de trouver les meilleures règles situées dans sa propre région. A la fin, les meilleures règles

de chaque région sont échangées pour calculer les meilleures règles globales. La difficulté de cet algorithm consiste à définir une stratégie efficace de partitionnement de l'espace de solutions dans le but de trouver des régions complètement disjointes.

1.5 Classification des algorithmes des règles d'association

Dans la litérature, il existe plusieurs classifications des algorithmes d'extraction des règles d'association [31], [30], [29]. Dans ce qui suit, on présente quelques classifications existantes ainsi, on propose une autre classification des algorithmes d'extraction des règles d'association.

Commençant par la classsification des algorithmes séquentiels, Garg et d'autres [30] ont établi une claassification des algorithmes séquentiels selon le type de représentation de transactions (verticale ou bien horizontale) et la structure de données utulisée lors de calcul des itemsets fréquents (Hash Tree ou bien Trie).

En 1999, Zaki et d'autres [31] proposent une classification des algorithmes d'extraction des règles d'association parallèles selon l'architecture utilisée (partagée ou distribuée), ils les classent aussi selon le type de parallélisation (parallélisation de données ou bien parallélisation des tâches).

Une autre classification des algorithmes parallèles a été proposée en 2009 par Mishra et d'autres [30]. La classification se base à la version séquentielle dans lequel l'algorithme parallèle est inspiré.

Nous pouvons diviser les algorithmes d'extractions des règles d'associations en deux catégories (séquentiels et parallèles).

Pour les algorithmes séquentiels, on distingue deux catégories d'algorithmes : La premirére catégorie : Les algorithmes exacts qui sont généralement trés coûteux en terme de calcul et de l'espace mémoire. Dans cette catégorie, il y'a des algorithmes (générer et tester) qui génèrent d'une manière récursive les itemsets et qui testent ces itemsets pour savoir s'ils sont fréquent ou pas. D'autres algorithmes (diviser pour régner), construisent l'arbre des itemsets et explorent d'une maniére récursive cet arbre pour trouver les itemsets fréquents. La deuxième

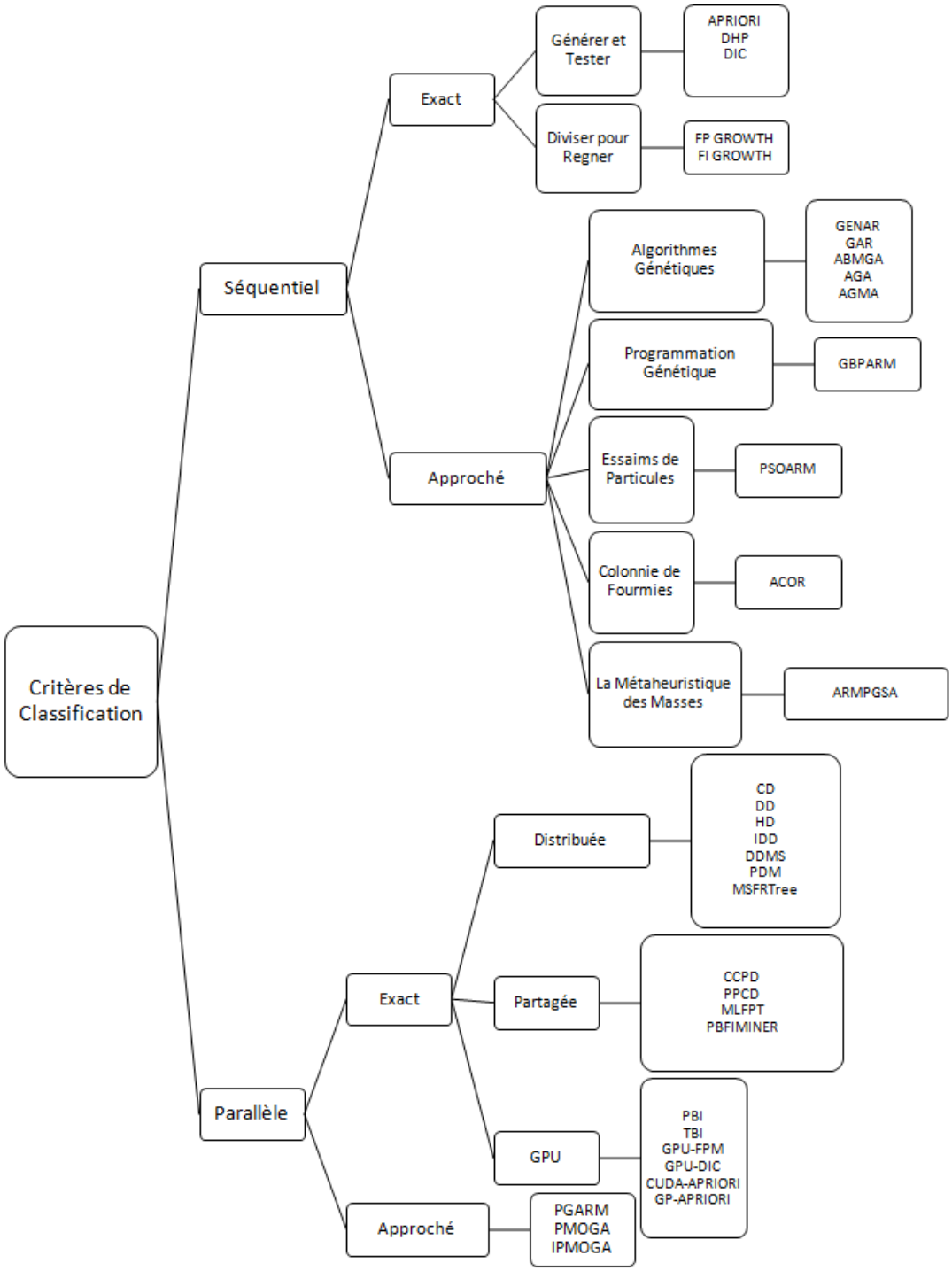


FIGURE 1.4 – *Une nouvelle classification des algorithmes d'extractions des règles d'association*

catégorie : Les algorithmes approchés qui sont généralement meilleurs que les algorithmes exacts en terme de temps du calcul. Cependant, tous ces algorithmes donnent des solutions approchées c'est à dire une partie des règles parmi toutes les règles pertinentes. Ces algorithmes peuvent générer des règles inadmissibles. A titre d'exemple, un item qui appartient à la partie antécédente et la partie conséquente d'une même règle. Ainsi, il y'a trop de paramètres à gérer par ces algorithmes. On peut classer cette catégorie par rapport à la métaheuristique utilisée lors de l'exploitation de l'espace des règles.

De même, pour les algorithmes parallèles, on peut les classer en deux catégories : La première catégorie : les algorithmes exacts parallèles qui sont implémentés sur differentes architectures (mémoire distribuée, mémoire partagée, ou la téchnologie GPU). Ces algorithmes réduisent le temps d'exécution des algorithmes exacts. Cependant, ils ne traitent pas le problème d'extraction des règles d'association en temps réel. La deuxième catégorie : ce sont les algorithmes parallèles approchés. D'aprés notre connaissance, peu d'algorithmes ont été developés et sont basés sur les algorithmes génétiques.

La Figure 2.1 présente notre classification des algorithmes d'extraction des règles d'association.

1.6 Conclusion

Dans ce chapitre, on a présenté les algorithmes d'extraction des règles d'association. Ces algorithmes peuvent être divisés en deux catégories séquentielles et parallèles. dans ces deux catégories, on a des algorihmes exacts et des algorirhmes basés sur métaheuristiques. Les algorithmes basés métaheuristique s'exécutent dans un temps raisonnable en les comparant aux algorithmes exacts. Cependant, ils trouvent juste une partie de règles de toutes les règles pertinentes. Pour celà, notre première contribution qui sera présentée dans le quatriéme chapitre est d'améliorer la qualité des règles obtenues par le processus d'extraction des règles d'association tout en basant sur le principe des algorithmes approchés.

Dans le chapitre suivant, on va présenter un état de l'art sur la deuxième contribution de cette thèse qui est la fouille et la réduction des règles d'association.

Chapitre 2

La fouille et la reduction des règles d'association

2.1 Introduction

Le processus d'extraction des règles d'association génère un nombre important de règles. Par conséquent, l'exploitation de ces règles par l'utilisateur devient difficile et parfois impossible. Pour celà, plusieurs algorithmes d'extraction des règles d'association ont été proposés pour trouver un ensemble de règles trés réduit. A titre d'exemple, on cite les algorithmes Closest [52], MAFIA [53], qui consistent de trouver les itemsets fréquents maximums, un itemset fréquent maximum est une petite partie de tous les itemsets fréquents. On cite aussi l'algorithme GUHA [54] qui consiste à trouver les règles généralisées.

Une autre approche peut être considérée pour minimiser les règles obtenues par le processus d'extraction est d'utiliser les métaheuristiques, cette approche a été bien expliquée dans le premier chapitre.

Malgré tous ces algorithmes pour optimiser le processus d'extraction des règles d'association, le nombre des règles d'association résultant reste assez important. Un processus de fouille de ces règles d'association est nécessaire afin de réduire l'espace des règles.

Dans ce chapitre, on parlera sur deux aspects totalement liés, le premier aspect est la fouille des règles d'association, en présentant deux tâches de fouille qui

Minimum support (%)	BMP-Web-View1	BMP-Web-View2	IBM-Artificial
1.00	87	108	396
0.80	122	203	552
0.60	195	452	1110
0.40	404	1504	4278
0.20	1516	12665	44315
0.10	10360	119335	91174
0.08	33514	230996	118254
0.06	3011836	510233	148255
0.04	Bloqué	1096720	204439
0.02	Bloqué	10448483	413478
0.01	Bloqué	Bloqué	1376684

TABLE 2.1 – Le nombre de règles générées par l'algorithme Apriori en variant le minimum support

sont la classification des règles d'association et l'extraction des meta-règles. Le deuxième aspect est la réduction des règles d'association, on présentera deux catégories d'approches, la première basée sur les connaissances de domaines par contre la deuxième est basée sur l'extraction des connaissances.

2.2 Motivation

Dans cette partie, on va démontrer l'importance des règles générées par le processus d'extraction des règles d'association. La Table 2.1 montre que le nombre de règles générées augmente expotionnelement en diminuant le minimum support. De plus, par l'algorithme Apriori et avec un minimum support égal à 0.06 le nombre de règles dépasse les trois millions de règles.
Ce résultat nous ramène à une autre dimension qui est l'espace des règles. Une étude des règles d'association est nécessaire, dans ce stade, un autre processus de fouille des règles d'association est établi afin de réduire l'esapce des règles. Ce qui suit on détaillera les approches de fouille des règles ainsi des approches de réduction des règles d'association.

2.3 La classification non supervisée des règles d'association

Comme on a déja vu dans la section précédente, aprés le processus d'extraction des règles d'association, on génère un ensemble important de règles que l'utilisateur ne peut pas tous les exploiter. Une approche pour résoudre ce problème est de grouper les règles similaires dans le même cluster en utilisant la classification non supervisée. Ce regroupement permet une meilleure visualisation de l'espace des règles.

Dans la litérature [55] [56], [57] et [58], on distingue principalement deux catégories des méthodes de classification non supervisée des règles (hiérarchique et par partition), ce qui suit on détaillera ces deux catégories. Avant celà, on présente deux exemples qui vont être utilisés dans l'explication par la suite :
Exemple1 On considère l'ensemble de six règles suivantes :
$r_1 : A, B \Rightarrow C.$
$r_2 : A \Rightarrow B.$
$r_3 : B \Rightarrow C.$
$r_4 : B \Rightarrow A.$
$r_5 : A \Rightarrow C.$
$r_6 : A, D \Rightarrow C.$

Exemple2 On considère l'ensemble d'items {A, B, C, D, E} et sept transactions definies comme suit :
t_1 : A, B, C.
t_2 : A, B.
t_3 : A, C.
t_4 : A, B, E.
t_5 : B, C, D.
t_6 : A, B.
t_7 : A, B, D.
Supposant, aprés le processus d'extraction des règles d'association, on aura les règles suivantes : $r_1 : A \Rightarrow B.$
$r_2 : A \Rightarrow C.$

$r_3 : B \Rightarrow C.$
$r_4 : A, B \Rightarrow C.$
$r_5 : B \Rightarrow D.$
$r_6 : D \Rightarrow A.$

2.3.1 La classification non supervisée hiérarchique

Elle forme des groupes hiérarchiques des règles d'association pour fusionner ou bien diviser récursivement des clusters existants dans un nouveau cluster. Parmi les algorithmes de classification non supervisée hiérarchique, on distingue :

- **Agnes-Rules [55] :**
  Etant donnée N_R règles, Il commence avec N_R clusters, chacun contient une seule règle. Dans chaque itération, une paire de clusters les plus proches est regroupée afin de créer un nouveau cluster plus grand. Ce processus doit être répété jusqu'à ce qu'on obtienne un seul cluster contenant toutes les règles. Le résultat est un arbre hiérarchique, avec les règles individuelles qui sont mises dans les feuilles, les autres noeuds représentent des clusters de deux règles ou plus. La matrice *dist* contient toutes les distances entre les clusters. La fonction *Minimum(dist,* $C_{select-i}$*,* $C_{select-j}$*)* récupère la valeur minimale de la matrice *dist*, retourne les clusters associés à cette valeur et les met dans $C_{select-i}$ et $C_{select-j}$ réspectivement. La fonction *Fusionner-Clusters (*C_{novel}*,* $C_{select-i}$*,* $C_{select-j}$*)* permet de créer un nouveau cluster C_{novel} et de mettre les règles de $C_{select-i}$ et de $C_{select-j}$ réspectivement dans C_{novel}, à la fin de cette fonction les clusters $C_{select-i}$ et $C_{select-j}$ sont écrasés. De plus, la similarité entre deux clusters se fait de la manière suivante :

$$Similarite(C_1, C_2) = \frac{\sum_{i=1to|C_1|} \sum_{j=1to|C_2|} Distance(r_i, r_j)}{|C_1| \times |C_2|} \tag{2.1}$$

  $\forall (r_i, r_j) \in C_1 \times C_2$ La distance Distance(r_i,r_j) entre deux règles r_i et r_j est déduite en faisant les étapes suivantes :
  Premièrement, on détermine l'ensemble de transactions qui vérifie r_i et r_j, on les note réspectivement $S(r_i)$ et $S(r_j)$. Ensuite, le calcul de similarité

entre r_i et r_j est la cardinalité de l'union de $S(r_i)$ et $S(r_j)$ moins la cardinalité de l'intersection entre ces deux ensembles. Plus formelment on aura :

$$Distance(r_i, r_j) = |S(r_i) \bigcup S(r_j)| - |S(r_i) \bigcap S(r_j)| \tag{2.2}$$

L'algorithme Agnes-Rules

```
Debut
  Pour i allant de 1 à Nb_Rules faire
     C[i]= r[i]
  fin_pour
  Tanque il y'a plus d'un cluster faire
   Pour i allant de 1 à nombre de clusters
    Pour j allant de i à Nombre de clusters
      dist[i][j]=similarité(C[i], C[j])
    fin_pour
   fin_pour
   Minimum(dist,C[select-i],C[select-j])
   Fusion-Clusters(C[novel],C[select-i],C[select-j])
  fin_tanque
Fin
```

Exemple Considérons l'exemple2, initialement, on affecte chaque règle à un cluster :

C_1=$\{r_1\}$, C_2=$\{r_2\}$, C_3=$\{r_3\}$

C_4=$\{r_4\}$, C_5=$\{r_5\}$, C_6=$\{r_6\}$.

Ensuite, en utilisant la formule similarité (voir l'equation 2.1), la matrice *dist* est obtenue comme suit :

$$\begin{pmatrix} 0 & 5 & 5 & 4 & 5 & 4 \\ 5 & 0 & 2 & 1 & 4 & 3 \\ 5 & 2 & 0 & 1 & 2 & 3 \\ 4 & 1 & 1 & 0 & 3 & 2 \\ 5 & 4 & 2 & 2 & 0 & 1 \\ 4 & 3 & 3 & 3 & 1 & 0 \end{pmatrix}$$

A partir de la matrice *dist*, on extrait la distance minimale qui est égale à 1 entre

le deuxième et le quatrième cluster. Donc, on fusionne ces deux clusters comme suit :
C_7=$\{C_2, C_4\}$=$\{r_2, r_4\}$.
On répete le même processus jusuq'à ce qu'on obtienne un seul cluster. A titre d'exemple, le calcul de similarité entre le cluster C_7 et le cluster C_1 est :
Similarité(C_1, C_7)=$\frac{Distance(r_1,r_2)+Distance(r_1,r_4)}{2}$
= $\frac{5+4}{2}$=$\frac{9}{2}$.
Le résultat final de l'algorithme est :
C_7=$\{C_2, C_4\}$=$\{r_2, r_4\}$.
C_8=$\{C_5, C_6\}$=$\{r_5, r_6\}$.
C_9=$\{C_7, C_3\}$=$\{r_2, r_3, r_4\}$.
C_{10}=$\{C_8, C_9\}$=$\{r_5, r_6, r_2, r_3, r_4\}$.
C_{11}=$\{C_1, C_{10}\}$=$\{r_1, r_2, r_3, r_4, r_5, r_6\}$.

On peut aussi interpreter ce résultat sous forme d'un arbre hiérarchique comme dans la Figure 2.1

– **L'algorithme de Alipio [59] :** Alipio propose trois mesures de similarité entre clusters et une mesure de similarité entre règles. Ensuite, il applique le même raisonement que l'algorithme précédent Agnes-Rules. La mesure de similarité entre deux règles r_i et r_j est définie comme suit :

$$Dist - Alipio(r_i, r_j) = 1 - \frac{\{itemsdansr_i\} \bigcap \{itemsdansr_j\}}{\{itemsdansr_i\} \bigcup \{itemsdansr_j\}} \tag{2.3}$$

Pour le calcul de similarité entre deux clusters , Alipio utilise trois formules (**Average Linkage** est similaire que dans 2.1, **Single Linkage**, et **Complete Linkage**) qui sont définies $\forall (r_i, r_j) \in C_1 \times C_2$ comme :

$$Average - Linkage(C_1, C_2) = \frac{\sum_{i=1to|C_1|} \sum_{j=1to|C_2|} Dist - Alipio(r_i, r_j)}{|C_1| \times |C_2|} \tag{2.4}$$

$$Single - Linkage(C_1, C_2) = \min Dist - Alipio(r_i, r_j) \tag{2.5}$$

$$Complete - Linkage(C_1, C_2) = \max Dist - Alipio(r_i, r_j) \tag{2.6}$$

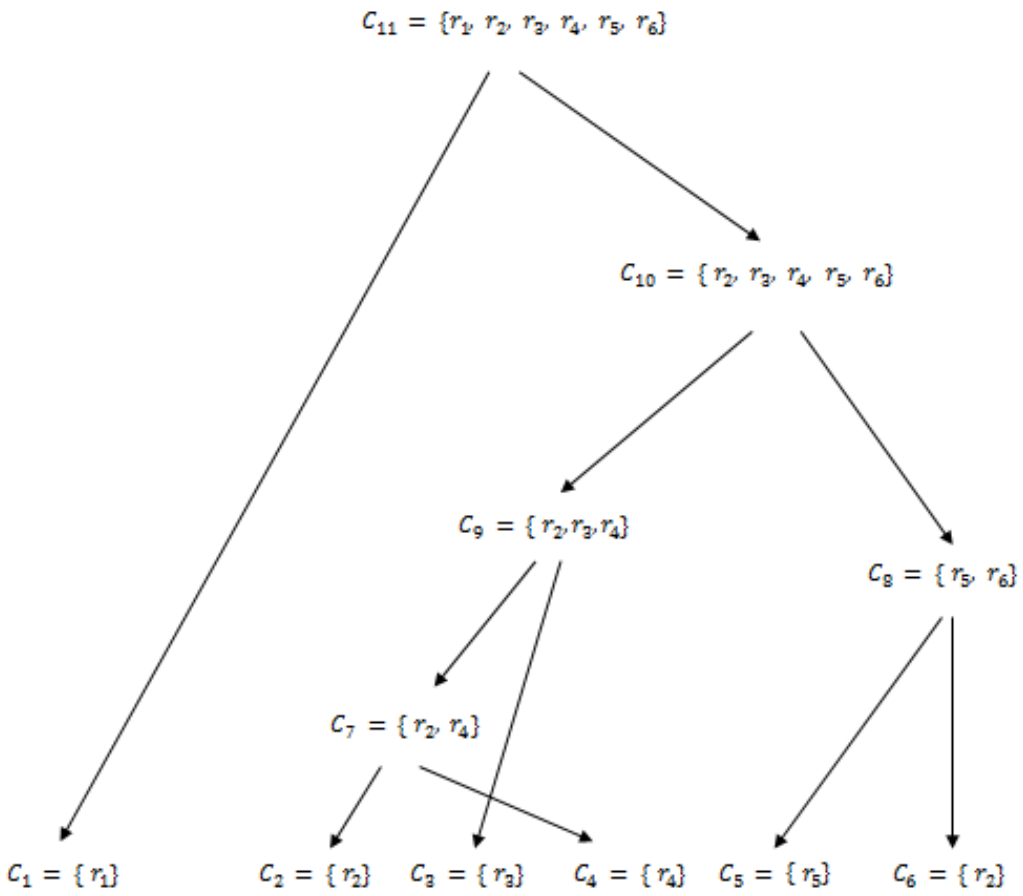


FIGURE 2.1 – *L'arbre hiérarchique des clusters*

Exemple Considérons l'exemple1, la distance de Alipio entre r_1(A, B⇒ C), r_2(A⇒ B) est calculée de la maniére suivante :
items dans r_1={A, B, C}.
items dans r_2={A, B}.
items dans $r_1 \bigcap r_2$={A, B}.
items dans $r_1 \bigcup r_2$={A, B, C}.
Dist-Alipio(r_1, r_2)=1-$\frac{2}{3}$= $\frac{1}{3}$.
Par exemple, on a deux clusters C_1={r_1, r_2} et C_2={r_3}. Le calcul de similarité entre ces deux clusters est :
Average-Linkage(C_1, C_2)=$\frac{Dist-Alipio(r_1,r_3)+Dist-Alipio(r_2,r_3)}{2}$

$=\frac{(1/3)+(2/3)}{2}=\frac{1}{2}$.
Single-Linkage(C_1, C_2)=min $Dist-Alipio(r_1, r_3), Dist-Alipio(r_2, r_3)$
$=\min(1/3), (2/3)=\frac{1}{3}$.
Complete-Linkage(C_1, C_2)=max $Dist-Alipio(r_1, r_3), Dist-Alipio(r_2, r_3)$
$=\max(1/3), (2/3)=\frac{2}{3}$.

2.3.2 La classification non supervisée par partitionement

La Classification non supervisée par partitionement consiste à partitionner les règles dans des clusters en séléctionnant récursivement une règle centre pour chaque cluster, déterminer la partition des autres règles en se basant sur les règles centres.

- **L'algorithme BitOp [60] :**
  Il est développé pour les règles qui ont des attributs continus, cependant, on peut l'appliquer aux règles booléennes. Il groupe les règles d'association qui ont la même partie conséquente selon les items communs de la partie antécédente. Initialement, on divise l'ensemble des règles en plusieurs sous ensembles selon la partie conséquente aprés on applique le même algorithme pour chaque sous ensemble.
  Soit un sous ensemble E qui contient des règles de même conséquente. Premièrement, on transforme les parties antécédentes de ces règles en matrice binaire appelée *bitmap* tel que $bitmap[i][j] = 1$ si le j^{eme} item est présent dans la partie antécédente de la i^{eme} règle de E. Ensuite, on utilise des masques afin de détecter les clusters. En effet, commençant par le premier masque qui est similaire de première ligne de la matrice bitmap. On fait une opération *et* entre ce masque et la deuxième ligne. On répete cette opération jusqu'à obtenir un masque égal à zéro ou bien on termine de parcourir toute la matrice *bitmap*. Ensuite, on recommence de la deuxième ligne de *bitmap*. On répete le même processus et ainsi de suite jusqu'a ce qu'on termine la détection de tous les clusters. L'algorithme suivant explique clairement le principe de cet algorithme pour grouper un sous ensemble de N_R règles qui partagent la même partie conséquente.

L'algorithme BitOp

```
Debut
Entrée: N: le nombre d'items. NBR: Le nombre de rêgles.
Bitmap, Masque: matrices booléenes(N*NBR).
i, indice, : entier.
Output:Clusters: matrices booléenes(N*NBR)
 Pour i allant de 0 à NBR faire
  copier le vecteur Bitmap[i] à Masque[i].
  indice =i+1.
  Tanque indice < NBR et Vecteur_zero(Masque[j]=faux faire
   Masque[indice]=Op_et(Masque[indice-1],Bitmap[indice])
   indice=indice+1
  fin_tanque
  Pour l allant de 0 à N faire
   Si Masque[indice][l]=1 alors
     Clusters[i][l]=1
     Sinon
      Clusters[i][l]=0
   fin_si
  fin_pour
 fin_pour
 Pour i allant de 0 à NBR faire
  Pour j allant de i+1 à NBR faire
   Si inclusion(Clusters[i],Clusters[j]=vrai) alors
   supprimer Clusters[i]
   fin_si
  fin_pour
 fin_pour
Fin
```

Exemple on considére l'exemple2. Si on veut grouper les règles qui ont comme partie conséquente l'item C. Tout d'abord, on extrait les règles qui ont comme partie conséquente l'item C. On aura ces quatre règles :

$r_1 : A, B \Rightarrow C.$

$r_2 : B \Rightarrow C.$

$r_3 : A \Rightarrow C.$

$r_4 : A, D \Rightarrow C.$

Ensuite, On construit Bitmap comme suit :

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

Chaque ligne représente la partie antécédente d'une règle parmi les règles (r_1, r_2, r_3 et r_4) dans l'ordre et chaque colonne représente dans l'ordre la présence (1) ou bien l'abscence (0) des items (A, B, D) dans chaque règle.

Pour chaque règle, on applique des masques afin de détecter les clusters. A titre d'exemple, pour la règle r_1, premièrement, on copie la ligne de r_1 (1,1,0) en première ligne de la matrice Masque. Ensuite, on fait une opération de conjonction(et) entre cette ligne et la deuxième règle (0,1,0). On met le résultat en deuxième ligne de matrice Masque. On fait la même chose pour la deuxième règle, on obtient des zeros donc on s'arrête. Le résultat final de cette première itération est montré dans la matrice masque :

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Le cluster associé à cette iteration est {A, B}. Ce processus doit être répété pour les autres règles (r_2, r_3 et r_4), tous les clusters obtenus sont :

$C_1 : \{A, B\}$

$C_2 : \{B\}$

$C_3 : \{B\}$

$C_4 : \{A, B\}$

$C_5 : \{A, D\}$

On remarque que C_2 et C_3 inclus dans C_1 et C_4, donc on peut garder un seul cluster $C_1 : \{A, B\}$ parmi ces quatres clusters. Le résultat final de l'algorithme est :

$C_1 : \{A, B\}$ ou bien $\{r_1, r_2, r_3\}$

$C_5 : \{A, D\}$ ou bien $\{r_4\}$.

– **L'algorithme Kmeans-IR [61] :**
Il est développé pour les règles d'inductions mais on peut l'utiliser dans le cadre des règles d'association. Les auteurs appliquent le classical Kmeans [62] en proposant une mesure de similarité entre les règles et une formule pour calculer le centre de gravité d'un ensemble de règles d'association.
La mesure de similarité entre deux règles r_1 et r_2 est calculée comme suit :

$$Ration(r_1, r_2) = \frac{|Items_1 \bigcup Items_2| - |Items_1 \bigcap Items_2|}{|Items_1| + |Items_2|}. \tag{2.7}$$

tel que
$Items_i$: représente l'ensemble d'items de la règle r_i.

Tandis que le centre de gravité d'un ensemble de r règles est calculé avec la formule *Freq-centers* comme suit :
Premièrement, la taille l de la règle *centre* est déterminée en calculant la moyenne du nombre d'items des r règles :

$$l = \frac{\sum_{i=1}^{r} |items(r_i)|}{r}$$

Ensuite, Les items de r règles sont triés selon leur fréquence dans les r règles et seulement les r items fréquents sont mis dans un vecteur appelé $Freq$ tel que :
$centre[Freq[j]] = 1$ and $\forall$ i $\neq$ Freq[j] centre[i]=0.
Les principales tâches de l'algorithme Kmeans-IR est le suivant : Premièrement, on choisit K règles parmi l'ensemble de règles telque K est le nombre de clusters. On considère chaque règle choisie r_i comme le centre gravité intial du cluster C_i. Ensuite, pour chaque règle qui n'est pas déja affectée r_j, on calcule sa distance avec chaque règle centre r_i de cluster C_i en utilisant la formule Ration. La règle r_j est affectée au cluster le plus proche de centre qui donne une distance minimale. On recalcule le centre de gravité de chaque cluster en utilisant la formule *Freq-centers*. Ce processus doit être répété jusqu'à ce qu' aucune règle ne change de cluster au bout d'une itération.

L'algorithme Kmeans-IR

```
Debut
Choisir les initials centres pour les K clusters
Tanque le critère d'arrêt non pas atteint faire
 Pour chaque règle r faire
  affecte r en groupe $i$ avec le centre le plus proche (
  en utilisant la fonction Ration)
 end_pour
 Si aucune règle ne change le cluster alors
  terminer et sortir
  Sinon
   Calculer les nouveaux centres de gravité
   en utilisant la formule Freq-centers
 Fin_si
Fin_tanque
Fin
```

Exemple On considère l'exemple1, avec nombre de clusters (K=2). Premièrement, on considère r_1, et r_2 comme centres de gravité de premier cluster C_1 et deuxième cluster C_2 réspectivement. Ensuite, on calcule la distance entre chaque règle et le centre de gravité de chaque cluster en utilisant la distance ration, a titre d'exemple, la distance entre r_3 et le centre de gravité du premier cluster g_1=r_1 est calculé comme suit :

Itemsg_1={A, B, C}.

Itemsr_3={B, C}.

Items$g_1 \bigcup$ Itemsr_3={A, B, C}.

Items$g_1 \bigcap$ Itemsr_3={B, C}.

|Items$g_1 \bigcup$ Itemsr_3|=3.

|Items$g_1 \bigcap$ Itemsr_3|=2.

|Itemsg_1|=3.

|Itemsr_3|=2.

Ration(g_1, r_3)=$\frac{3-2}{3+2}=\frac{1}{5}$.

De la même manière, on calcule les distances entre les centres de gravité et les règles restantes, on aura à la fin de cette première itération cette matrice de distance :

$$\begin{pmatrix} 0 & \frac{1}{5} \\ \frac{1}{5} & 0 \\ \frac{1}{5} & \frac{1}{2} \\ \frac{1}{5} & 0 \\ \frac{1}{5} & \frac{1}{2} \\ \frac{1}{3} & \frac{2}{5} \end{pmatrix}$$

Les colonnes de cette matrice représentent les centres g_1 et g_2, alors que les lignes représentent les règles r_1 jusqu'au r_6 dans l'ordre. Le résultat des clusters à la fin de cette première itération est :

C_1=$\{r_1, r_3, r_5, r_6\}$.

C_2=$\{r_2, r_4\}$.

Aprés, on recalcule les centres de gravité g_1 et g_2 en appliquant la formule $Freq_center stelque$:

On détermine la taille de la règle centre de chaque cluster :

$rC_1 = \frac{3+2+2+3}{4} \approx 2.$

$rC_2 = \frac{2+2}{2} = 2.$

On calcule les items fréquents de chaque cluster :

$Items - Frequents_{C_1}$ =*{ A, B, C , B, C , A, C, A, D, C} ={ (C,4), (A,3), (B,2), (D,1)}.*

$Items - Frequents_{C_2}$ =*{ A, B, A, B} ={ (A,2), (B,2)}.*

On extrait juste les r items fréquents de chaque cluster et on les assigne à sa règle centre :

g_1 =*{A, C}.*

g_2 =*{A, B}.*

On répete le même processus pour le deuxième itération, A la fin on aura cette matrice de distance :

$$\begin{pmatrix} \frac{1}{5} & \frac{1}{5} \\ \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \\ \frac{1}{5} & \frac{3}{5} \end{pmatrix}$$

Pour les règles r_1 et r_3, on remarque que ses distances avec g_1 et g_2 sont égales alors on choisit au hasard un cluster parmi les deux clusters soit C_1. Le résulat final de cette itération est le même que dans la première itération donc on s'arrête et le résultat final de l'algorithme est :
C_1=$\{r_1, r_3, r_5, r_6\}$.
C_2=$\{r_2, r_4\}$.
Remarque : Si on ne choisit pas C_1 pour les règles r_1 et r_3, alors on aura un autre état des clusters qui est different de l'état de la première itération. Par conséquent, on doit réitérer le même processus jusqu'à ce qu'on ne trouve pas un changement de clusters entre deux itérations successives.

- **L'algorithme Medoids-Rules [55]** :

L'algorithme Medoids est une adapatation de l'algorithme Kmeans. La difference entre Medoids et Kmeans consiste dans le calcul de centre de gravité d'un ensemble d'objets. En Medoids, le centre de gravité d'un ensemble d'objet E_o=$\{o_1, o_2, o_3 ... o_p\}$ est un objet o_i inclus dans E_o qui vérifie la contrainte suivante :
$\sum_{o_j \in E_o} Distance(o_i, o_j)$ est minimale.
Medoids-Rules applique le même raisonement que Medoids sauf que le calcul de similarité entre deux règles r_1 et r_2 suit l'equation 2.2.

L'algorithme Medoids-Rules

```
Debut
Pour i allant de 1 à NBR faire
 Pour j allant de 1 à NBR faire
  Distances[i][j]=Distance(r[i], r[j])
 fin_pour
fin_pour
choisir les initials centres des K clusters
```

```
Tanque critére d'arrêt non pas atteint faire
Pour chaque regle r faire
 affecte r en groupe i avec le centre le plus proche
 (en utilisant la matrice Distances)
fin_pour
Si aucune règle ne change le cluster alors
 terminer et sortir
 Sinon
  Pour chaque cluster C[i] faire
   min=m
   Centre[i]=vide
   Pour chaque règle r[p] dans C[i] faire
    somme=0
    Pour chaque règle r[j] dans C[i] faire
     somme=somme+Distances[p][j]
    fin_pour
    Si somme <min alors
     min=somme
     Centre[i]=r[p]
    fin_si
   fin_pour
  fin_pour
fin_si
fin_tanque
Fin
```

Exemple On considére l'exemple2, avec un nombre de clusters qui est égal à deux.

Initialement, on détermine la matrice *Distances* qui contient toutes les distances de l'ensemble de règles d'entrées. A titre d'exemple, pour calculer la distance entre r_1 et r_2 on suit les étapes suivantes :

S(r_1)=$\{t_1, t_2, t_4, t_6 , t_6, t_7\}$.

S(r_2)=$\{t_1, t_3\}$.

S(r_1) $\bigcup$ S(r_2)=$\{t_1, t_2, t_3, t_4, t_6 , t_6, t_7\}$.

$S(r_1) \bigcap S(r_2)=\{t_1\}$.
Distance(r_1, r_2)=|$S(r_1) \bigcup S(r_2)$|-|$S(r_1) \bigcap S(r_2)$|=6-1=5.
De la même maniére, on calcule les autres distances, à la fin de cette première étape on aura la matrice de distances suivante, chaque case de Distances[i][j] représente la distance entre les règles r_i et r_j :

$$\begin{pmatrix} 0 & 5 & 5 & 4 & 5 & 4 \\ 5 & 0 & 2 & 1 & 4 & 3 \\ 5 & 2 & 0 & 1 & 2 & 3 \\ 4 & 1 & 1 & 0 & 3 & 2 \\ 5 & 4 & 2 & 3 & 0 & 1 \\ 4 & 3 & 3 & 2 & 1 & 0 \end{pmatrix}$$

Aprés cette étape, on choisit deux règles soit r_1 et r_2 et on les affecte aux centres des clusters C_1 réspectivement C_2. Ensuite pour chaque règle restante, on détermine sa distance avec g_1 (centre de C_1) et g_2 (centre de C_2). On obtient la matrice *Dist* dont la case Dist[i][j] représente la distance entre la règle r_i et le centre g_j du cluster C_j :

$$\begin{pmatrix} 0 & 5 \\ 5 & 0 \\ 5 & 2 \\ 4 & 1 \\ 5 & 4 \\ 4 & 3 \end{pmatrix}$$

Le calcul des éléments de *Dist* se fait explicitement en utilisant la matrice *Distances*. Par exemple, pour calculer les distances qui séparent g_1=r_1 et les autres règles, on doit juste récupérer la ligne de r_1 de la matrice *Distances* et la mettre dans la colonne de g_1 de la matrice *Dist*. A la fin de cette première itération, le résultat des clusters est :
C_1=$\{r_1\}$.
C_2=$\{r_2, r_3, r_4, r_5, r_6\}$.
Ensuite, pour déterminer les nouveaux centres de gravité, on doit appliquer la formule 2.2 en utilisant la matrice *Distances*. A titre d'exemple, si on veut appliquer la formule 2.2 pour la règle r_2, on additionne la deuxième ligne de la matrice *Distances* qui est égale à 15. On répete le même processus pour les autres règles,

et on considère la règle qui donne un résultat minimum comme un centre de gravité de chaque cluster. En effet, les nouveaux centres sont :
g_1=r_1.
g_2=r_4.
on répete le même processus pour les nouveaux centres, on aura comme matrice *Dist* :

$$\begin{pmatrix} 0 & 4 \\ 5 & 1 \\ 5 & 1 \\ 4 & 0 \\ 5 & 3 \\ 4 & 2 \end{pmatrix}$$

Aucune règle ne change de cluster, donc on s'arrête et le résultat final des clusters est :
C_1=$\{r_1\}$.
C_2=$\{r_2, r_3, r_4, r_5, r_6\}$.

– **L'algorithme Medoids-KNN-Rules [63]**

Afin de réduire la complexité de la classification non supervisée des règles d'association, une approche a été proposée dans [63]. Dans un premier temps, une partie de règles éparpillées est choisie avec l'algorithme génétique. Ensuite, il applique l'algorithme Medoids-Rules sur cette partie de règles. et il place les règles restantes dans les clusters obtenus par l'algorithme KNN [64].
De plus, la distance entre deux règles d'association r_1 :$(x_1 \Rightarrow y_1)$ et r_2 :$(x_2 \Rightarrow y_2)$ est calculée en utlisant la formule suivante :

$$d(r_1, r_2) = (|x_1| + |x_2| - |x_1 \bigcap x_2|) + |y_1| + |y_2| - |y_1 \bigcap y_2|) \tag{2.8}$$

L'algorithme Medoids-KNN-Rules

```
Debut
choisir une partie de règles (list-règles) les plus éloignées
Medoids(list-règles)
Pour chaque règle r restante faire
KNN(r)
```

```
fin_pour
Fin
```

Exemple Considérons l'exemple1, on suppose que les règles choisies aprés l'exécution de l'algorithme génétique sont les quatres premières règles, le nombre de clusters est égal à deux. En appliquant Medoids-Rules avec la mesure de similarité mentionnée par l'equation 2.8, on obtient les clusters suivants :
C_1={r_1, r_3}.
C_2={r_2, r_4}.
Ensuite, on applique l'algorithme KNN (avec $K = 1$) pour placer les deux autres règles r_5 et r_6 de la manière suivante :

On calcule les distances r_5 réspectivement r_6 avec les quatres règles{r_1, r_2, r_3, r_4}.
A titre d'exemple, la distance entre r_5 et r_1 est la suivante :
d(r_5, r_1)=(|{A}|+|{A, B}|-|{A}|)+(|{C}|+|{C}|-|{C}|)
=(1+2-1)+(1+1-1)=3.
Aprés cette étape, on choisi les K règles les plus proches :
Pour r_5 : la règle la plus proche est r_1, par conséquent, $r_5 \in C_1$.
Pour r_6 : la règle la plus proche est r_2, par conséquent, $r_6 \in C_2$.
Le résultat final des clusters est :
C_1={r_1, r_3, r_5}.
C_2={r_2, r_4, r_6}.

2.4 La découverte des meta-règles d'association

La découverte des meta-règles d'association est un processus d'extraction de connaissances à partir d'un ensemble de règles d'association, c'est à dire appliquer le même processus d'extraction des règles d'association sur l'ensemble de règles d'association. Ces nouvelles règles sur des règles sont appelées des meta-règles. Donc, une meta-règle est une règle representant une relation entre un ensemble de règles d'association.

D'aprés nos connaissances et notre recherche sur l'extraction des meta-règles, il y'a un seul travail intéressant qui utilise l'aspect de meta-règles proposé par

Berrado et d'autres [65].

Il consiste de trouver des $1-associations$ entre un ensemble de règles. Les $1-associations$ sont des meta-règles contenant une seule règle dans la partie antécédente et une autre dans la partie conséquente. L'approache facilement resume les relations asymetriques entre deux règles d'association.

2.4.1 La transformation des règles d'association en base de transactions

On Considère un ensemble I de n items $\{I_1, I_2,...,I_n\}$ et un ensemble T de m transactions $\{T_1, T_2,...,T_m\}$. R=$\{r_1, r_2,..., r_p\}$ est un ensemble de p règles qui ont la même partie conséquente et sont découvertes à partir de l'ensemble de transactions T tout en appliquant le processus d'extraction des règles d'association.

On dit que la transactions t_i vérifie la règle r_j ($r_j \subseteq t_i$), si tous les items de r_j sont contenus dans la transaction t_i.

Chaque élément q_j de la nouvelle base de transactions Q=$\{q_1, q_2,..., q_l\}$ / $l \leq m$ contient un sous ensemble de règles de R tel que :

q_j=$\{r_i \in R/\ r_i \subseteq t_j\}$

En d'autres termes, chaque règle est considérée comme un item, aussi chaque transaction de Q est liée à un sous-ensemble de règles de R dans lequel il la verifie.

Si cet sous-ensemble est non vide, il est considère comme une transaction dans Q. De plus, si chaque transaction dans T vérifie au moins une règle de R alors $l = m$.

Transformation des règles en base de transactions

```
Debut
R: Ensemble de règles qui ont une même partie conséquence
T: Ensemble de transactions de m transactions.
Q: initialiser à vide.
Pour chaque règle r dans R faire
 Pour chaque transaction t dans T faire
```

```
  Si r vérifie t alors
   créer une entrée q dans Q
   insérer r dans q
  fin_si
 fin_pour
fin_pour
Fin
```

Exemple Soit l'ensemble de transactions :

t_1 : A, B, C.

t_2 : A, B, C, D.

t_3 : B, D.

t_4 : C, D.

t_5 : A, D.

Aprés le processus d'extraction des règles d'association, on obtient les règles suivantes :

$r_1 : A \Rightarrow B$

$r_2 : A, C \Rightarrow B$

$r_3 : A \Rightarrow C$

$r_4 : A, D \Rightarrow B$

$r_5 : B \Rightarrow D$

$r_6 : C \Rightarrow D$

$r_7 : D \Rightarrow B$

A titre d'exemple, on veut transformer l'ensemble de règles qui ont comme partie conséquente l'item B en une base de transactions appelée Q. Premièrement, on extrait cet ensemble de règles comme suit :

$r_1 : A \Rightarrow B$

$r_2 : A, C \Rightarrow B$

$r_3 : A, D \Rightarrow B$

$r_4 : D \Rightarrow B$

Aprés, pour chaque règle, on séléctionne les transactions qui la vérifie par exemple pour la règle r_1 on aura t_1 et t_2 alors on insère la règle r_1 dans q_1 et q_2. La nouvelle base de transactions Q sera :

$q_1 : r_1, r_2.$

$q_2 : r_1, r_2, r_3, r_4.$

$q_3 : r_4.$

$q_4 : \emptyset.$

$q_5 : \emptyset.$

On remarque qu'il y a des transactions vides, en d'autres termes, elles ne contiennent aucune règle, on peut éliminer ce genre de transactions. La base de transactions Q sera donc :

$q_1 : r_1, r_2.$

$q_2 : r_1, r_2, r_3, r_4.$

$q_3 : r_4.$

2.4.2 L'algorithme One-Association-Meta-Rules

Le but de l'algorithme **One-Association-Meta-Rules** est de trouver les relations de degré 1 entre l'ensemble de règles R. Une relation de degré 1 est une relation entre deux règles d'association. Il applique le même raisonnement que Apriori mais pour deux itérations seulement. La première itération consiste à trouver les $1-itemsets$ fréquents à partir de la base de transactions Q. Ensuite, il génère les $2-itemsets$ candidats tel que chaque $2-itemsets$ est une paire de règles dans R. Puis, il détermine les $2-itemsets$ fréquents à partir de $2-itemsets$ candidats en utilisant $MinSup$. A la fin, il génère des meta-règles à partir de $2-itemsets$ fréquents en utilisant $MinConf$.

Une meta-règle prend la forme $r_i \Rightarrow r_j$ tel que r_i et r_j sont des règles de R. Les meta-règles résultantes fournissent une description des différentes relations existantes entre les règles. Le seuil $MinConf$ pour l'extraction des meta-règles est defini comme l'extraction classique des règles d'association. Le seuil $MinSup$ dans cet algorithme prend la valeur 0 afin de déduire toutes les relations possibles entre les règles. Pour celà, on peut générer des meta-règles directement de $2-itemsets$ candidats et ce n'est pas nécessaire de calculer les $2-itemsets$ fréquents.

Considérons MR $= mr_1, mr_2, \ldots, mr_k$ l'ensemble des k meta-règles obtenu à partir de Q. Si on analyse les meta-règles de MR, on peut facilement comprendre les différentes relations entre les règles d'association de R. De plus, afin de bien

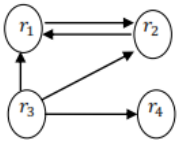


FIGURE 2.2 – *Représentation graphique des meta-règles*

visualiser ces relations entre les règles, une représentation graphique de ces meta-règles est definie. Chaque règle est représentée par un noeud et chaque meta-règle est un arc qui relie deux règles. Le point de départ de l'arc représente la partie antécédent de la meta-règle alors que le point d'arrivée représente la partie conséquente de cette meta-règle.

Exemple Prenons l'exemple précédent, rappelons qu'on a quatres différentes règles $\{r_1, r_2, r_3, r_4\}$ et trois différentes transactions $Q = \{q_1, q_2, q_3\}$. En appliquant l'algorithme **One-Association-Meta-Rules** sur l'ensemble de transactions Q et avec $MinSup = 0\%$ et $MinConf = 50\%$, on aura douze différentes meta-règles. La Table 5.4 présente ces meta-règles avec ces supports et ces confiances.

meta-règles	Antécedent	Conséquente	Support(%)	Confiance(%)
mr_1	r_1	r_2	66	100
mr_2	r_2	r_1	66	100
mr_3	r_1	r_3	33	50
mr_4	r_3	r_1	33	100
mr_5	r_1	r_4	33	50
mr_6	r_4	r_1	33	50
mr_7	r_2	r_3	33	50
mr_8	r_3	r_2	33	100
mr_9	r_2	r_4	33	50
mr_{10}	r_4	r_2	33	50
mr_{11}	r_3	r_4	33	100
mr_{12}	r_4	r_3	33	50

TABLE 2.2 – Illustration des meta-règles

D'aprés la Table 5.4, on remarque qu'il ya beacoup de meta-règles redondantes ($r_1 \Rightarrow r_4$ et $r_4 \Rightarrow r_1$). Afin d'éviter cette redondance, on augmente la valeur de $MinConf$ (soit 100%). On aura juste cinq meta-règles $\{mr_1, mr_2, mr_4,$

mr_8, mr_{11}}. Le graphe de la Figure 2.2 visualise les différentes relations entre les règles. Quand on obtient assez de meta-règles, on aura assez de relations entre les règles. Le graphe est composé de quatre noeuds, chaque noeud est associé à une règle. chaque arc représente une meta-règle parmi les cinq meta-règles intéressantes{mr_1, mr_2, mr_4, mr_8, mr_{11}}. Par exemple l'arc de r_1 à r_2 représente la meta-règle mr_1.

2.5 Réduction de l'espace des règles d'association

Les approches de réduction des règles d'association peuvent être divisées en deux catégories. La première est basée sur les connaissances de domaines, tandis que la deuxième est basée sur la fouille des règles d'association. Ce qui suit, on représente les téchniques les plus utilisées des deux catégories.

2.5.1 Les approches basées sur les connaissances de domaines

Il existe plusieurs travaux sur la réduction de l'espace des règles en utilisant les connaissances de domaines introduites généralement par l'utilisateur.
Dans [66], Les règles pertinentes sont séléctionnées en utilisant un ensemble de constraintes ajoutées par l'utilisateur. Ensuite, un hyper-graphe est construit afin de représenter ces règles.
Dans [67], une approche interactive avec l'utilisateur a été proposée. Les connaissances de l'utilisateur sont interpretées par l'utilisation de trois ensembles de différentes précisions. Les règles délivrées par le processus d'extraction des règles d'association sont comparées avec les trois ensembles de l'utilisateur afin de les grouper dans différentes classes. A la fin, les règles de chaque classes sont triées par une mesure sémantique afin de trouver facilement les règles pertinentes.

Dans [68], l'algorithme CAP est proposé dont le but de la séléction des règles pertinentes. Premièrment, un ensemble de contraintes formalisées en logique de premier ordre est intégré par l'utilisateur. Ensuite, les règles qui vérifient au moins une de ces contraintes sont gardées, par contre les autres sont supprimées. Le pro-

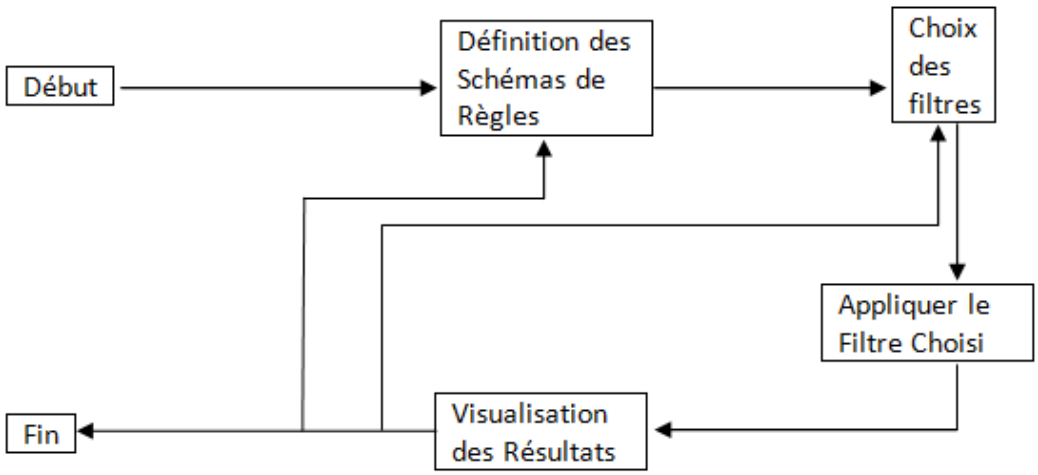


FIGURE 2.3 – *L'architecture de l'algorithme ARIPSO*

cessus est interactive et itérative avec l'utilisateur. En effet, dans chaque itération, on peut supprimer, modifer, ou bien ajouter n'importe qu'elle contrainte.

Aprés l'apparition des ontologies de domaines, qu'on peut les voir comme un autre type de représentation de connaissances. Il avait beaucoup de travaux intéressants qui parlent sur la réduction de l'espace des règles d'association. Dans ce qui suit, on détaillera quelques travaux récemment proposés.

- **L'algorithme ARIPSO [69]** : Marinica et d'autres proposent une approche interactive avec l'utilisateur afin de réduire l'espace des règles d'association. Pour bien formaliser les besoins de l'utilisateur, une ontologie a été conçu. Une autre structure a été employé, appelée *Schémas des règles*, c'est une structure qui contient un ensemble de contraintes introduites par l'utilisateur. De plus, elle est intermédiaire entre l'ontologie et la base de transactions. L'algorithme est itératif de telle sorte que dans chaque itération l'utilisateur peut réviser ces contraintes, modifier l'ontologie ou bien négliger les règles obtenues. La réduction des règles d'association se fait en appliquant les filtres (*MICF, IRF, PRS*). L'architecture globale de l'algorithme ARIPSO est détaillée dans la Figure 2.3.

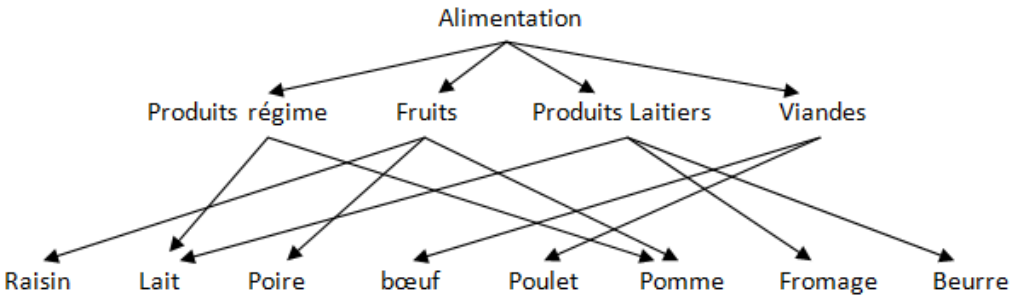


FIGURE 2.4 – *Exemple d'une ontologie*

Tout d'abord, on construit l'ontologie correspondante à l'ensemble d'items. L'utilisateur définit un ensemble de contraintes qui seront placées dans la structure *Schémas des règles*. Par la suite, ce dernier séléctionne un filtre parmi (*MICF, IRF, PRS*). Un ensemble de règles trouvées en appliquant le filtre choisi sont visualisées à l'utilisateur ; Par conséquent, l'utilisateur peut valider les règles obtenues, réviser ces contraintes ou bien séléctionner un autre filtre.
Les filtres sont définis comme suit :

MICF : Séléctionner les règles dont lequelle ses confiances sont supérieurs que les confiances de ces simplifications.
IRF : Tout d'abord, on doit calculer la distance entre les items de la partie antécédente d'une règle et la partie conséquente. La distance entre deux items est le chemin le plus court entre ces deux items dans l'ontologie. A la fin, on choisit juste les règles qui ont une distance inférieure à un seuil déterminé par l'utilisateur.
PRS : On séléctionne les règles vérifiant les contraintes de l'utilisateur.

Exemple Soit une base de transactions qui contient huit items { Raisin, Lait, Poire, Pomme, Poulet, Boeuf, Fromage et Beurre}.
et soit l'ontologie de la Figure 2.4 :
Aprés le processus d'extraction des règles d'association, on obtient les règles

suivantes :
r_1 : Raisin, Poire ⇒ Lait (Conf=85%)
r_2 : Raisin ⇒ Lait (Conf=90%)
r_3 : Poire ⇒ Lait (Conf=83%)
r_4 : Raisin, Poire, Beurre ⇒ Lait (Conf=75%)
r_5 : Fromage ⇒ Boeuf (Conf=77%)
r_6 : Boeuf ⇒ Lait (Conf=72%)

En appliquant le filtre *MICF*, la règle r_1 est supprimée car il existe r_2 qui est une simplification de r_1 ($r_2 \subseteq r_1$) tel que Conf(r_1)<Conf(r_2).
En appliquant le filtre *IRF*, et avec $min - distance = 1$ la règle r_4 est supprimée car distance(r_4)=min(d(Raisin, Lait), d(Poire, Lait), d(Beurre, Lait))= min(4,4,2)=2.
En appliquant le filtre *PRS* et avec la contrainte C_1=<{lait, Fromage}*, Boeuf}>, on peut garder juste les deux dernières règles car les autres règles contiennent des fruits. Alors, elle ne vérfient pas la contrainte C_1.
Un autre algorithme qui suit le principe de ARIPSO est proposé dans[70]. Ici, la différence est de diviser les règles obtenues en quatre groupes comme suit :

nouvelles règles avec une qualité meilleure : Extraire des règles qui ne vérifient pas les contraintes mais ont une qualité meilleure. Ces règles ajoutent une nouvelle croyance à l'utilisateur.

règles connues avec une meilleure qualité : Extraire des règles qui vérifient les contraintes imposées par l'utilisateur et qui ont une meilleure qualité. Ces règles confirment les croyances précédentes de l'utilisateur.

nouvelles règles avec une qualité moindre : Ces règles vont être supprimées car elles ne portent pas de nouvelles connaissances à l'utilisateur.

règles connues avec une qualité moindre : Ces règles représentent les croyances de l'utilisateur qui ne sont pas supportées par la base de transac-

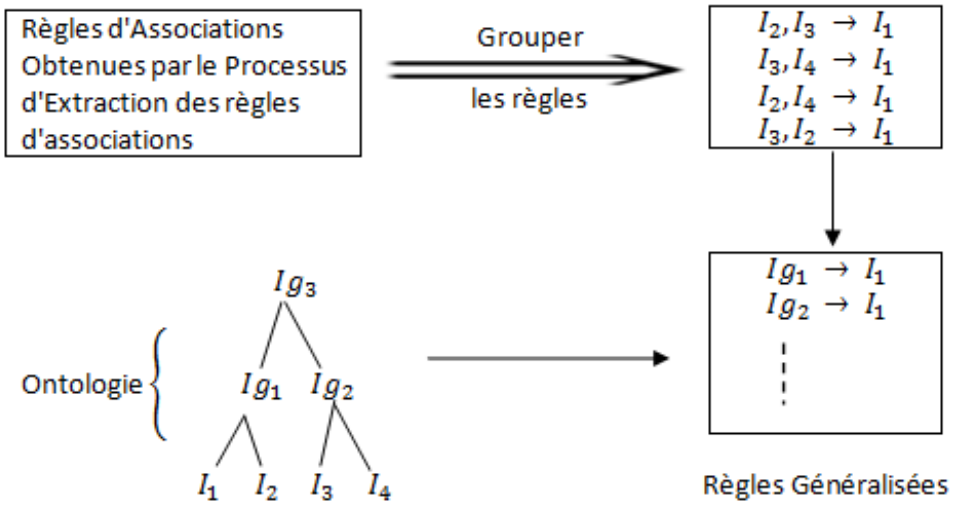


FIGURE 2.5 – *L'algorithme général de GART*

tions. Ces règles sont visualisées à l'utilisateur et par la suite il décide de les garder ou non.

– **L'algorithme de GART [71] :** Le but de l'algorithme GART est de créer des règles généralisées à partir des règles d'association simples. L'utilisateur aura donc une vision générale aux règles obtenues ce qui permet la réduction de l'espace des règles.
La généralisation des règles d'association se fait à l'aide d'une ontologie. Dans un premier temps, on groupe les règles qui ont la même partie antécédente ou bien la même partie conséquente. Ensuite, on crée une ontologie selon la partie inverse de notre groupement. Par exemple, si on veut grouper les règles qui ont la même partie conséquente alors on crée une ontologie pour les items de la partie antécédente. Puis, pour chaque règle on remplace l'item courant par le concept général correspondant. Par conséquent, on aura des règles généralisées réduites, à la place de l'ensemble des règles obtenues. L'architecture générale est décrite dans la Figure 2.5. **Exemple**
Soit les règles suivantes qui ont une même partie conséquente :
Pantacour, Pantafoule ⇒ Chapeau.

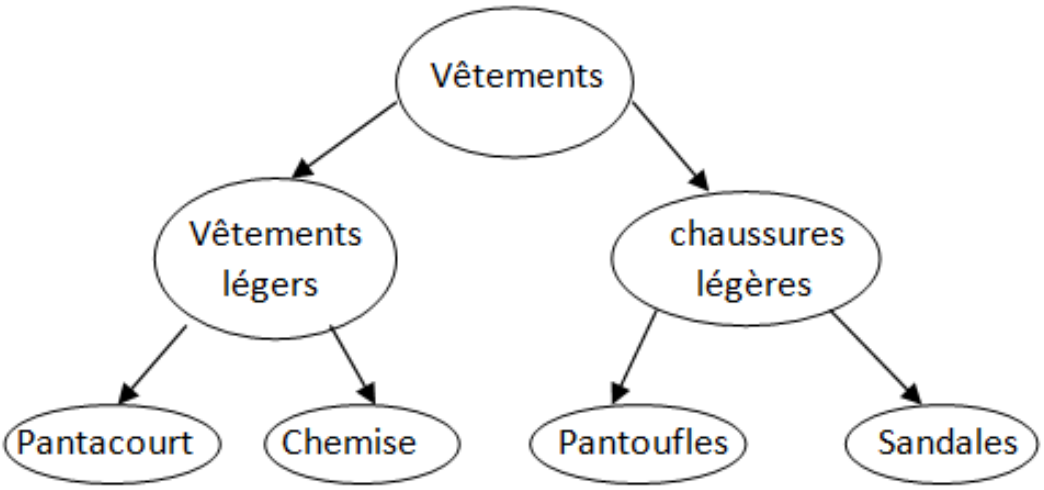


FIGURE 2.6 – *L'ontologie des vetêments*

Sandale, Pantacour ⇒ Chapeau.
Sandale, Chemise ⇒ Chapeau.
Pantafoule, Chemise ⇒ Chapeau.
Et soit l'ontologie de la figure **??** :
Ensuite, on aura la règle suivante qui généralise ces quatres règles :
Vettement légéres, Chausures légéres ⇒ Chapeau.
On remarque bien qu'on a réduit l'espace des règles, passant de quatre à une seule règle.

Malgré que les connaissances introduites par l'utilisateur permettent de réduire l'espace des règles, ces algorithmes restent inefficaces car le processus d'extraction des règles d'association est considéré comme une étape de prétraitement dans plusieurs applications ou l'utilisateur ne peut pas intervenir. Pour cette raison, une autre catégorie d'algorithmes de réduction des règles d'association basée sur l'exctraction de connaissances a été proposée.

2.5.2 Les approches basées sur l'extraction de connaissances

De nombreux algorithmes basés sur l'extraction des connaissances sont proposés afin de réduire l'espace des règles. Dans [72], le nombre d'items est réduit en utilisant la méthode à double échelle. Puis, les items les plus proches sont mis dans la même classe en utilisant l'algorithme *AKMS*. Aprés, les clusters contenant un certain nombre d'items sont considérés et les autres sont supprimés.

L'idée de Watanabe et d'autres [73] consiste à supprimer les règles redondantes durant le processus d'extractions des règles d'association. D'aprés Watanabe, une règle redondante est une règle dont la confiance est inférieure à celle de toutes les règles qui partagent la même partie conséquente avec elle et les parties antécédentes de ces règles sont incluses dans la partie antécédente de la première règle.

Stummer *et d'autres.*. [74] utilisent des structures intelligentes appelées *DG* et *Luxenburger* qui permettent de réduire l'espace de règles tout en gardant toutes les connaissances fournies par ces règles.

Dans [75], les auteurs proposent l'algorithme StatApriori. Ils ajoutent des propriétes statistiques à l'algorithme APRIORI classique afin de supprimer les règles redondantes. Un autre algorithme appelé *PDS* basé sur les propriétés statiqtiques est proposé dans [76]. Premièrement, il définit des corrélations entre la partie antécédente et conséquente de chaque règle par le test de Khi-deux. Puis et avec ces dépendances, il construit un ensemble de règles qui représente parfaitement l'espace de règles d'association.

Tous ces algorithmes sont basés sur les propriétes statistiques entre les règles d'association. Une autre catégorie d'algorithmes de réduction basée sur l'extraction des connaissances a été developée. Elle utilise le principe de meta-règles.

on va détailler l'algorithme One-Associations-Meta-Rules déja discuté dans la section précédente mais le but est de réduire le nombre des règles et de ne pas les regrouper.

L'algorithme One-Associations-Meta-Rules Dans la section 2.4.2, on a parlé sur le principe de l'algorithme One-Associations-Meta-Rules qui permet d'extraire des meta-règles contenant une seule règle dans la partie antécédente et une autre dans la partie conséquente. Maintenant, on va voir comment Berrado

et d'autres utilisent ces meta-règles afin de réduire l'espace de règles.

Définition En se basant sur l'ensemble des meta-règles MR, pour chaque règle $r_i \in$ R, on peut définir deux sous ensembles OUT(r_i) et IN(r_i) tel que :
OUT(r_i)=$\{r_j \in \text{R}/r_i \Rightarrow r_j \in \text{MR}\}$
IN(r_i)=$\{r_j \in \text{R}/r_j \Rightarrow r_i \in \text{MR}$
r_i et r_j sont dites équivalentes si et seulement si
$r_i \in$ OUT(r_j) et $r_i \in$ IN(r_j).
OUT(r_i)$\setminus \{r_j\}$=OUT(r_j)$\setminus \{r_i\}$.
et
IN(r_i)$\setminus \{r_j\}$=IN(r_j)$\setminus \{r_i\}$.
Maintenant, si les deux règles r_i et r_j sont équivalentes et l'antécédente de r_i est incluse dans l'antécédente de r_j alors on peut garder r_i et supprimer r_j.
Exemple Prenons l'exemple de la section 2.4.2, on aura :
OUT(r_1)=$\{r_2\}$.
OUT(r_2)=$\{r_1\}$.
IN(r_1)=$\{r_2, r_3\}$.
OUT(r_2)=$\{r_2, r_3\}$.
r_1 et r_2 sont équivalentes car :
$r_2 \in$ OUT(r_1).
$r_1 \in$ OUT(r_2).
$r_2 \in$ IN(r_1).
$r_1 \in$ IN(r_2).
OUT(r_1)$\setminus \{r_2\}$=OUT(r_2)$\setminus \{r_1\}$=$\emptyset$.
IN(r_1)$\setminus \{r_2\}$=IN(r_2)$\setminus \{r_1\}$=$\{r_3\}$.
Et l'antécédente r_1 est égale à {A} et incluse dans l'antécédente de r_2 qui est égale à {A, C}.
Par conséquent, on garde r_1 et on supprime r_2.

Dans [77], une extension de l'algorithme One-Associations-Meta-Rules est developée. Premièrement, la connexion de Galois est établie entre les règles afin de générer une structure appelée (OR pour Object Rules). Chaque item dans OR est une règle et chaque transaction est un item (dans la première base de tran-

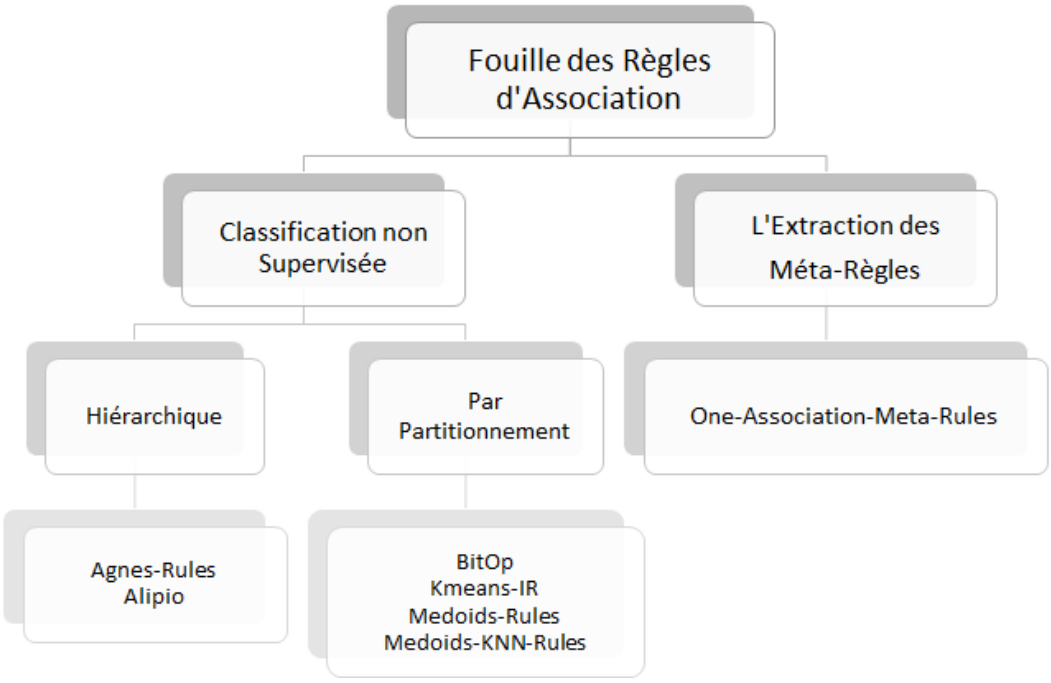


FIGURE 2.7 – *classification des algorithmes de fouille des règles*

sactions) qui couvre un certain nombre de règles. Aprés il génère les meta-règles tout en appliquant l'algorithme Closest. Finalement, il utilise les règles et les meta-règles afin de garder juste les règles pertinentes.

Il est claire que lorsqu'on utilise l'algorithme Closest à la place de l'algorithme APRIORI, un nombre réduit de meta-règles est obtenu. Cependant, l'exploitation des meta-règles est basée seulement sur la propriété de closure qui diminue proportionnelement la qualité des règles résultantes.

2.6 Analyse et Classification

Dans cette section, on va faire une étude théorique des algorithmes déja présentés en proposant une classification de ces algorithmes.

2.6.1 Analyse et classification des algorithmes de fouille de règles d'association

Les algorithmes de fouille des règles peuvent être divisés en deux catégories selon la tâche désirée : la classification non supervisée et l'extraction des meta-règles.

Pour la classification non supervisée, on a vu deux types d'algorithmes selon la façon et la manière de regroupement des règles d'association (Hiérarchique et par partitionnement).

Deux algorithmes de classification non supervisée hiérarchique des règles d'association sont détaillés (Agnes-Rules et l'algorithme d'Alipio). Les deux algorithmes suivent le même processus de regroupement des règles tout en appliquant l'algorithme HAC tandis que la différence entre ces deux algorithmes est dans le calcul de distance entre deux règles d'association. En effet, Dans l'algorithme Agnes-Rules, on fait l'intersection et l'union entre l'ensemble de transactions qui vérifient les deux règles. Au pire des cas la complexité de cette distance est vérifie lorsque les deux règles ont un support de 100% c'est à dire qu'elles supportent toute la base de transactions. La complexité de cette mesure au pire des cas est donc $O(2 \times m)+$. La complexité au pire des cas de la deuxième distance est vérifie lorsque les deux règles contiennent tous les items. Pour celà, la complexité de la distance d'Alipio sera $O(2 \times n)$. $O(n)$ pour l'opération de l'intersection des items et un autre $O(n)$ pour l'opération de l'union. La complexité des deux algorithmes au pire des cas est la complexité de HAC en remplaçant le calcul de similarité des données par les règles d'association. En effet, la complexité au pire des cas de l'algorithme Agnes-Rules est $O(2 \times m^2 \times \log NBR)$ et la complexité est au pire des cas de l'algorithme d'Alipio est $O(2 \times m^2 \times \log NBR)$. On peut conclure que la complexité de Agnes-Rules augmente tout en augmentant le nombre de transactions et la complexité de l'algorithme d'Alipio augmente tout en ajoutant des items.

Quatre d'autres algorithmes de classification non supervisée par partitionement des règles d'association sont étudiés. On commence par analyser l'algorithme BitOp, En découpant l'algorithme BitOp en plusieurs parties, la complexité de l'algorithme BitOp au pire des cas peut être calculée comme suit :

- De la ligne 3 à la ligne 17 on répete le processus suivant pour toutes les règles : On copie tous les items de la i^{emme} règle dans la i^{emme} ligne de la matrice *masque*. Puis, on fait une opération *et* du coût de $O(n)$ entre une ligne d'une matrice *masque* et toutes les règles. A la fin, on fait une opération de $O(n)$ pour calculer les nouveaux clusters trouvés, la complexité de cette étape est $O(NBR \times (n + (NBR \times n) + n)$.
- De la ligne 18 à la ligne 24 on supprime les clusters qui sont inclus dans d'autres clusters, la complexité de cette étape est $O(NBR^2)$.

On additionne les complexités des deux étapes de l'algorithme. Aprés simplification, la complexité de l'algorithme BitOp est : $O(NBR^2 \times n)$.

Les trois algorithmes de classification non supervisée par partitionnement suivent le même raisonement. Ils diffèrent lors de calcul de similarité entre deux règles et la détermination du centre de gravité d'un ensemble de règles. Le calcul de similarité pour les deux algorithmes Kmeans-IR et KNN-Medoids-Rules est $O(2 \times n)$, alors que dans Medoids-Rules la similarité entre deux règles est $O(2 \times m)$. La détermination de centre de gravité est du coût de $O(NBR)$ pour les deux algoirthmes (Medoids-Rules et KNN-Medoids-Rules) alors que pour l'algorithme Kmeans-IR est $O(NBR + n)$. La complexité des trois algorithmes au pire des cas est :

- Kmeans-IR : $O(K \times NBR \times Iter \times (3 \times n + NBR))$ tel que $Iter$ est le nombre d'itérations de l'algorithme Kmeans-IR.
- Medoids-Rules : $O(K \times NBR \times Iter \times (2 \times m + NBR))$ tel que $Iter$ est le nombre d'itérations de l'algorithme Medoids-Rules.
- KNN-Medoids-Rules : $O(K \times p \times Iter \times (2 \times m + p) + (NBR - p))$ tel que p est le nombre des règles séléctionées pour le premier regroupement tout en appliquant l'algorithme Medoids-Rules.

Pour l'extraction de meta-règles d'association, on a vu un seul algorithme One-Association-Meta-Rules. Il consiste en deux étapes : La transformation de base de règles à la base de transactions, et la recherche des meta-règles fréquentes. La complexité de la transformation est $O(NBR \times n \times m)$ et la complexité de la recherche des meta-règles fréquentes est $O(NBR^2)$. Par conséquent, la complexité

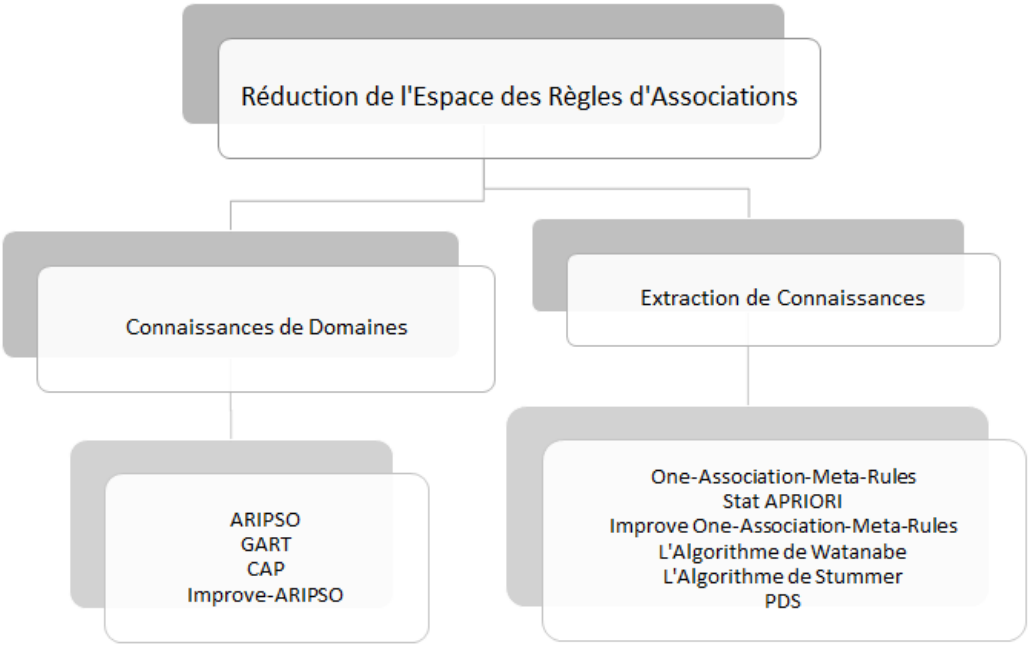


FIGURE 2.8 – *classification des algorithmes de réduction des règles d'associations*

de l'algorithme One-Associations-Meta-Rules est $O((NBR \times n \times m) + (NBR^2))$.

De plus, dans la Figure 2.7, on propose une classification des différents algorithmes de fouille de règles d'associations selon la tâche effectuée.

2.6.2 Analyse et classification des algorithmes de reduction des règles d'association

On a vu deux catégories d'algorithmes de réduction des règles d'association, la première catégorie introduit les connaissances de l'expert ou l'utilisateur au processus de l'analyse des règles obtenues. Cependant, cette démarche ne peut être appliquée lorsqu'il s'agit des domaines d'applications comme la recherche d'informations où l'utilisateur ne peut y intervenir. De plus, le coût de construction des ontologies est vraiment important tout en augmentant le nombre d'items.
La deuxième catégorie des algorihmes de réduction des règles essait de trouver et de calculer les différentes dépendances entre les règles, ces dépendances

aident à éliminer les règles redondantes, inutiles et déductives. Ces algorithmes utilisent soit des approches basées sur des testes statistiques comme le test de Khi-deux, soit carément ils appliquentt un autre processus de fouille comme on a vu dans l'algorithme One-Association-Meta-Rules. Cette classe d'algorithmes résout le problème de la première catégorie tout en évitant l'interaction avec l'utilisateur mais ils restent inefficaces car le temps de calcul est assez important.

De plus, dans la Figure 2.8, on propose une classification des différents algorithmes de réduction de règles d'association.

2.7 Conclusion

Ce chapitre a été divisé en deux parties : La première partie à été consacrée aux algorithmes de fouille de règles d'association, on a présenté deux tâches de fouille des règles, la classification non supervisée des règles d'association, et l'extraction des meta-règles à partir de l'ensemble de règles obtenues. Une étude théorique et une classification de ces algorithmes ont été établies. Dans la deuxième partie, on a détailé les algoritmes de réduction des règles d'association. Ils sont regroupés en deux classes, il y'a des algorithmes qui utilisent des informations supplémentaires introduites par l'utilisateur. D'autres algorithmes utilisent les différentes relations entre les règles pour distinguer les règles pertinentes et les règles redondantes. Lorsque le nombre de règles devient important, tous ces algorithmes (Les algorithmes de fouille et de réduction des règles) restent inefficaces. Pour celà, une deuxième contribution de cette thèse est d'améliorer les algorithmes de fouille et de réduction des règles d'association.

Chapitre 3

Les essaims d'abeilles

3.1 Introduction

Le mot essaim est utilisé pour un groupe d'animaux qui réalisent un comportement collectif tels que les oiseaux, les poissons, et les insectes comme : les fourmies, les termites et les abeilles. Chaque agent dans l'ensemble des essaims possède un état local de l'environment, l'interaction et la communication entre les différents agents de la colonie produisent une certaine intelligence appellée l'intelligence par essaims.

Les esssaims utilisent leur environement et les ressources qui disposent afin de produire une intelligence collective. L'auto-organisation est la charactéristique principale de la colonie des essaims. Bonabeau et d'autres. [78] interprètent l'auto-organisation des essaims à partir de trois charactéristiques : (a) la réaction positive ou bien négative qui représente un simple comportement d'un seul agent lorsqu'il y'a un changement positif ou bien négatif de son état local. (b) L'aléatoire qui est souvent nécessaire car il permet de découvrir de nouvelles solutions. (c) Les interactions multiples des différents agents de la colonie des essaims permettent une diffusion rapide de l'information trouvée ou bien produite par un simple agent de la colonie.

Dans ce chapitre, on va voir les differentes approches basées sur l'intelligence par essaims, en particulier, les essaims d'abeilles. On analyse le comportement des abeilles dans la nature. Ensuite, on étudie les différentes implémentations des

abeilles artificielles proposées dans la litérature.

3.2 L'intelligence par essaims

L'intelligence par essaims est un domaine émergent dans l'optimisation et les chercheurs développent plusieurs algorithmes, en simulant le comportement des différents essaims comme : les insectes, les oiseaux, les abeilles, les poissons et les termites.

Dans les années 90, les algorithmes de Particle Swarm Optimization (PSO), basés sur le comportement des oiseaux, et ceux d'Ant Colony Optimization (ACO), basés sur le comportement des colonies de fourmies, sont introduits. Ils ont été appliqués dans plusieurs problèmes d'optimisation. Par la suite, le comportement des abeilles a été trés utilisé dans les années 2000. Dans cette section, on présentera les deux algorithmes ACO et PSO. Pour les deux prochaines sections, on détaillera sur les essaims d'abeilles dans la nature et leurs implémentations comme des approches d'optimisation combinatoire.

3.2.1 L'algorithme des colonies de fourmies

L'algorithme de colonies de fourmies (ACO) provient du comportement des fourmies, initialement proposé par Marco Dorigo et d'autres [79] pour le problème du voyageur de commerce. Il sinspire du comportement des fourmies lors de la recherche de la source de nourriture. En effet, la collectivité produite par la communication entre les fourmies de la colonie, permet de trouver le plus court chemin entre une source de nourriture et le nid.

La communication entre les fourmies se fait en déposant une quantité de phéromones sur le chemin de chaque fourmie. Ensuite, la colonie suit le chemin de la fourmie qui a plus de phéromones.

Ce système porte le nom de « stigmergie », et se trouve chez plusieurs animaux sociaux tels que les oiseaux, les abeilles et les termites.

L'algorithme ACO

```
Debut
 Initialisation des pistes de phéromone
    Tanque critère d'arrêt non atteint faire
       construire les solutions composant par composant
       mise à jour des pistes de phéromone
    fin_tanque
Fin
```

3.2.2 L'algorithme des essaims de particules

Dans PSO [80], une population de particules commence à se déplacer dans l'espace de recherche vers la meilleure particule courante afin de trouver la particule optimale. Les diifférentes positions des particules sont considérées comme des solutions potentielles. Dans chaque itération, chaque particule calcule sa nouvelle position en fonction de la qualité de la position précédente, de la vitesse de déplacement et de la qualité de la position de la meilleure particule générique. Ce processus est répété jusqu'à ce que le nombre maximal de déplacements soit atteint ; ou bien la position optimale soit trouvée, par l'une des particules.

L'algorithme PSO

```
Debut
 affecter des positions initiales à l'ensemble de particules
    Tanque critère d'arrêt non atteint faire
       calculer la fitness de chaque particule
       choisir la meilleure particule
       calculer la vitesse de chaque particule
       mise à jour la position de chaque particule
    fin_tanque
Fin
```

3.3 Les abeilles naturelles

Les essaims d'abeilles sont considérés comme les plus importants essaims dans la nature. Ces essaims allouent dynamiquement des tâches et s'adaptent, d'une manière collective et intelligente, aux différents changements de l'environement.

Les abeilles ont une mémoire photographique, des antennes et des systèmes de navigation, qui permettent, d'une part, de détecter de nouvelles ruches, et d'une autre part, d'accomplir leurs différentes tâches biologiques telles que : la recherche de nourriture et la reporduction.

Plusieurs chercheurs se sont inspirés du comportement collectif des abeilles naturelles afin de créer des modèles résolvant des problèmes d'optimisation combinatoire.

Avant de présenter les différents algorithmes basés sur les essaims des abeilles, nous allons voir, dans ce qui suit, les différents types d'abeilles tels que : la reine, les ouvrières et les mâles, ainsi que le comportement des abeilles afin de réaliser collectivement les activités biologiques comme la sélétion du nid, la nourriture, et le mariage).

3.3.1 Types d'abeilles

- La reine : Elle est la seule femelle fertile de la communauté, Elle est donc la mère de tous les individus composant cette communauté (les faux bourdons, les ouvrières et les futures reines). Sa capacité à pondre est très importante, sa production journalière dépasse souvent 1500 oeufs. La reine n'a pas les appendices des ouvrières comme les paniers à pollen, les glandes secrétrices de la cire et le sac à miel bien développé. Sa nourriture quasi exclusive est une secrétion, appelée gelée royale, produite par les glandes situées dans la tête des ouvrières. La durée de vie moyenne de la reine est de un à trois ans.
- L'ouvrière : Les ouvrières sont toujours beaucoup plus nombreuses que les mâles. Dans la ruche d'une région tempérée, le nombre d'ouvrières est compris entre 8000 et 15000 individus au printemps, mais peut dépasser 80000 au début de l'été. Les ouvrières sont incapables de s'accoupler et donc de se reproduire. Elles secrètent la cire, construisent les alvéoles, récoltent le

nectar, le pollen et l'eau, transforment le nectar en miel, nettoient la ruche et la défendent contre les prédateurs. L'espérance de vie des ouvrières est d'environ six semaines. Pendant les trois premières semaines de leur vie, les ouvrières s'emploient, entre autres, à construire les rayons, nettoyer et polir les alvéoles, nourrir les jeunes et la reine, contrôler la température, évaporer l'eau du nectar jusqu'à ce qu'il prenne une consistance de miel épais. Après cette période, elles vont récolter le nectar sur les fleurs et défendent la ruche.

- Le mâle : Il ne travaille pas, il est incapable de se nourrir lui-même et ne peut même pas défendre la colonie, puisqu'il n'a pas de dard, le nombre de mâle est entre 300 et 3000 dans la colonie, sa seule fonction est de s'accoupler avec la reine, quand celle-ci quitte la ruche, les mâles, attirés par son odeur. Après l'accouplement, le mâle meurt de faim rapidement ; la durée moyenne de sa vie est de 90 jours.

3.3.2 Le comportement des abeille dans la nature

- La séléction de la location du nid : Les sociétés d'abeilles vivent dans des nids appelés ruches. Dans le dernier mois du printemps ou au début de l'été, la reine s'envole accompagnée des milliers d'ouvrières, pour aller construire une autre colonie, elles se divisent en deux groupes : la reine avec la moitié des ouvrières et les filles de la reine avec le reste des ouvrières. Pour la séléction du nid, certaines abeilles explorent les sites ; les autres abeilles restent inactives, probablement conservent l'énergie d'essaim, jusqu'à prendre une décision et migrer vers le nouveau nid. Les abeilles responsables de la recherche de nourriture indiquent les différents nids par plusieurs danses qui s'appellent la "danse remuer". La direction est indiquée par un angle à partir du soleil, la distance est définie par la durée d'une partie remue de la danse, la remue tourne 40° à droite pour indiquer la source de nourriture. A travers le chemin figuré par la danse, l'abeille tourne verticalement autour du centre et forme un angle avec son corps par une vibration. Cet angle relie la ruche avec le nectar.
- La nourriture : Le rôle principal de l'abeille qui cherche la nourriture est

de trouver une source très riche de nourriture pour obtenir une quantité maximale de nectar. Et pour celà, il y a de différents types de ces abeilles : Abeille en chômage : il y a des abeilles n'ayant aucune connaissance sur la source de nourriture et aucune vue de source de nourriture à exploiter. Donc, ces abeilles restent dans la ruche et attendent la danse excercée par les autres abeilles pour établir la source de nourriture.
Abeille employé : est associée avec une source de nourriture particulière. Elle a une connaissance suffisante pour connaître la source de nourriture. Elle trouve et exploite la source, mémorise la location et charge une partie de nectar et la décharge dans la ruche.

- Le mariage : L'accouplement se fait en plein ciel entre avril et juin, et une semaine après son éclosion, la reine rentre dans la période idéale pour son accouplement. Elle réduit son poids en passant d'un régime de gelée royale à un régime de miel et se prépare à l'envol nuptial. Un jour ensoleillé, elle se glisse lentement hors de la ruche et prend son envol. La rencontre se fera quelque part dans le ciel, entre la reine et des mâles de la ruche mais aussi des mâles d'autres ruches. Ce qui est étrange c'est que même pendant la période de reproduction, une rencontre entre la reine et les mâles, dans la ruche ne produira aucune suite. C'est seulement à la sortie de la ruche que la reine se rend disponible à l'accouplement.

3.4 Les abeilles artificielles

Il existe plusieurs algorithmes qui simulent le comportement des abeilles naturelles, ce qui suit on détaillera les algorithmes les plus utilisés dans la litérature [81].

- **L'algorithme ABC [82]** Dans l'algorithme ABC, les sources de nouriture représentent l'espace de solutions d'un problème donné et nectar de chaque source représente la qualité d'une seule solution. On distingue trois types d'abeilles (les abeilles travailleuses, spéctatrices et maîtresses). Le nombre des abeilles travailleuses ou bien spéctatrices est égal à la taille de population d'une seule itération. Dans un premier temps, ABC génère aléatoirement la population initiale qui contient SN solutions tel que SN

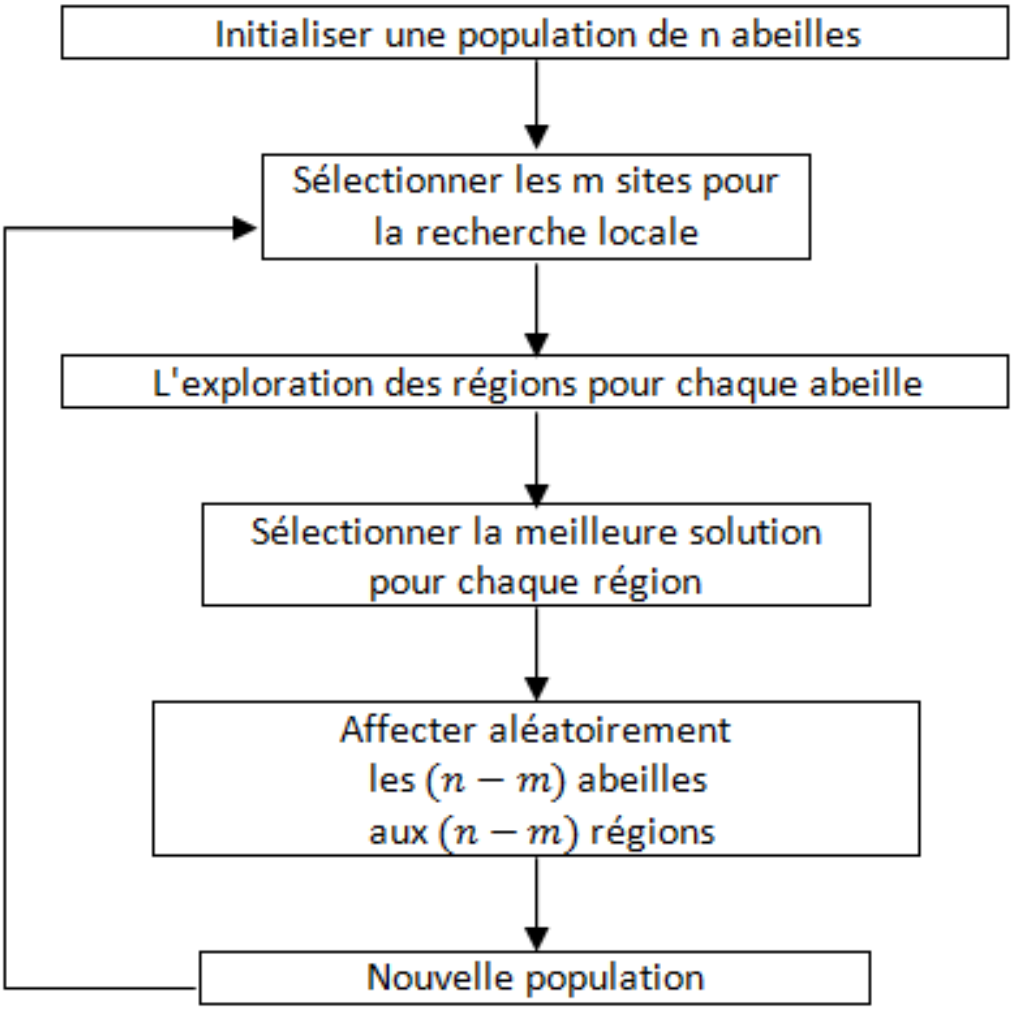


FIGURE 3.1 – *L'algorithme général de BA*

est le nombre des abeilles travailleuses ou bien spéctatrices. Aprés chaque abeille travailleuse établit des modifications sur la solution courante en fonction de la qualité de la solution et l'information locale de sa mémoire. Si la qualité de la nouvelle solution est meilleure que la précédente alors cet abeille remplacera la nouvelle solution à la place de la précédente. Aprés, les abeilles travailleuses partagent leurs solutions avec les abeilles spéctatrices. Par la suite, chaque abeille spéctatrice choisit la solution la plus proche de la solution courante. Comme le cas des abeilles travailleuses chaque abeille spéctatrice établit des modification sur la solution courante et évalue la solution trouvée tout en gardant la meileure solution parmi les deux solutions. A la fin, les abeilles maîtresses explorent d'autres régions de recherche en sauvegardant à chaque fois les meilleures solutions produites par les abeilles travailleuses et spéctatrices. Ce processus doit être répété pour un nombre d'itérations maximum.

L'algorithme ABC

```
Debut
 Initialiser la population initiale
Tanque critère d'arrêt non atteint faire
 placer les abeilles travailleuses sur leurs sources de nouriture
 placer les abeilles spetatrices sur les solutions déjà trouvées par
 les abeilles travailleuses
 sauvegarder la meilleure solution trouvée par les abeilles
 travailleuses et spectatrices
 envoyer les abeilles maitresses dans l'espace de solutions afin de
 trouver d'autres sources de nouriture
fin_tanque
Fin
```

– **L'algorithme BCO [83]** Initialement, tous les abeilles sont situées dans la ruche. Chaque abeille réalise une série de mouvement ou bien de déplacement, afin de construire une solution pas à pas. En effet, Les abeilles ajoutent incrémentalement des composants à la solution courante jusuq'à ce qu'elles obtiennent une ou plusieurs solutions admissibles. Durant l'ex-

ploitation de l'espace de solutions, les abeilles effectuent deux étapes :
l'étape vers l'avant : dans cette étape les abeilles créent plusieurs solutions partielles en se basant sur l'expérience collective accomplie dans les itérations précédentes.
l'étape vers l'arriére : Aprés la première étape, les abeilles retournent à la ruche. Dans la ruche, toutes les abeilles communiquent entre elles afin de participer à la construction des solutions complètes de chaque itération. En effet, chaque abeille partage avec les autres abeilles, la solution partielle déja créée ainsi de sa qualité. Les solutions complètes d'une seule itération sont conçues tout en sauvegardant la meilleure solution. Ce processus doit être répété jusqu'à ce qu'on obtienne un nombre maximum d'itérations.

L'algorithme BCO

```
Debut
Initialiser la population initiale
Tanque critère d'arrêt non atteint faire
 chaque abeille se déplace dans l'espacce de recherche en
 construisant une solution partialle pas à pas
 chaque abeille envoie à toutes les abeilles sa solution
 partielle ainsi que de sa qualité
 construire les solutions complétes de cette itération
 sauvegarder la meilleure solution de cette itération
fin_tanque
Fin
```

- **L'algorithme BA [84]** Initialement, on génère une population de n abeilles, ces abeilles sont divisées en deux ensembles. Les premières m abeilles tel que $m < n$, explorent m différentes régions de l'espace de recherche, chaque abeille explore sa région indépendament et retourne la meilleure solution de sa région. On sauvegarde à chaque itération les meilleures solutions de chaque région de telle sorte que les meilleures régions seront probablement explorées dans les prochaines itérations. Les $n - m$ abeilles restantes établissent une recherche aléatoire afin de reconstruire une nouvelle popula-

tion pour la prochaine itération. Ce processus doit être répété jusqu'à un nombre maximum d'itérations. La Figure 3.1 explique clairement les différentes étapes de cet algorithme.

– **L'algorithme MBO [85]** Il est basé sur le comportement d'accouplement des abeilles, la reine est l'individu principale de cet évenement, elle se déplace selon une certaine énergie et une certaine rapidité, elle choisi les mâles, et elle séléctionne les meilleurs mâles. Si le mâle est séléctioné par la reine, la production est réalisée tout en ajoutant son spèrme dans l'ovaine de la reine. En utilisant la mutaition et le croisement, la production des larves est effectuée, par la suite, chaque ouvriére occupe d'une seule larve. La meilleure larve sera la nouvelle reine de la prochaine itération. L'algorithme **??** décrit la structure de cet algorithme.

L'algorithme MBO

```
Debut initialisation aléatoire de la reine améliorer la reine par
les ouvrières en faisant une recherche local à partir de cette
reine
Tanque le nombre d'accouplement n'est pas atteint faire
initialiser énergie et rapidité de la reine
 Tanque énergie de la reine > 0 faire
  La reine se déplace et choisie les mâles
   Si le male est selectionné alors
    ajouter son spèrme dans l'ovaire de la reine
    modifier l'énergie et la rapidité de la reine
   fin_si
 fin_tanque
 produire les larves avec les mutations et les croisements
 utiliser les ouvrieres pour l'occupation des larves en
 faisant une recherche locale à partir de chaque larve
 modifier la fitness des ouvrières
 Si la meilleure larve adéquate la reine alors
 remplacer les chromosomes de la reine avec les chromosomes
```

```
 de la meilleure larve
 fin_si
 tuer toutes les larves
fin_tanque
Fin
```

– **L'algorithme BSO [86]** La métaheuristique BSO proposée dans [86] simule le comportement collective des abeilles dans la recherche de la nourritue. Dans un premier temps, une abeille appellée InitBee essaye de trouver une solution appelée Sref qui représente de bonnes caratéristiques. A partir de cette solution, on détermine un ensemble de points de l'espace de recherche dont lesquelles les abeilles réalisent une recherche aprofondie afin de trouver d'autres solutions plus efficaces que Sref. Cet ensemble de solutions est appelé l'espace des régions, il est calculé de telle sorte que les différents points sont trés éloignés entre eux. Chaque abeille considère un point de l'espace des régions où elle effectue une recherche locale à partir de ce point. Aprés la terminaison d'exploitation des régions pour chaque abeille, les abeilles se communiquent entre elles dans le but de trouver la meilleure solution de toute la colonie, cette opération est effectuée grâçe à la table Dance. Par la suite, la meilleure solution est considérée comme la solution de référence Sref pour la prochaine itération. Afin d'éviter des cycles, d'autres terme éviter que les abeilles retournent à une solution déja exploitée, dans chaque itération, la solution référence est sauvegardée dans une structure appelée taboo list. Cependant, si aprés un nombre d'itérations donné, les abeilles observent qu'il y'a aucune amélioration, elles introduisent la qualité de diversification. Le critère de diversification consiste de séléctionner à partir de taboo list, la solution la plus éloignée que la solution courante. l'algorithme termine quand la solution optimale est trouvée ou bien un nombre maximal d'itérations est atteint.

L'algorithme général de BSO

```
Debut
Sref=la solution trouvée par InitBee
```

```
  Tanque le critére d'arrêt non atteint faire
   insérer Sref dans taboo list
   espace-regions(Sref)
   affecter une solution de l'espace des régions pour
   chaque abeille
   Pour chaque abeille a faire
    recherche-locale(a)
    sauvegarder le résultat dans la table Dance
   fin_pour
   choisir la nouvelle solution référence Sref
  fin_tanque
 Fin
```

- **L'algorithme BeeHive [87]** Dans la nature, on distingue deux types d'abeilles, les abeilles à petite distance, il s'agit des abeilles qui cherchent leurs nourriture prés de la ruche. D'autres abeilles appelées abeilles à longue distance, ces abeilles explorent d'autres régions loin de la ruche. L'algorithme BeeHive s'est inspiré de cette idée. En effet, l'ensemble des abeilles est divisé en deux sous ensemble. Le premier sous ensemble applique une recherche locale prés de la ruche, tandis que le deuxième applique une opération de diversification en explorant d'autres zones de recherche. A la fin de chaque itération on sauvegarde les meilleures solutions trouvées par les deux types d'abeilles.

L'algorithme BeeHive

```
Debut
 Diviser la population initiale en deux sous ensembles
 (abeilles à petite distance et abeilles à longue distance)
 Tanque le critére d'arrêt non atteint faire
  chaque abeille à petit distance applique une recherche locale
  à sa solution
  détermine la meilleure solution des abeilles à petite distance
```

```
  chaque abeille à longue distance explore sa région localement
  détermine la meilleure solution des abeilles à longue distance
  Sauvegarder la meilleure solution de ces sous ensembles
 fin_tanque
Fin
```

3.5 Conclusion

Dans ce chapitre, on a vu les différents comportement des essaims des abeilles dans la nature. On a vu aussi les différents algorithmes qui simulent le comportement des abeilles pour la résolution des problèmes complèxes. Ces algorithmes ont été appliqués dans plusieurs domaines comme suit : la fouille de données, la recherche d'infomrations, les systèmes multi-agents, le traitement d'images et la vérification des contraintes.

Parmi ces algorithmes, on a l'algorithme BSO développé par notre laboratoire, c'est une métaheuristique efficace, simple à implémenter et gére peu de pramètres en comparant avec les autres algorithmes des essaims d'abeilles. De plus, elle est montrée son efficacité dans plusieurs domaines tel que : la fouille de données, le problème SAT et MAX-W-SAT, la recherche d'informations..ect. Par conséquent, Cette métaheuristique BSO est utilisée dans notre contribution (voir les chapitres 4 et 5).

Chapitre 4

Amélioration des algorithmes d'extraction des règles d'association

4.1 Introduction

Dans le premier chapitre de cette thèse, on a détaillé les algorithmes d'extraction des règles d'association. Il y'a deux catégories d'algorithmes d'extraction des règles d'association, les algorithmes exacts qui trouvent toutes les règles pertinentes mais avec un temps considérable. D'autres catégories d'algorithmes d'extraction des règles d'association sont basés sur les métaheuristiques, le but est de réduire le temps d'exécution des algorithmes exacts, cependant, avec ces algorithmes la qualité des solutions diminue.

Pour celà, la première contribution de cette thèse consiste à améliorer les algorithmes d'extraction des règles d'association.

Dans un premier temps, nous proposons deux approches séquentielles d'extraction des règles (BSO-ARM, HBSO-ARM) qui sont basés sur les essaims d'abeilles et plus particulièrment sur la métaheuristique BSO qui a été bien expliquée dans le troisième chapitre.

Cependant, avec l'apparition du web, les données sont de plus en plus accroîtes, et à cause de cette grande quantité de données, les algorithmes basés sur métaheuristique se bloquent. A titre d'exemple, en fouillant la base de tran-

sactions WebDocs qui contient plus d'un million de transactions, nos approches séquentielles se bloquent, pour celà, une parallélisation de l'algorithme BSO-ARM sera proposée afin de traiter la base de transactions WebDocs. La version parallèle de BSO-ARM se fait en utilisant la platforme GPU qui est trés pratique pour un problème d'optimisation combinatoire.

4.2 Les essaims d'abeilles pour les règles d'association

Dans cette première partie, on propose trois algorithmes d'extraction des règles d'association, le premier appelé BSO-ARM, basé sur le principe de la métaheuristique des essaims d'abeilles BSO, le deuxième algorithme est une amélioration de l'algorithme BSO-ARM, en développant d'autres stratégies pour la détermination des régions, alors que dans le troisième algorithme, une hybridation entre le deuxième algorithme et la métaheuristique (la recherche tabou) est établie dans le but d'améliorer l'opération de calcul du voisinage.

4.2.1 L'algorithme BSO-ARM

Dans [107], nous avons proposé l'algorithme BSO-ARM pour l'extraction des règles d'association qui est basé sur la métaheuristique BSO déja expliquée en troisième chapitre.

Les opérations de base de l'algorithme BSO-ARM

Afin d'adapter BSO aux règles d'association, on doit définir les opérations suivantes : La représentation des solutions, la stratégie de détermination des régions, la fonction d'évaluation d'une solution, et la stratégie de calcul du voisinage.

- **La représentation des solutions :** Dans la littérature des algorithmes d'extraction des règles d'association[20] [23], on distingue deux types de représentation des solutions, appelées réspectivement, la représentation binaire et la représentation entiére. Dans la représentation binaire, chaque solution (règle) est représentée par un vecteur S de n éléments tel que n

est le nombre d'items. De plus, $S[i] = 1$ si l'item i est une règle, 0, sinon. Alors que dans la représentation entiére, la solution (la règle) est représentée par un vecteur S de $k+1$ éléments tel que k est la taille de la règle. Le premier élément est un séparateur entre la partie antécédente et la partie conséquente de la solution. Pour chaque autre élément i dans S, si $S[i] = j$ alors l'item j apparaît dans la i^{me} position de la règle. Dans BSO-ARM, on combine les deux représentations afin de faciliter l'implémentation des deux opérations de BSO qui sont la détermination des régions et le calcul de voisinage. En effet, chaque solution S est un vecteur de n éléments tel que :

$S[1] = indice$ tel que indice est un séparateur ente la partie antécédente et la partie conséquente.

$S[i] = j$ tel que $j > 0$ si l'item j apparaît dans la i^{me} position de S.

$S[i] = 0$ s'il n'aura pas un item dans la i^{me} position de la solution S.

Exemple Considérons T=$\{t_1,t_2,...,t_{10}\}$ est un ensemble d'items :

$S1 = \{3, 2, 4, 5, 3, 0, 0, 0, 6, 7, 8\}$ représente la règle $r_1 : t_2, t_4, t_5 \Rightarrow t_3, t_6, t_7, t_8$.

$S2 = \{2, 0, 0, 5, 3, 0, 0, 0, 1, 2, 9\}$ représente la règle $r_2 : \xi \Rightarrow t_5, t_3, t_1, t_2, t_9$.

$S3 = \{5, 1, 6, 8, 7, 0, 0, 0, 3, 2, 4\}$ représente la règle $r_3 : t_1, t_6, t_8, t_7 \Rightarrow t_3, t_2, t_4$.

- **L'évaluation d'une solution :** Considérons α et β deux paramètres empiriques, une solution S peut être évaluée comme suit :
  F_{max} (S) = $\alpha\times$ confiance(S)+$\beta\times$ support(S)
  si Confiance(S) > MinConf et Support(S) > Minsup
  F_{max} (S) = -1 sinon.
- **Le calcul de voisinage :** Le calcul de voisinage est déterminé à partir de chaque solution S en ajoutant 1 ou bien en soustrayant 1 à chaque bit de la solution S.

  Exemple Considérons la solution suivante : $S = \{3, 2, 4, 5, 3, 0, 0, 0, 6, 7, 9\}$
  En ajoutant la valeur 1 du premier bit de la solution S, on aura $S1 = \{4, 2, 4, 5, 3, 0, 0, 0, 6, 7, 9\}$
  En soustrayant la valeur 1 du premier bit de la solution S, on aura : $S2 = \{2, 2, 4, 5, 3, 0, 0, 0, 6, 7, 9\}$
  En ajoutant la valeur 1 du second bit de la solution S, on obtient :$S3 =$

$\{3, 3, 4, 5, 3, 0, 0, 0, 6, 7, 9\}$
En soustrayant la valeur 1 du second bit de l solution S, on obtient : $S4 = \{3, 1, 4, 5, 3, 0, 0, 0, 6, 7, 9\}$

- **La détermination des régions :** On assume Sref comme la solution référence trouvée par l'abeille intiale (Initbee). La détermination des régions de cet algorithme est comme suit :
  Premièrement, une variable de type entier appelée Flip est choisie dans l'intervale $[1 - n]$. Deuxièment, les solutions sont générées tout en ajoutant ou bien soustrayant la valeur Flip à un bit de la solution $Sref$. Ce processus se répétera pour tous les autres bits de Sref. **Exemple** Considérons $Flip = 2$ et Sref= $\{3, 2, 4, 5, 3, 0, 0, 0, 6, 7, 9\}$.
  En ajoutant la valeur Flip du premier bit de la solution Sref, on aura : S1= $\{5, 2, 4, 5, 3, 0, 0, 0, 6, 7, 9\}$
  En soustrayant la valeur Flip du deuxième bit de Sref, on obtient :
  $S2 = \{1, 2, 4, 5, 3, 0, 0, 0, 6, 7, 9\}$

4.2.2 L'algorithme général de BSO-ARM

L'initialisation de la solution référence et le degré de diversification sont choisis comme dans [86].

L'algorithme général de BSO-ARM

```
Debut
 Les paramètres empiriques: k (nombre d'abeilles), flip, MaxIter
 Entrées: un ensemble de transactions
 Sortie: ensemble de règles d'association
   i=0
   Sref=la solution initiale générées alétoirement
   Fitness(Sref)
   S=Sref
```

```
    Tanque i < MaxIter faire
      TabouList=TabouList + Sref
      solutions=Determination-régions(Sref,Flip)
      K-solutions=selectionner-K-solution(solutions)
        Pour chaque solution s dans K-solutions faire
          Fitness(s)
          affécter une solution de K-solutions à chaque abeille
             Pour chaque abeille a faire
               Dance[a]=meilleur-voisin(a)
             end_pour
          Sref=la meilleure solution de la table Dance
        end_pour
        Si (Fitness(Sref)> Fitness(S)) alors
           S=Sref
        fin_si
        i=i+1
    fin_tanque
    Pour chaque solution s dans la table Dance faire
      générer la règle associée à partir de la solution s
    fin_pour
Fin
```

4.2.3 L'algorithme BSO-ARM avec plusieurs stratégies

Le précédent algorithme BSO-ARM a principalement deux limites : premièrement, la représention de solutions est inefficace, en effet, tout en augmentant le nombre des items, une solution peut prendre un espace mémoire important ce qui réduit la performance de l'algorithme. Deuxièment, la stratégie utilisée dans la détermination des régions doit être améliorée afin de bien diviser l'espace de recherche à toute la colonie des abeilles.

Pour celà, dans [108] nous avons développé une extension de l'algorithme BSO-ARM tout en proposant une autre représentation de solutions et d'autres stratégies pour la détermination des régions.

4.2.4 La représentation des solutions

Chaque solution S est un vecteur de n éléments où n est le nombre de tous les items et leurs positions sont définies comme suit :

1. $S[i] = 0$ si l'item i n'appartient pas à la solution S.
2. $S[i] = 1$ si l'item i appartient à la partie antécédente de la solution S.
3. $S[i] = 2$ si l'item i appartient à la partie conséquente de la solution S.

Considérons à titre d'exemple l'ensemble d'items T=$\{t_1,t_2,...,t_6\}$, la solution $S_1 = \{0, 1, 0, 2, 2, 1, 0, 0, 0, 0\}$ représente la règle $r_1 : t_2, t_6 \Rightarrow t_4, t_5$.
Dans l'implémentation, on peut considérer une transaction et une solution s comme deux vecteurs binaires de $2 \times n$ bits On codifie la valeur 0 de s par 00, la valeur 1 par 01 et la veleur 2 est codifiée par 10. De cette manière, on a répondu à la première limite de l'algorithme BSO-ARM.

4.2.5 La détermination des régions

Dans cette partie, on propose trois stratégies pour la détermination des régions afin de bien explorer l'espace de recherche pour chaque abeille. Les stratégies appelées réspectivement(*modulo*, *next* et *syntactic*), sont détaillées comme suit :

La stratégie Modulo

Dans la stratégie modulo, chaque abeille a_j construit sa propre région en changeant successivement dans la solution Sref les bits $j + i \times Flip$ où i varie de 0 à $n - 1$ et $Flip$ est un paramètre donné.

Cette stratégie peut être utilisée si et seulement si le nombre d'abeilles est inférieur ou égal à $\frac{n}{Flip}$. Si la distance entre les solutions est le nombre de différents bits alors la distance entre les abeilles et la solution référence est égale à $\frac{n}{Flip}$.

L'algorithme de la stratégie modulo

```
Debut
 Entrées: Sref, flip, k (nombre d'abeilles)
 Sortie: Espace-abeilles: Matrice de [1...K][1...n]
```

```
  i=0
     Tanque i < K faire
        copier(Sref, Espace-abeilles[i])
        Pour j=i; j < n-flip; j=j+1 faire alors
         changer-bit(Espace-abeilles[i][j])
        fin_pour
        i=i+1
     fin_tanque
  retourner (Espace-abeilles)
 Fin
```

La Figure 4.1 illustre le scénario de cette stratégie en supposant que la solution référence Sref = 1102112.

La stratégie Next

Etant donné un nombre entier nommé *Flip*, dans la stratégie *next*, chaque abeille a_j modifie les premiers *Flip* bits de Sref tout en commençant à partir du j^{me} bit. Par exemple, Si $Flip = 3$, l'abeille a_0 change les bits 0, 1 et 2 et l'abeille a_2 change les bits 2,3, 4 et ainsi de suite. Cette stratégie est appelée *next* parceque à chaque fois que l'abeille a_j change le j^{me} bit elle change aussi les $Flip-1$ bits qui le suivent. La distance entre les abeilles et Sref est égale à *Flip*.

L'algorithme de la stratégie next

```
 Debut
  Entrées: Sref, flip, k (nombre d'abeilles)
  Sortie: Espace-abeilles: Matrice de [1...K][1...n]
  i=0
     Tanque i < K faire
        copier(Sref, Espace-abeilles[i])
        Pour j=i; j < i+flip; j=j+1 faire alors
         changer-bit(Espace-abeilles[i][j])
        fin_pour
```

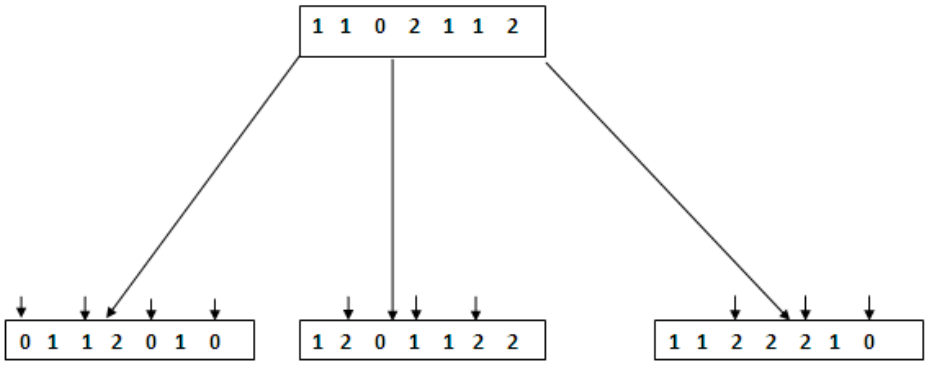


FIGURE 4.1 – *La stratégie modulo*

```
        i=i+1
     fin_tanque
  retourner (Espace-abeilles)
Fin
```

La Figure **??** illustre le scénario de cette stratégie avec la même solution référence Sref = 1102112.

La stratégie Syntactic

Contrairement aux deux premières stratégies, la stratégie *syntactic* applique une forme syntaxique lors de la génération des solutions en lieu de les générer d'une manière aléatoire. Pour celà, on associe la notion des poids à chaque solution.

Formelment, le poids d'une solution $S = a_0 a_1 a_2 \dots a_{n-1}$ noté $W(S)$ est défini comme $W(s) = \sum_{i=0 to n-1} a_i$

tel que

n est la taille de la solution.

Par exemple, si on considère la solution $S = 0\,1\,1\,2\,2\,1\,1$ alors $W(S) = 0 + 1 + 1 + 2 + 2 + 1 + 1 = 6$.

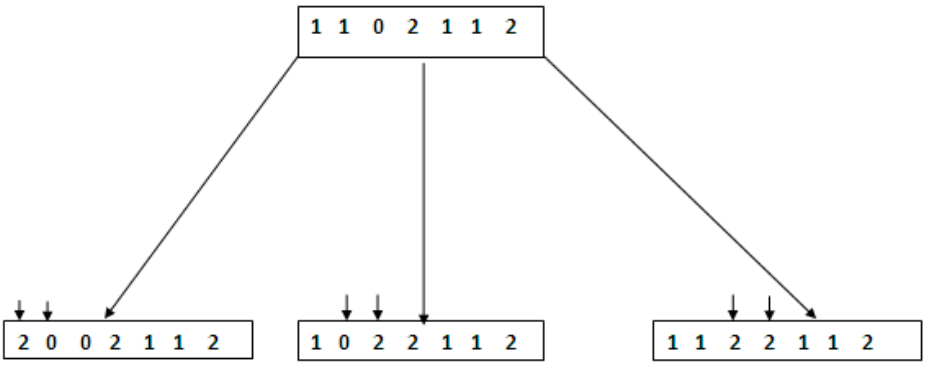


FIGURE 4.2 – *La stratégie next*

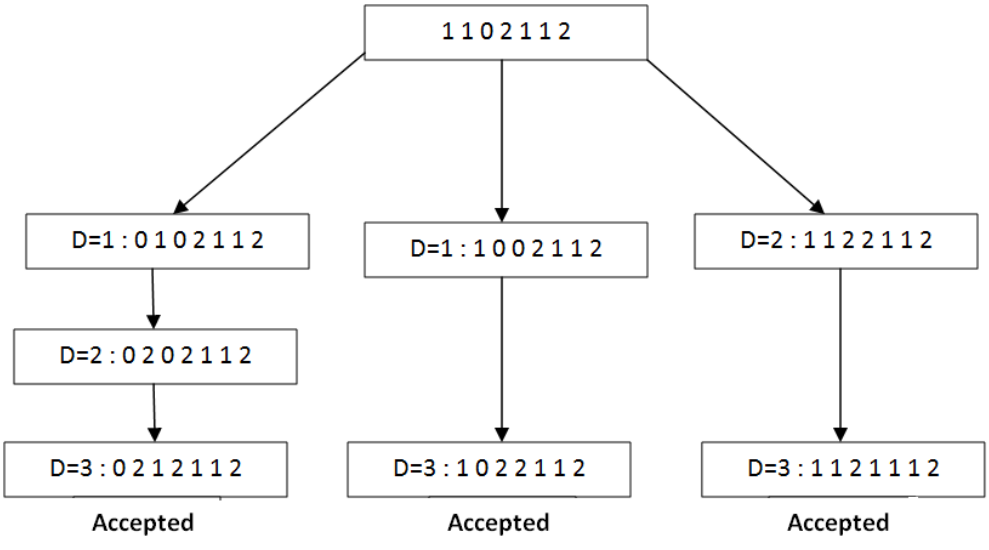


FIGURE 4.3 – *La stratégie syntactic*

Le principe de cette stratégie peut être résumé comme suit

- Si l'item t appartient aux solutions S_1 et S_2 en même temps. Plus précisement, s'il apparaît à la partie antécédente de S_1 et à la partie conséquente de S_2, alors il produit une distance de 1 entre S_1 et S_2.
- Si l'item t appartient à la partie antécédente de S_1 et n'appartient pas à S_2, alors il produit une distance de 1 entre S_1 et S_2.
- Si l'item t appartient à la partie conséquente de S_1 et n'appartient pas à S_2, alors il produit une distance de 2 entre S_1 et S_2.

Pour le premier et le deuxième cas, l'item t produit juste une distance de 1 entre les deux solutions car au moins il appartient à la partie antécédente d'une solution. Cependant, Concernant le dernier cas, il produit une distance de deux car il n'appartient pas à S_1 et il est la partie conséquente de S_2. Grâce à cette idée chaque bit i peut générer une solution qui a une certaine distance de valeur $Distance$ avec Sref. Premièrement, l'algorithme calcule $W(Sref)$. Puis, chaque abeille a_j change des bits successives de Sref en commençant par le j^{me} bit de Sref. Chaque abeille a_j répete ce processus juqu'à ce qu'elle obtienne une solution S qui satisfait une de ces contraintes :
$W(s) = W(Sref) - Distance$
$W(s) = W(Sref) + Distance.$

L'algorithme de la stratégie syntactic

```
Debut
 Entrées: Sref, Distance, k (nombre d'abeilles)
 Sortie: Espace-abeilles: Matrice de [1...K][1...n]
 Calcul-poids(Sref)
 i=0
    Tanque i < K faire
       copier(Sref, Espace-abeilles[i])
       Pour j=1; j < n; j=j+1 faire alors
        changer-bit(Espace-abeilles[i][j])
        Si (W(Espace-abeilles[i])=W(Sref)+|Distance|) alors
```

```
            Accepter(Espace-abeilles[i])
            sortir de la boucle
            Sinon
             Calculer le nouveau center pour chaque cluster
         fin_si
        fin_pour
        i=i+1
    fin_tanque
 retourner (Espace-abeilles)
Fin
```

La Figure 4.3 illustre le scénario de cette stratégie en utilisant $1, 1, 0, 2, 1, 1, 2$, comme solution référence et en prenant 3 comme valeur de paramètre *Distance*.

4.2.6 L'algorithme HBSO-ARM

Le précédent algorithme améliore l'étape de diversification du premier algorithme BSO-ARM en proposant trois stratégies de détermination des régions. Afin d'équilibrer entre l'intensification et la diversification et de bien explorer les différentes régions de recherche, nous avons proposé dans [109] [111], une autre extension (HBSO-ARM) du deuxième algorithme. C'est une sorte d'hybridation entre la métaheuristique BSO et la métaheuristique locale (la recherche tabou). On applique le même principe que BSO-ARM sauf dans l'exploration de région de chaque abeille se fait grâce à la recherche tabou. En effet, la différence ici est dans le calcul de voisinage, il est calculé par chaque abeille en utilisant la recherche tabou. Dans un premier temps, tous les voisins d'une abeille donnée sont générés en modifiant un bit à la fois de cette abeille. Si le meilleur voisin n'appartient pas à la liste tabou alors il est inséré dans la liste tabou. Puis, on applique le même processus pour ce voisin et ainsi de suite jusqu'à ce que le nombre d'itérations de la recherche tabou (IMAX-RT) soit atteint. L'algorithme général de HBSO-ARM est expliqué dans la Figure 4.4. Il commence par calculer Sref, puis il calcule les K régions en utilisant une des trois stratégies du deuxième algorithme. Chaque région est assignée par la suite à une abeille. Ensuite, chaque abeille explore sa région en utilisant la recherche tabou. Le résultat est mis dans la table Dance.

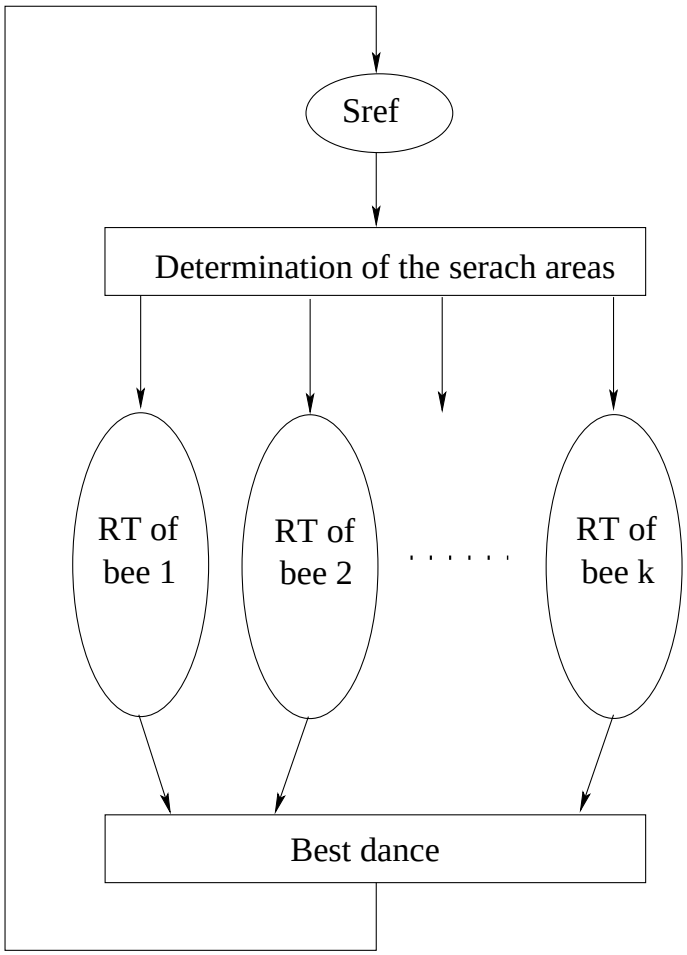


FIGURE 4.4 – L'algorithme HBSO-ARM

A ce moment, une communication entre les abeilles est effectuée afin de trouver la meilleure solution qui va devenir par la suite la solution référence pour la prochaine itération. Ce processus doit être répété jusqu'à ce que le nombre d'itérations de la métaheuristique BSO soit atteint.

4.2.7 Etude théorique et expérimentale des algorithmes séquentiels proposés

Dans cette section, on propose une étude théorique aprés une étude expérimentale des algorithmes séquentiels proposés.

Etude théorique

La complexité des deux algorithmes BSO-ARM et HBSO-ARM est dans O(IMAX × Coût_détermination-régions + K× Coût_calcul-voisinage) tel que K est le nombre d'abeilles, IMAX est le nombre maximal d'itérations, Coût_détermination-régions est le coût produit par la stratégie de détermination des régions et Coût_calcul-voisinage est le coût produit par le calcul de voisinage.

Preuve Dans chaque itération, chaque abeille détermine sa région en utilisant une des stratégies de détermination des régions, ce qui résulte K différentes régions. Chaque abeille explore les n voisins avec l'opération de calcul du voisinage. Dans le pire des cas, ce processus peut être répété $IMAX$ fois. Le coût des algorithmes BSO-ARM et HBSO-ARM sera donc :
$Cot_{BSO-ARM}$ = IMAX× [$Cot_{dtermination-rgions}$+$K$ × $Cot_{calcul-voisinage}$].
$Cot_{HBSO-ARM}$ = IMAX× [$Cot_{dtermination-rgions}$+$K$ × $Cot_{recherche-tabou}$].

Commençant par la détermination de l'espace des régions, son coût dépend de la stratégie utilisée, trois cas peuvent être distinguer :

- **Premier cas : La stratégie** *modulo* : Cette stratégie s'effectue aux K itérations, une pour chaque abeille. Le coût de l'opération copier est $O(n)$, tel que n est le nombre de tous les items. Ensuite, chaque abeille modifie sa solution avec [n÷ Flip] fois. Par conséquent, la complexité de la modification est O([n÷ Flip]). La complexité totale de cette stratégie est : $Cot_{modulo} = O(K \times n \times Flip)$.

- **Deuxième cas : La stratégie** *next* : Comme dans la première stratégie, K itérations sont réalisées avec la copie dans chaque itération de la solution référence en une seule abeille. Ensuite chaque solution est modifiée $Flip$ fois. La complexité de cette stratégie est :
$Cot_{next} = O(K \times n \times Flip)$.

- **Troisème cas : La stratégie** *syntactic* : Contrairement à modulo et next, cette stratégie calcule les poids de Sref, dont son coût est O(n). Ensuite,

K itérations sont réalisées tel que dans chacune, Sref est copiée dans une abeille. Puis, chaque solution est modifiée *Distance* fois (dans le pire des cas). La complexité de cette stratégie est,
$Cot_{syntactic} = O(K \times n \times Distance)$.

On note que pour le calcul de voisinage, BSO-ARM utilise une recherche local simple, cependant HBSO-ARM utilise la métaheuristique (la recherche tabou). Pour celà, deux complexités sont considérées :

- **La stratégie** *recherche locale simple* : Cette stratégie est réalisée en une seule itération, de telle sorte que chaque abeille explore n voisins, le coût de cette stratégie est : $Coût_{Recherche\ Local\ Simple}$=$O(n)$.

- **La stratgie** *recherche tabou* : Contrairement à la première stratégie, celle-ci est réalisée dans $IMAX - RT$ itérations tel que $IMAX - RT$ est le nombre maximum d'itérations appliqué à la métaheuristique recherche tabou. De plus, Dans chaque itération, une recherche locale simple est effectuée. En effet, la complexité de la stratégie recherche tabou est
$Cot_{Recherche\ Tabou} = O(n\ \times IMAX - RT)$.

La Table 4.1 présente la complexité théorique des deux algorithmes (BSO-ARM et HBSO-ARM) en fonction de la stratégie de détermination des région utilisée.

	La stratégie modulo	La stratégie next	La stratégie syntactic
BSO-ARM	V1	V1	V2
HBSO-ARM	V3	V3	V4

TABLE 4.1 – La complexité théorique des algorithmes (BSO-ARM, HBSO-ARM avec différentes stratégies de détermination des régions

Tel que
V1=$IMAX \times K \times n \times Flip$.
V2=$IMAX \times K \times n \times Distance$.
V3$IMAX \times K \times n \times IMAX - RT \times Flip$.

Type	Noms	taille(transactions)	taille(items)	taille moyenne
petite	Bolts	40	8	8
petite	Sleep	56	8	8
petite	Pollution	60	16	16
petite	Basketball	96	5	5
petite	IBM Q.St	1000	40	20
petite	Quake	2178	4	4
petite	Chess	3196	75	37
petite	Mushroom	8124	119	23
moyenne	Pumbs_star	40385	7116	50
moyenne	BMS-WebView-1	59602	497	2.5
moyenne	BMS-WebView-2	77512	3340	5
moyenne	Korasak	80769	7116	50
moyenne	retail	88162	16469	10
moyenne	Connect	100000	999	10
large	BMP POS	515597	1657	2.5
large	WebDocs	1692082	5267656	n'est pas mentionnée

TABLE 4.2 – Description des données

V4=$IMAX \times K \times n \times IMAX - RT \times Distance$.

Etude expérimentale

Ce qui suit, plusieurs expérimentations ont été effectuées dans le but de valider les approches proposées. Commençant à partir de la petite base de transactions (Bolts) jusuq'à la grande base de transactions(WebDocs), les résultats obtenues vont être comparés avec l'état de l'art des algorithmes d'extraction des règles d'association.

- **Déscription des données :** Afin d'évaluer la performance des approches proposées, on a considéré plusieurs bases de transactions avec différentes tailles. Ces bases de transactions sont trés connues dans la communauté d'extraction des règles d'association et elles sont disponibles dans Data-set Repository [89] et Bilkent University Function Approximation Repository [88] réspectivement. A partir de la petite base (*Bolts*) à la grande base (*WebDocs*), les bases sont divisées en trois catégories (petite, moyenne, large). La Table 4.2 présente la description des différentes bases de tran-

Nombre de transactions	Version1/Version2
20000	1.15
40000	1.60
60000	1.84
80000	1.96
100000	2.15

TABLE 4.3 – comparaison de la première version de BSO-ARM avec sa deuxième version en terme de CPU Time (Sec)

sactions utilisées ensuite dans l'expérimentations. D'aprés cette table, on remarque que les bases de transactions diffèrent en fonction de la taille moyenne des items. D'une part, il y'a de grandes bases avec un nombre réduit d'items par transaction. D'autre part, il y'a de petites bases avec un nombre important d'items par transaction.

- **La version 1 de BSO-ARM avec la Version 2 de BSO-ARM**
  La Table 4.3 présente le rapport entre le temps d'exécution de la première version de l'algorithme BSO-ARM et la deuxième version de l'algorithme BSO-ARM en utilisant la stratégie modulo comme stratgéie de détermination des région. En variant le nombre de transactions de 20000 à 100000, le rapport du temps d'exécution entre les deux approches augmente. On peut expliquer celà par la bonne représentation des solutions de la deuxième version de l'algorithme BSO-ARM. De plus, pour 100000 transactions le temps d'exécution de la première version de l'algorithme BSO-ARM est deux fois plus grand que le temps d'exécution de la deuxième version de l'algorithme BSO-ARM.
- **Résultat et comparaison de la deuxième version de BSO-ARM :**
  Afin de montrer la performance de la deuxième version de l'algorithme de BSO-ARM par rapport à l'état de l'art des algorithmes d'extraction des règles d'association, on applique la deuxième version de BSO-ARM sur les différentes bases de transactions.
  Malheureusemen, tous les algorithmes d'extraction des règles d'association approchés ont été testé sur de petites bases de transactions. Pour cette raison, nos expérimentations sont divisées en deux parties. En utilisant les petites bases de transactions, on compare notre algorithme avec les algo-

Bases	modulo	next	syntactic	ACO_R	SA	G3APARM	ARMBGSA
Bolts	1.0	1.0	1.0	0.69	0.60	0.92	0.45
Sleep	1.0	1.0	1.0	0.67	0.53	0.90	0.39
Pollution	1.0	1.0	0.73	0.66	0.50	0.92	0.56
Basketball	0.97	1.0	0.92	0.61	0.66	0.93	0.45
IBM Q.St	0.94	0.9	0.93	0.45	0.30	0.88	0.40
Quake	1.0	1.0	1.0	0.73	0.52	0.90	0.39
Chess	0.88	0.88	0.60	0.3	0.15	0.86	0.38
Mushroom	0.52	0.75	0.72	0.1	0.05	0.85	0.35

TABLE 4.4 – Comparaison de l'algorithme BSO-ARM avec d'autres approaches en terme de la qualité des solutions trouvées

Les bases de transactions	modulo	next	syntactic	Apriori	FPGrowth
Pumbs_star	150	150	180	500	600
BMS-WebView-1	350	370	400	1000	800
BMS-WebView-2	500	600	650	2500	2300
Korasak	800	810	820	3000	2900
Retail	1075	1150	1200	4500	3800
Connect	950	950	1000	2600	2900

TABLE 4.5 – BSO-ARM avec les méthodes exactes en terme de temps CPU(Sec)

rithmes apprachés d'extraction des règles d'association, et en utilisant les bases de transactions de taille moyenne, notre algorithme est comparé avec les algorithmes exacts d'extraction des règles d'association. Toutes les expérimentations sont testées sur un ordinateur 4GB Intel Core I3 machine running Windows 7. Tous les programmes sont implémentés en Java. Le nombre d'abeilles est fixé à 50, le nombre d'itérations IMAX est fixé à 100, tous les résultats sont la moyenne de 100 tests successifs.

La Table 4.4 résume les résultats obtenus en terme de qualité des solutions trouvées de la deuxième version de l'algorithme BSO-ARM avec différentes stratégies de détermination des régions, et les algorithmes approchés d'extraction des règles d'association suivants : ACO_R, SA, G3APARM et ARMBGSA. On remarque que ACO_R, SA et ARMBGSA donnent de mauvais résultats. En effet, la moyenne de la fonction fitness des solutions trouvées par lesdits algorithmes ne dépasse pas 0.75. Ce qui explique l'efficacité de la stratégie de recherche de ces algorithmes. Cependant, G3APARM et nos stratégies ne descendent pas en générale en dessous de 0.90 sauf dans le cas de la base de transactions Chess. Ces bons résutats de la deuxième version de notre algorithme sont obtenus grâce à la bonne exploration de l'espace de recherche affecté à chaque abeille.

La Table 4.5 présente le temps d'exécution en traitant les différentes bases de transactions. On peut dire que la deuxième version de l'algorithme BSO-ARM surpasse les algorithmes exacts d'extraction des règles d'association en terme de temps d'exécution grâce à la rapidité et l'efficacité de notre recherche. De plus, le temps d'exécution de nos stratégies ne dépasse pas 1200 sec alors que Apriori et FPGrowth atteindent 4500 sec dans toutes les bases de transactions sauf dans Retail. Basé sur ces expérimentations, on peut dire que le nombre d'items influence sur le temps d'exécution. En effet, malgré que le nombre de transactions dans Connect est plus grand que dans Retail et Korasak, le temps d'exécution de tous les algorithmes sur Connect est moins par rapport au temps d'exécution dans Retail et Korasak en raison de son petit nombre d'items qui est 1000 items.

- **Comparaison de la deuxième version de BSO-ARM avec HBSO-ARM :** La Table 4.12 résume tous les résultats obtenus en terme de la

Noms des bases des transactions	HBSO-ARM	BSO-ARM (version2)
Bolts	1.0	1.0
Sleep	1.0	1.0
Pollution	1.0	1.0
Basket ball	0.97	0.97
IBM Quest Standard	0.94	0.94
Quake	1.0	1.0
Chess	0.90	0.88
Mushroom	0.75	0.52
Pumbs_star	0.72	0.40
BMS-WebView-1	0.55	0.35
BMS-WebView-2	0.70	0.55
Korasak	0.65	0.35
Retail	0.62	0.36
Connect	0.50	0.26
BMP POS	0.47	0.20
WebDocs	bloqué	bloqué

TABLE 4.6 – Comparaison de HBSO-ARM avec la deuxième version de BSO-ARM en terme d'évaluation des solutions

noms des bases des transactions	HBSO-ARM	BSO-ARM (version2)
BMP POS	12562	10000
WebDocs	Bloqué	Bloqué

TABLE 4.7 – Comparaison de HBSO-ARM avec BSO-ARM (version2) en terme de temps CPU (Sec)

fonction fitness par l'exécution de HBSO-ARM et BSO-ARM en utilisant la stratégie modulo comme stratégie de détermination des régions. On remarque que BSO-ARM donne de mauvais résultats par rapport à HBSO-ARM. En effet, la fitness moyenne de BSO-ARM ne dépasse pas 0.50. On peut expliquer celà par la mauvaise exploration de voisinage de BSO-ARM par rapport à HBSO-ARM qui utilise la recherche tabou dans le calcul des voisinages de différentes abeilles.

La Table 4.7 présente le temps CPU de BSO-ARM et HBSO-ARM sur de larges bases de transactions. Les deux algorithmes sont bloqués en appliquant la base de transactions WebDocs. Celà est dû par le fait que dans

WebDocs, il y'a un nombre important de transactions (plus qu'un million et demi de transactions). Par conséquent, la parallélisation peut certianement apporter plus d'efficacité des deux algorithmes.

4.3 les essaims d'abeilles sur GPU pour les règles d'association

Avant de développer la version parallèle de l'algorithme BSO-ARM, on doit savoir quelle est l'opération la plus coûteuse dans cet algorithme. Pour celà, des indicateurs sont mis avant et aprés chaque étape de l'algorithme BSO-ARM. La Table.4.8 illustre le temps d'exécution de chaque étape sur différentes bases de transactions.

On note : Nom des bases des transactions(DN), le nombre de transactions (DST), Calcul de fitness(FC), calcul de voisinage (NS), Détermination des régions (DSA) et l'étape de dance des abeilles.

DN	DST	FC (%)	NS (%)	DSA (%)	DS (%)
Bolts	40	**50.00**	30.00	18.00	1.00
Sleep	56	**46.00**	32.00	20.00	2.00
Pollution	60	**70.00**	20.00	9.00	1.00
Basket ball	96	**40.00**	30.00	28.00	2.00
IBM Que. St.	1000	**98.00**	1.00	0.80	0.20
Quake	2178	**98.00**	1.10	0.70	0.20
Chess	3196	**98.00**	1.00	0.90	0.10
Mushroom	8124	**99.00**	0.60	0.35	0.05
Pumbs_star	40385	**99.00**	0.50	0.47	0.03
BMS-WebView-1	59602	**99.00**	0.70	0.20	0.10
BMS-WebView-2	77512	**99.50**	0.20	0.20	0.10
Korasak	80769	**99.60**	0.02	0.01	0.01
retail	88162	**99.50**	0.02	0.02	0.01
Connect	100000	**99.70**	0.02	0.01	0.01
BMP POS	515597	**99.70**	0.01	0.01	0.01
—–	Average	**86.34**	7.76	5.24	0.65

TABLE 4.8 – Proportion du temps CPU des différentes étapes de BSO-ARM avec différentes bases de transactions (%)

On remarque que dans les bases de transactions, le calcul de fitness est l'étape la plus coûteuse. En effet, le calcul de fitness consomme en moyenne 86% du temps global de BSO-ARM. De plus, ce rapport augmente en traitant des bases de transactions trop grande et il atteint les 99.70% pour BMP-POS.

Pour celà et comme deuxième contribution de ce chapite, la parallélisation du calcul de fitness est nécessaire. Motivés par le succées des approches des essaims d'abeilles sur l'architecture GPU qui permettent de résoudre plusieurs problèmes complexes, on propose dans cette section deux versions de l'algorithme BSO-ARM appelées Single Evaluation on GPU (SEGPU) et Multiple Evaluation on GPU (MEGPU) afin de résoudre les limites de l'algorithme BSO-ARM.

4.3.1 L'algorithme SEGPU

On remarque que l'opération principale qui nécessite un temps de calcul considérable est l'évaluation d'une solution. Pour calculer l'évaluation de chaque règle, la base de transactions doit être parcourue afin de déterminer le support et la confiance de cette règle.

SEGPU consiste à exécuter séquentiellement sur CPU les opérations qui demandent peu de temps comme (NS, DSA, DS) et charger l'opération de l'évaluation sur GPU qui demande un temps de calcul intensif. A chaque fois, une seule règle r est évaluée, en d'autres termes, tous les blocs de GPU sont occupés pour évaluer une seule règle r. En effet, chaque transaction est affectée à un seul thread sur GPU. chaque thread i vérifie la présence de cette règle r à la i^{me} transaction. Si la règle appartient à la transaction, le thread i met dans son entrée la valeur 1, sinon, il met la valeur 0. Quand tous les threads traitent leurs transactions, un thread particulier additionne toutes les entrées et envoie le résultat en CPU. La base de transactions est copiée une seule fois au GPU en début de l'algorithme. L'architecture générale de cet algorithme est illustrée dans la Figure 4.5.

Comme mentionnée dans [90], on dit qu'une méthode d'optimisation implémentée sur GPU est efficace si elle vérifie les chanllenges de GPU. Ce qui suit, on présente le comportement de cet algorithme avec ces challenges.

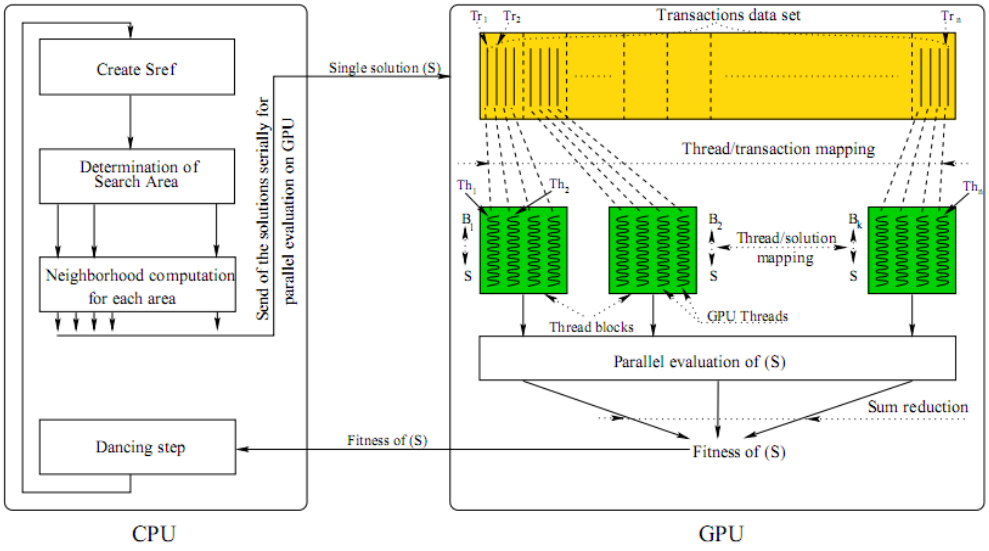


FIGURE 4.5 – L'architecture de SEGPU

La divergence de thread

Dans GPU, les threads du même Wrap(32 threads par Wrap dans l'architecture G80 GPU model) doivent exécuter la même instruction. Si cette contrainte n'est pas satisfaite, alors la performance du système diminue. Ce phénomène est appelé par la divergence de thread.

Dans SEGPU, pour évaluer une solution donnée, chaque thread compare une transaction avec cette solution. Si on bénéficie de la représentation binaire fournie par BSO-ARM où chaque thread parcours les $2 \times n$ bits d'une solution donnée pour chaque item dans une transaction, alors, tous les threads comparent le même nombre de bits. Le processus de comparaison est terminé si l'item courant n'appartient pas à la solution courante. Ce majeur problème doit être pris en considération afin de minimiser la divergence de thread. Par conséquent, le nombre de la divergence de thread ($/$TD)) est calculé en fonction de nombre de comparaisons comme suit :

$$TD = max\{max\{(|t_{(k*32)+i}| - |t_{(k*32)+j}|\}/(i,j,k) \in [1...32]^3\}. \quad (4.1)$$

tel que $|t_{(k*32)+i}|$ est la de la $(k*32)+i^{me}$ transaction affectée au i^{me} thread qui est alloué au k^{me} wrap.

La gestion des accées mémoire

Dans GPU, Il existe différentes mémoires avec différentes tailles. En utilisant la mémoire globale, on peut sauvegarder une grande quantité de données mais avec un temps d'accés considérable ce qui génère un temps d'exécution important. Cependant, en utilisant la mémoire partagée, on peut bénéficier d'un temps d'accés trés réduit avec l'enregistrement d'une taille trés limitée de données.

En se basant sur le principe de SEGPU et pour évaluer une seule solution S, chaque thread utilise S. Pour celà, la solution S doit être alouée dans la mémoire partagée de chaque bloc. Une fois CPU génère S, il l'envoi en mémoire globale. Ensuite, on crée une copie d'elle dans la mémoire partagée de chaque bloc. Par la suite, chaque thread compare S avec une seule transaction. La taille de chaque transaction est n octets. Si on considère que la taille de chaque bloc en fonction du nombre des threads est l alors la mémoire partagée de chaque bloc occupe ((l+1)*N) octets. Cependant, il existe différents types de mémoires partagées avec différentes tailles, maintenant si on suppose que la taille de la mémoire partagée de chaque bloc est x, alors on distingue deux cas :

- Si x est supérieur à ((l+1)*N) alors on envoie toutes les transactions dans la mémoire partagée.
- Si x est inférieur à ((l+1)*N) alors on transforme les $((x/N)-1)$ premières transactions de chaque bloc dans sa mémoire partagée. Les (l-((x/N)-1) transactions restantes pour chaque bloc sont allouées à la mémoire globale, les (l-((x/N)-1) premières transactions sont assignées aux derniers (l-((x/N)-1) threads du premier bloc, les deuxièmes (l-((x/N)-1) transactions sont assignées aux derniers (l-((x/N)-1) threads du deuxième bloc et ainsi de suite.

La synchronisation des threads

Un bon algorithme est capable de minimiser la communication et la synchronisation entre les threads. Dans l'algorithme SEGPU, on n'observe aucune communication entre les threads. Pour la synchronisation, il y'a seulement une seule synchronisation entre les threads lors de l'évaluation d'une seule solution. Le nombre de synchronisation total de l'algorithme est égal au nombre total des solutions générées, soit $(n * K * IMAX)$.

La communication CPU/GPU

Dans un premier temp, on envoie les transactions en GPU avec un seul acces CPU/GPU. La taille de données transmises est (m*n) octets. Puis, dans chaque itération, on envoie tous les voisins de chaque abeille au GPU pour l'évaluation. Si on a K abeilles et n voisins de chaque abeille, alors, dans chaque itération on fait (n*K) communications entre CPU et GPU. De plus, le nombre total de communication de CPU/GPU de l'algorithme SEGPU est (IMAX*k*N)+1 avec (N^2*k*IMAX)+(M*N) octets transmis.

Les pseudo codes exécutés en CPU et en GPU de l'algorithme SEGPU sont donnés comme suit :

L'algorithme SEGPU : CPU

```
Debut
 créer la solution référence Sref
 cudaMemcpy(Sref, cudaMemcpyHostToDevice)
 Fitness(Sref)=Evaluer-dans-GPU(Sref)
 cudaMemcpy(Fitness(Sref), cudaMemcpyDeviceToHost)
    Pour i=1 à IMAX faire
       générer K abeilles à partir de Sref
       Pour chaque abeille a faire
         détermine n voisins à partir de a
```

```
        fin_pour
        Pour chaque S dans n *K solutions générées faire
          cudaMemcpy(S, cudaMemcpyHostToDevice)
          Fitness(S)=Evaluer-dans-GPU(S)
          cudaMemcpy(Fitness(S), cudaMemcpyDeviceToHost)
        fin_pour
        extraire la meilleure solution de la table Dance
        affecter la meilleure solution à Sref
     fin_pour
  générer les règles d'association à partir des solutions trouvées
 Fin
```

L'algorithme SEGPU : GPU

```
 Debut
  idx=blockIdx.x * blockDim.x + threadIdx.x
  Comparer la solution S et la transaction d'indice idx
     Si (S n'appartient pas à cette transaction) alors
        count[idx]=0
         Sinon
          count[idx]=1
     fin_si
     Fitness(S)=Sum_Reduction(count)
     cudaMemcpy(Fitness(S), cudaMemcpyDeviceToHost)
 Fin
```

4.3.2 L'algorithme MEGPU

Généralement, on dispose un nombre important de solutions, si on envoie à chaque fois une seule solution pour l'évaluation, un temps d'execution considérable est nécessaire pour évaluer toutes ls solutions générées.

Afin de minimiser E/S opérations, dans chaque étape de l'algorithme, plusieurs solutions sont envoyées en GPU pour l'évaluation. par la suite, l'ensemble de threads de chaque bloc est affecté à une seule solution. Les différences principales entre cet algorithme et SEGPU sont le temps de communications entre

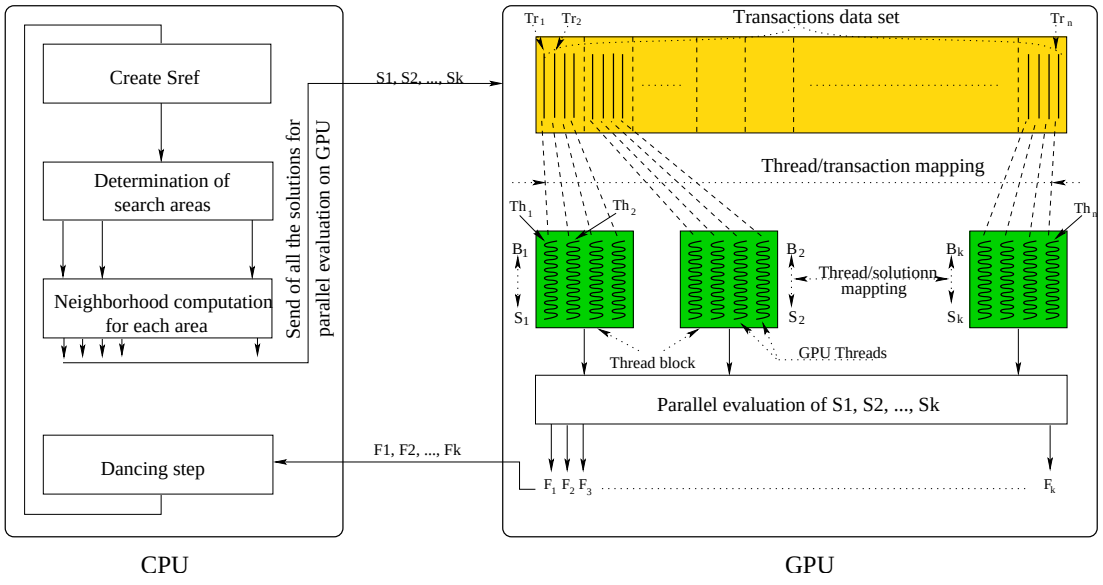


FIGURE 4.6 – L'architecture de MEGPU

CPU et GPU et le nombre de threads occupés pour l'évaluation d'une solution. De telle sorte que, SEGPU permette de bien bénéficier de GPU en déclenchant un nombre important de threads pour l'évaluation d'une seule solution. Cependant, MEGPU déclenche juste un bloc par solution, mais il établit moins de communication entre CPU et GPU. On peut dire que SEGPU favorise le calcul intensif à la communication CPU/GPU, tandis que MEGPU favorise la communication CPU/GPU au calcul intensif.

L'architecture de l'algorithme MEGPU est détaillée dans la Figure 4.6.

La divergence de thread

En regardant l'architecture MEGPU, chaque thread compare p transactions avec une seule solution. Cet algorithme est aussi bénéficié par la bonne représentation de la solution. Si on considère que la divergence de thread est la différence entre le nombre de comparaisons faites par les différents threads alors la divergence de thread sera le maximum de toutes les comparaisons établies entre deux threads situés dans le même wrap. De plus, le nombre de comparaisons faites par chaque thread pour une transaction est égal à la taille de cette transaction.

Formellement, la divergence de thread peut être calculée comme :

$$TD = max\{max\{\sum_{k=1}^{p}(|t_{(i*p)+k+(32*l)}|-|t_{(i*p)+k+(32*l)}|)/(i,j) \in [1...32]^2 etl \in [1...\frac{M}{p*32}]\}. \tag{4.2}$$

.

La gestion des accées mémoire

Selon le principe de l'algorithme MEGPU, dans chaque itération, on transforme $(K \times n)$ solutions à la fois dans la mémoire globale. Chaque solution est évaluée par les threads de même bloc. C'est pour cette raison qu'on envoie chaque solution s_i à la mémoire partagée de i^{me} bloc.
Chaque bloc a l'accées à toute la base de transactions. Afin de bien gérer les accées aux données, on doit copier le maximum de transactions dans les mémoires partagées. On suppose que la taille d'une seule transaction est n et la taille de la mémoire partagée de chaque bloc est x. On transforme alors les $\frac{x}{n}$ transactions au mémoire partagée de chaque bloc et les transactions restantes sont alouée en mémoire globale. Par conséquent, si chaque thread utilise p transactions pour l'évaluation alors les $\frac{\frac{x}{n}}{p}$ premiers threads de chaque bloc ont l'accées à la mémoire partagée, les autres ont l'accées à la mémoire globale.

La communication CPU/GPU

Comme dans l'algorithme SEGPU, on envoie les transactions sur GPU avec un seul accés CPU/GPU, la taille des données transmises est (m*n) octets. Puis, dans chaque itération on envoie tous les voisins générés à la fois. On a IMAX itération dont chacune on a un seul accés CPU/GPU avec (n*K) solutions transmises. A la fin de l'algorithme, on aura (IMAX+1) accées CPU/GPU.

la synchronisation des threads est la même que l'algorithme SEGPU. On a aucune communication entre les threads et (n*K*IMAX) synchronisations. La synchronisation est faite entre un thread spécifique et seullement les threads de chaque bloc.

Les pseudo codes exécutés en CPU et en GPU de l'algorithme MEGPU sont donnés comme suit :

L'algorithme MEGPU : CPU

```
Debut
 créer la solution référence Sref
 cudaMemcpy(Sref, cudaMemcpyHostToDevice)
 Fitness(Sref)=Evaluer-dans-GPU(Sref)
 cudaMemcpy(Fitness(Sref), cudaMemcpyDeviceToHost)
    Pour i=1 à IMAX faire
       générer K abeilles à partir de Sref
       Pour chaque abeille a faire
         détermine n voisins à partir de a
       fin_pour
       Initialiser Buff_temp à vide
       Pour chaque S dans n *K solutions générées faire
         Ajouter S à Buff_temp
       fin_pour
       cudaMemcpy(Buff_temp, cudaMemcpyHostToDevice)
       Buff_evaluations=Evaluer-Dans-GPU(Buff_temp)
       Pour chaque élément e dans Buff_evaluations faire
         Récupérer e et le mettre dans la table Dance
       fin_pour
       extraire la meilleure solution de la table Dance
       affecter la meilleure solution à Sref
    fin_pour
 générer les règles d'association à partir des solutions trouvées
Fin
```

L'algorithme MEGPU : GPU

```
Debut
idx=blockIdx.x * blockDim.x + threadIdx.x
```

```
Comparer la solution Buff[blockIdx.x]
Pour i=0 à l transactions faire
Si(Buff[blockIdx.x]vérifie la transaction(i*blockDim.x)+idx alors
 count[blockIdx.x][i]=1
  Sinon
   count[blockIdx.x][i]=0
fin_si
fin_pour
Fitness(Buff[blockIdx.x])=Sum_Reduction(count[blockIdx.x])
cudaMemcpy(Fitness(Buff[blockIdx.x]), cudaMemcpyDeviceToHost)
Fin
```

4.3.3 Etudes théorique et expérimentale des algorithmes parallèles proposés

Etude théorique

Dans cette section, on va faire une analyse théorique entre les algorithmes SEGPU et MEGPU en calculant la divergence de thread théorique. Cette divergence peut être calculée en fonction de la distribution des items dans la base de transactions. Par conséquent, on distingue les deux cas possibles :

- Distribution irrégulière des items : Dans ce premier cas, il y'a une grande différence entre les tailles des transactions. Le pire des cas de l'algorithme SEGPU consiste à trouver dans le même wrap une transactions contenant tous les items et une autre transaction qui contient un seul item. Pour celà, on peut converger la divergence de thread de SEGPU quand le nombre de transactions est trés considérable au nombre maximal d'items moins un et on peut écrire :

$$\lim_{m \to +\infty} TD(SEGPU, m) = n - 1. \tag{4.3}$$

  Le pire des cas de l'algorithme MEGPU consiste à trouver dans le même wrap deux threads de p transactions chacun. Les transactions assignées au

premier thread contiennent en tous p items. Cependant, les transactions allouées en deuxième thread contiennent en tous $n*p$ items. La divergence de thread de cet algorithme peut converger vers (p*(n-1)) et on écrit :

$$\lim_{m \to +\infty} TD(MEGPU, m) = p * (n - 1). \tag{4.4}$$

Exemple

Considérons la base de transactions irrégulière suivante (voir la Table 4.9) et un seul wrap de 4 threads :

En utilisant SEGPU, TD=max({1-1, 50-1, 20-1, 50-20})=max{0,49,19,30}=49.

Transaction	Nombre d'items
1	1
2	1
3	50
4	20

TABLE 4.9 – Exemple d'une distribution irrégulière

En utilisant MEGPU, et avec p=2, le premier thread traite la première et la deuxième transaction alors que le deuxième thread traite la troisième et la quatrième transaction alors :
TD=max((50+20)-(1+1))=68.

– Distribution régulière des items : Contrairement au premier cas, on a ici une petite différence entre les tailles des différentes transactions. On pose que la variation entre les transactions v_1, on aura donc :

$$\lim_{m \to +\infty} TD(SEGPU, m) = v_1. \tag{4.5}$$

Maintanant, pour l'algorithme MEGPU, on pose que la variation des tailles des différentes transactions alouées à chaque thread i est v_i. La somme des variations de chaque thread i est donc (p*v_i). La divergence de thread de MEGPU peut converger à :

Transaction	Nombre d'items
1	5
2	7
3	6
4	8

TABLE 4.10 – Example d'une distribution régulière

$$\lim_{m \to +\infty} TD(MEGPU, m) = max(p * (v_i - v_j)). \tag{4.6}$$

Maintenant, avec une distribution régulière on assume :

$v_1 \simeq v_2 \simeq v_3 \simeq v_4 \simeq ... \simeq v_k$
Pour celà
(v_n) est une suite de cauchy [**?**], c'est à dire

$$\lim_{(i,j) \to +\infty} (v_i - v_j) = 0. \tag{4.7}$$

Aussi, (p_n) est une suite constante de valeur p. (p_n) est donc une suite cauchy particulière car elle vérifie

$$\lim_{(i,j) \to +\infty} (p_i - p_j) = p - p = 0. \tag{4.8}$$

En utilisant la propriété de cauchy, on obtient

$$\lim_{m \to +\infty} p_n * v_n = 0. \tag{4.9}$$

Ce qui implique

$$\lim_{m \to +\infty} TD(MEGPU, m) = \lim_{m \to +\infty} p * (v_i - v_j) = 0. \tag{4.10}$$

Exemple Considérons la base de transactions régulière suivante (voir la Table 4.10)
En utilisant SEGPU
TD=max{7-5,6-5,7-6,8-5,8-7,8-6}=3.
Pour MEGPU et avec p=2

TD=max{(8+6)-(5+7)}=2.

On peut dire que TD de la distribution régulière est meilleure que TD dans le cas d'une distribution irrégulière.

Etude expérimentale

Afin d'évaluer la performance des algorithmes parallèles SEGPU et MEGPU, on considère les mêmes bases de transactions qui sont utilisées dans la section 4.2.7.2. Les deux algorithmes sont implémentés en utilisant C-CUDA 4.0 et les expérimentations ont été réalisées en connectant une machine CPU avec la machine GPU. La machine CPU est 64-bit quad-core Intel Xeon E5520 contenant une horloge d'une vitesse qui est égale à 2.27GHz. La machine GPU est Nvidia Tesla C2075 avec 448 CUDA cores (14 multiprocessors avec 32 cores chacun), une horloge d'une vitesse de 1.15GHz, une mémoire globale de 2.8GB, une mémoire partagée de 49.15KB, et la taille d'un wrap est 32. La CPU et le GPU sont tous les deux utilisés avec une précision unique.

Ce qui suit, plusieurs test ont été employés afin de valider SEGPU et MEGPU. Premièrement, on évalue les performances des deux algorithmes parallèles en les comparant avec la version séquentielle de BSO-ARM en terme du temps. De plus, on démontre que SEGPU et MEGPU donnent aussi des solution acceptables comme l'algorithme BSO-ARM. Deuxièment, on analyse l'effet du nombre d'abeilles, le nombre d'items et le nombre de transactions sur les performances de SEGPU et MEGPU. Puis, on évalue quelques challenges de l'architecture GPU dans les algorithmes SEGPU et MEGPU.

- **Les algorithmes parallèles Vs L'algorithme BSO-ARM :**
  On compare les deux algorithmes (SEGPU et MEGPU) avec l'algorithme BSO-ARM en terme du temps d'exécution et la qualité des solutions trouvées. Les expérimentations sont faites sur des données réelles avec différentes tailles comme il a été mentionné dans 4.2.7.2.

La performance des algorithmes parallèles en terme du temps

d'exécution

La Table4.11 présente le temps d'exécution des algorithmes parallèles

Type de la base des transactions	Nom de la base des transactions	BSO-ARM	SEGPU	MEGPU
Petite	Bolts	9	20	1.5
	Sleep	15	32	3
	Pollution	22	48	4
	Basket ball	35	20	4
	IBM Q.S.	618	545	10
	Quake	80	495	8
	Chess	149956	845	95
	Mushroom	28815	1064	150
Moyenne	Pumbs_star	144120	2475	550
	BMS-W.V.-1	180524	3487	620
	BMS-W.V.-2	249985	4177	850
	Korasak	258451	4525	948
	Retail	299658	4910	1185
	Connect	300985	5815	1485
Large	BMP POS	bloqué aprés 15 jours	398526	12565
	WebDocs	bloqué aprés 15 jours	785652	35965

TABLE 4.11 – Le temps d'exécution des algorithmes parallèles proposés avec différentes bases de transactions(en Sec)

(SEGPU, MEGPU) et la version séquentiels de l'algorithme BSO-ARM. Afin de bien explorer l'espace de recherche, le nombre d'abeilles K est fixé à 20, et le nombre d'itérations $IMAX$ est fixé à 100. Les résultats montrent qu'il y'a une lagrge différence entre le temps d'exécution des deux algorithmes parallèles. En effet, La performance de MEGPU en terme de temps d'exécution est meilleure que l'algorithme SEGPU dans toutes les bases des transactions et surtout dans les bases de transactions de trés grande taille. Pour les bases de données de petite taille, SEGPU présente des mauvais résulats en comparant avec la version séquentielle de l'algorithme BSO-ARM. On peut expliquer celà par le fait que l'évaluation d'une seule solution en CPU nécessite un temps d'exécution moins long que celui de la commu-

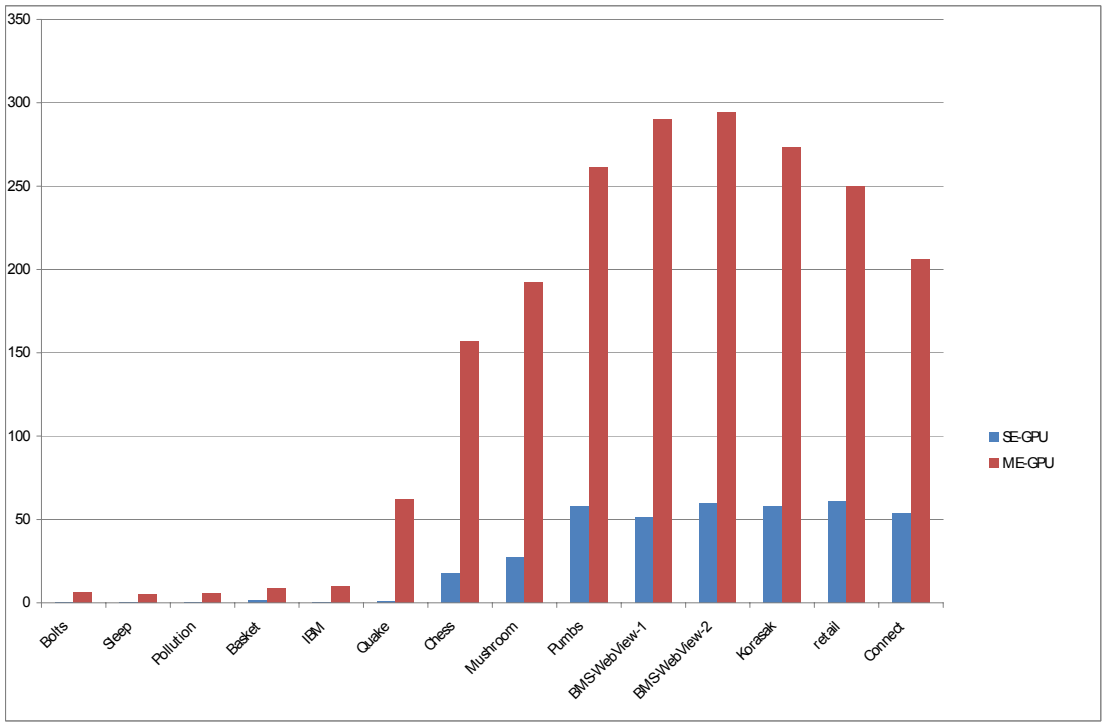


FIGURE 4.7 – L'accélération de SEGPU et MEGPU

nication CPU/GPU. De plus, le temps d'exécution dépend non seulement du nombre de transactions mais aussi du nombre d'items qui composent chaque transaction. A titre d'exemple, malgré que le nombre de transactions de Quack qui dépasse celui de IBMQuest, le temps d'exécution de ce dernier est plus supérieur qu'en Quack. celà est dû au fait que le nombre d'items qui composent IBMQuest est plus large que Quack.

Les deux versions parallèles de BSO-ARM permettent de traiter les deux grandes bases de transactions (BMP POS et Web Docs) contenant plus que 1.5 millions de transactions et plus que 0.5 million d'items. D'aprés nos connaisances, ces deux bases de transactions n'ont pas été déja résolues dans la litérature des algorithmes d'extraction des règles d'association. En effet, l'algorithme BSO-ARM est bloqué aprés 12 jours alors qu'il faut 10 jours avec SEGPU et juste 10 heures avec MEGPU afin de traiter la base de transactions WebDocs.

La Figure4.7 présente l'accélération de ces deux algorithmes (SEGPU et

MEGPU) sur différentes bases de transactions. On remarque qu'il existe une large différence entre les deux algorithmes en utilisant des transactions condensées (le nombre de d'items est grand) et une petite différence en utilisant des transactions non condensées. Ces résultats confirment l'expérimentation précédente. De plus, l'accélération en MEGPU dépasse 290. Cependant, l'accéleration en SEGPU ne dépasse pas 100. On peut dire aprés cette expérimentation que si on a une large base de transactions non condensée alors SEGPU est plus approprié avec un nombre d'abeilles trés réduit. Sinon, si on a une large base de transactions condensée alors MEGPU est plus approprié en utilisant un nombre imporant d'abeilles.

La performance des algorithmes parallèles en terme de qualité des solution trouvées

Les expérimentations précédentes révèlent que SEGPU et MEGPU surpassent la version séquentielle de BSO-ARM en terme du temps d'exécution. Cependant, les méthodes d'optmisations ne donnent pas des solutions exactes. En effet, la question qui se pose dans ce contexte est comment la parallélisation influence à la qualité des solutions trouvées. Pour celà, plusieurs expérimentations sont faites afin d'évaluer la qualité des solutions trouvées. D'aprés la Table4.12, on observe une légère différence entre les algorithmes proposés. En effet, dans certains cas BSO-ARM trouve des meilleures solutions par rapport au SEGPU et MEGPU, d'autres cas, les deux algorithmes parallèles trouvent des meilleures solutions par rapport au BSO-ARM. Cette différence est réalisée car la solution initiale est choisie aléatoirement. Encore plus loin, dans WebDocs, SEGPU et MEGPU trouvent des solutions avec des qualités de 0.01 et 0.02 respéctivement ce qui est impossible en utilisant la version séquentielle de l'algorithme BSO-ARM.

– **SEGPU Vs MEGPU**

Le but de cette expérimentation est de déterminer quels sont les cas où SEGPU est meilleure que MEGPU et vise versa. Pour celà, on varie le nombre d'abeilles K, le nombre d'items n et le nombre de transactions m

Nom de la base de transactions	BSO-ARM	SE-GPU	MEGPU
Bolts	1.0	1.0	1.0
Sleep	1.0	1.0	1.0
Basket ball	0.97,	0.96	0.98
IBM Quest Standard	0.95	0.94	0.92
Quake	1.0	1.0	0.99
Chess	0.92	0.88	0.92
Mushroom	0.60	0.62	0.58
Pumbs_star	0.40	0.39	0.37
BMS-WebView-1	0.35	0.34	0.38
BMS-WebView-2	0.55	0.49	0.52
Korasak	0.35	0.37	0.39
retail	0.36	0.33	0.31
Connect	0.26	0.29	0.25
BMP POS	0.20	0.18	0.17
WebDocs	Null	0.02	0.01

TABLE 4.12 – Evaluation de la qualité des solutions des algorithmes proposés

et on regarde à chaque fois le comportement des deux algorithmes SEGPU et MEGPU en terme de temps d'exécution. Les transactions sont générées aléatoirement et le nombre d'itérations est fixé à une seule itération.
Les Figures4.8, 4.9 et4.10 montrent le rapport entre le temps d'exécution de MEGPU sur le temps d'exécution de SEGPU en augmentant le nombre d'items (de 10 à 100 items pour 1000 transactions, de 100 à 1000 items pour 10000 transactions, et de 1000 à 10000 items pour 100000 transactions) et en variant le nombre d'abeilles de 5 à 15 abeilles. Ces figures révèlent qu'en augmentant le nombre d'abeilles, MEGPU travaille mieux par rapport à SEGPU. On peut dire aussi que le nombre d'items et le nombre de transactions influencent positivement sur la performance de MEGPU en comparant avec la performance de SEGPU. En effet, à partir du nombre d'items qui est égal à 60 pour que MEGPU surpasse SEGPU avec une base contenant 1000 transactions. Cependant, avec 100000 transactions MEGPU surpasse SEGPU pour n'importe quel nombre d'items. Celà est dû par le fait que le calcul intensif en SEGPU est plus efficace que MEGPU car dans SEGPU, tous les blocs libres de GPU sont exploités pour évaluer une seule solu-

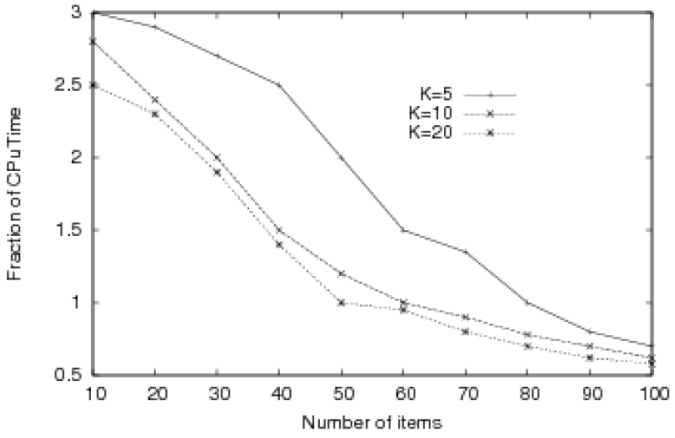


FIGURE 4.8 – Le rapport entre SEGPU et MEGPU en terme du temps d'exécution utilisant 1000 transactions

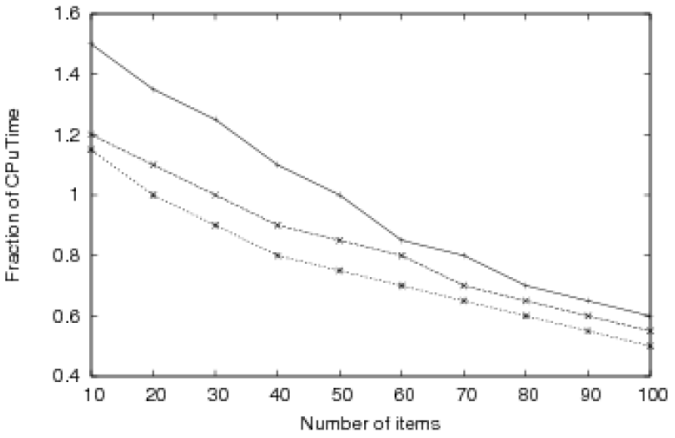


FIGURE 4.9 – Le rapport entre SEGPU et MEGPU en terme du temps d'exécution utilisant 10000 transactions

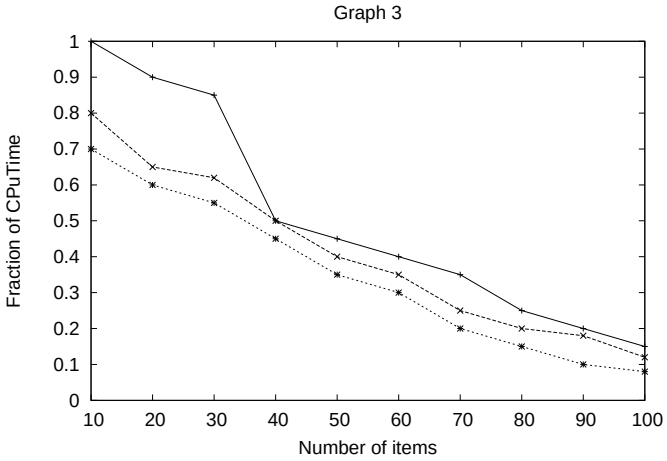


FIGURE 4.10 – Le rapport entre SEGPU et MEGPU en terme du temps d'exécution utilisant 100000 transactions

tion. Tandis que, dans MEGPU, un seul bloc est aloué pour évaluer chaque solution. A chaque fois que l'espace de recherche est grand, le nombre de CPU/GPU est important. Par conséquent, le temps d'exécution augmente considérablement. Cependant, la puissance de MEGPU réside dans la coût des communications CPU/GPU qui nécessite seulement un seul accés pour chaque itération et pour n'importe quel nombre de solutions générées. A partir des expérimentations, on peut conclure que SEGPU est plus efficace de MEGPU dans des bases de transactions non condensées. En contrepartie, MEGPU est plus efficace que SEGPU quand on traite des bases de transactions condensées.

– **Gestion des accés mémoires :**
Dans SEGPU, les ((l+1)*n) octets sont alloués dans la mémoire partagée et dans MEGPU, les (m*(n+1)) octets doivent être aloués dans la mémoire partagée. Quand on traite des bases de transactions de petite taille, il est évident que MEGPU nécessite un nombre d'accés mémoire trés réduit par rapport au SEGPU parceque dans MEGPU deux blocs distincts traitent deux solutions différentes alors que dans SEGPU tous les blocs traitent une seule solution en même temps. De plus, si le nombre de transactions est trés grand et le nombre de blocs disponible sur GPU est petit, alors MEGPU demande beaucoup d'accés en mémoire globale que SEGPU. Les

deux algorithmes ont été expérimentés en utilisant les bases de transactions réelles déja expliquées avec un nombre d'abeilles égal à 5 et *IMAX* égal à 1. Pour chaque exécution, on calcule le nombre d'accés à la mémoire globale. Les résultats sont présentés dans la Table4.13.

Type de la base des transactions	Nom de la base de transactions	SEGPU	MEGPU
Petite	Bolts	40	1
	Sleep	40	1
	Pollution	80	1
	Basket ball	25	1
	IBM Quest Std.	200	796
	Quake	20	1974
	Chess	375	2992
	Mushroom	595	7919
Moyenne	Pumbs_star	73991	40283
	BMS-WebView-1	61063	59500
	BMS-WebView-2	93188	77410
	Korasak	114825	80667
	retail	87383	88060
	Connect	103971	99898
Large	BMP POS	522858	515495
	WebDocs	4325595	1692031

TABLE 4.13 – Nombre d'accés au mémoire globale de SEGPU et MEGPU

D'aprés cette table, on peut dire que le nombre d'accés à la mémoire globale dépend du nombre d'items et du nombre de transactions, ce qui confirme les résultas obtenus dans les expérimentations précédentes.

- **Les communications CPU/GPU** Dans SEGPU, il y'a ((IMAX*K*n)+1) CPU/GPU communications. Alors que, avec MEGPU, seulement (IMAX+1) communications sont effectuées. Lorsque le nombre d'items est trop grand, SEGPU sera pénalisé par un nombre important de communication CPU/GPU. Aussi, le nombre d'abeilles doit être petit concernant SEGPU. Cependant, en utilisant MEGPU, on peut utiliser un nombre important d'abeilles. Cette contrainte nous permet de bien exploiter l'espace de recherche par MEGPU que par SEGPU. Pour prouver celà, la Figure4.11 présente le

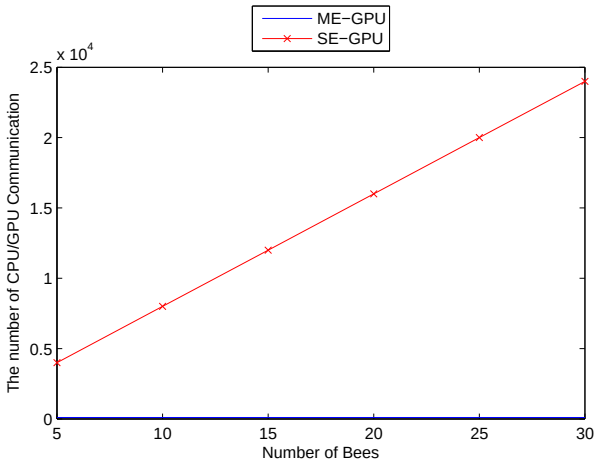


FIGURE 4.11 – Le nombre de communications CPU/GPU de SEGPU et MEGPU

nombre de communications CPU/GPU avec Bolt. Ici, IMAX est fixé à 100 itérations. D'aprés cette figure, on peut dire que tout en augmentant le nombre d'abeilles, le nombre de communications CPU/GPU stabilise à 101. Alors que le nombre de communications CPU/GPU de SEGPU augmente tout en augmentant le nombre d'abeilles de 10 à 30 .

4.4 Conclusion

Dans ce chapitre, on a réussi à développer deux approches séquentielles (BSO-ARM et HBSO-ARM)d'extraction des règles d'association basées sur la métaheuristique BSO. Dans BSO-ARM, la détermination des régions de chaque abeille se fait en utilisant une simple recherche locale, alors que pour HBSO-ARM, la métaheuristique recherche tabou a été employée pour que chaque abeille bien explore sa région de recherche. Les deux algorithmes améliorent la qualité des solutions ainsi que le temps d'execution de l'état de l'art des algorithmes d'extraction des règles d'association. Cependant, ces algorithmes se bloquent en traitant un nombre important de transactions. Pour celà, on a aussi proposé dans ce chapitre deux versions parallèles de l'algorithme BSO-ARM qui sont implémentées en utilsant la platforme GPU. Les deux algorithmes parallèles exploitent le cal-

cul intensif de GPU pour évaluer les solutions générées dans CPU. Le premier algorithme traite solution par solution en GPU, alors que le deuxième algorithme traite un ensemble de solutions à la fois en GPU. Avec ces versions parallèles, on a pu traiter la base de transactions WebDocs (qui contient plus d'un million de transactions) ce qui est impossible dans le cas des algorithmes séquentiels.

Chapitre 5

Amélioration des algorithmes de fouille et de réduction des règles d'association

5.1 Introduction

Dans plusieurs applications, les algorithmes d'extraction des règles d'association donnent généralement un très grand nombre de règles qui peuvent atteindre plusieurs millions. Leurs exploitation par l'utilisateur est devenue difficile et parfois impossible. Par conséquent, la suppression des règles non pertinentes est devenue une tâche nécessaire et trés utile. Plusieurs algorithmes de réduction des règles ont été détaillés dans le deuxième chapitre. il y a quelques algorithmes qui introduisent les connaissances de l'utilisateur, ces algorithmes ne peuvent pas être appliqués dans des domaines où l'utilisateur ne peut pas y intervenir, comme dans le cas de la recherche d'informations. D'autres algorithmes sont basés sur l'extraction des connaissances automatique sans utiliser l'expérience de l'utilisateur. Ces derniers algoirthmes considèrent un autre espace appelé espace des règles qui contient toutes les règles possibles, ensuite ils appliquent un autre processus de fouille à partir de cet espace. Cette fouille permet de déduire d'autres connaissances supplémentaires sur l'espace des règles qui seront utiles dans le processus de réduction des règles d'association.

Cependant, ces algorithmes d'extraction présentent quelques inconvenients comme le nombre des meta-règles générées qui est trés limité ainsi que le temps d'exécution considérable causé par l'application des méthodes exactes à ce nouveau espace des règles.

Pour celà, comme deuxième contribution de cette thèse est l'exploration rapide de l'espace des règles en utilisant les différentes méthodes approchées qui ont été proposées dans le chapitre précédent. Ainsi, une nouvelle approche de réduction des règles sera proposée en utilisant les connsaisances (meta-règles) trouvées par le prcessus de fouille de l'espace des règles. Cette nouvelle approche de réduction permet de garder les règles représentatives de l'espace des règles et supprimer les autres règles qui peuvent être déduisent à partir des règles représentatives.

5.2 L'algorithme Kmeans-AR

En raison de la grande quantité des règles générées par le processus d'extraction des règles d'association, il est difficile de les utiliser en même temps. Afin d'organiser cette grande quantité de règles, l'espace de ces règles est divisé en plusieurs ensembles de règles similaires. L'idée est d'appliquer le processus de classification non superivisée de ces règles. Afin d'obtenir des ensembles totalement indépendants, l'algorithme Kmeans [**?**] est appliqué. Il est l'algoirthme le plus utilisé parmis les algorithmes de classification non supervisée par partionnement. L'algorithme Kmeans est principalement composé de deux fonctions : La mesure de similarité entre deux objets et le calcul de centre de gravité d'un ensemble d'objets. Dans la litérature, plusieurs mesures de similarité et des formules de calcul des centres de gravité ont été développées pour des données simples [**?**] ou pour des règles d'inductions [63]. Pour l'étape suivante, nous proposons une adaptation pour les règles d'association de plusieurs mesures et formules de calcul des centres de gravaité qui ont déjà été développées.

5.2.1 Les opérations mathématiques de la classification non superivsée des règles d'association

Comme nous nous intéressant à la classification non supervisée des règles d'association et Kmeans est l'un des algorithmes les plus simples de cette tâche [13], nous proposons une adaptation de Kmeans à ce genre de représentation des connaissances. Les concepts fondamentaux de l'algorithme, qui sont la mesure de similarité et le calcul des centres de gravité doivent être bien définis. Kmeans généralement traite des données simples tandis que dans notre situation, il s'agit des objets complexes tel que les règles d'association. Le challenge donc est de bien déterminer, une mesure de similarité entre deux règles d'association et de bien définir une formule pour le calcul de centre de gravité d'un ensemble de règles d'association.

Mesure de similarité

La similarité entre deux règles mesure le degré de ressemblance entre elles. Dans le cas contraire, leurs dissemblance signifie le degré de disparité entre ces règles. Intuitivement, elles sont similaires quand elles partagent un grand nombre d'items. Dans la littérature, il existe plusieurs mesures de distances, certains d'entre eux, sont destinéess à n'importe quel type d'objets, d'autres sont développées spécialement pour les règles d'association. Cependant, on va étendre les mesures les plus utilisées dans la litérature pour les règles d'association.

On assume deux règles d'association r_1 et r_2 représentées par deux vecteurs de n éléments qui appartiennent à l'ensemble $\{0, 1, 2\}$. Le i^{me} élément de la règle r est égal à 1 respéctivement 2 si le i^{me} item appartient à la partie antécédente réspectivment la partie conséquente de r, il est égal à 0, sinon. La distance entre r_1 et r_2 est calculée comme :

- La distance de Manhattan

$$d(r_1, r_2) = \sum_{i=1}^{n} |r_1[i] - r_2[i]|. \tag{5.1}$$

- La distance de Chebyshev

$$d(r_1, r_2) = \max_i |r_1[i] - r_2[i]|. \tag{5.2}$$

– La distance d'Euclidian

$$d(r_1, r_2) = \sqrt{\sum_{i=1}^{n} (r_1[i] - r_2[i])^2}. \tag{5.3}$$

– La distance d'Euclidian_squared

$$d(r_1, r_2) = \sum_{i=1}^{n} (r_1[i] - r_2[i])^2. \tag{5.4}$$

– La distance Dist_items

$$d(r_1, r_2) = Total\_Nbr\_items(r_1, r_2) - Common\_Nbr\_items(r_1, r_2). \tag{5.5}$$

tel que
Total_Nbr_items(r_1,r_2)=$\sum_{i=1}^{n} TNI[i]$.
Common_Nbr_items(r_1,r_2)=$\sum_{i=1}^{n} CNI[i]$.
TNI[i]=1 si r_1[i]$\neq$0 ou $r_2\neq$0 sinon TNI[i]=0.
CNI[i]=1 si r_1[i]$\neq$0 et $r_2\neq$0 sinon CNI[i]=0.

– La distance Ration

$$d(r_1, r_2) = \frac{|Items_1 \bigcup Items_2| - |Items_1 \bigcap Items_2|}{|Items_1| + |Items_2|}. \tag{5.6}$$

tel que
$Items_i$: représente l'ensemble d'items de r_i.

Exemple Considérons I=$\{i_1, i_2, i_3, i_4\}$ et deux règles r_1 et r_2 :
r_1 : i_1, $i_2 \Rightarrow i_3$.
r_2 : i_2, $i_4 \Rightarrow i_3$.
r_1=$\{1,1,2,0\}$ et r_2=$\{0,1,2,1\}$.

En utilisant les différentes mesures de similarité déjà expliquées auparavant, la similarité entre r_1 et r_2 est calculée comme suit :

- Manhattan(r_1,r_2)=|1-0|+|1-1|+|2-2|+|0-1|=2.
- Chebyshev(r_1,r_2)=max(1,0,0,1)=1.
- Euclidian(r_1,r_2)=$\sqrt{(1-0)^2+(1-1)^2+(2-2)^2+(0-1)^2}=\sqrt{2}$.
- Euclidian_squared(r_1,r_2)=$(1-0)^2+(1-1)^2+(2-2)^2+(0-1)^2=2$.
- En utilisant Eq(5), $TNI=\{1,1,1,1\}$ et $CNI=\{0,1,1,0\}$. alors, Dist_items(r_1,r_2)=(1+1+1+1)-(0+1+1+0)=2.
- en utilisant Eq(6), $Items_1$=$\{i_1,i_2,i_3\}$ et $Items_2$=$\{i_2,i_3,i_4\}$ alors, Ration(r_1,r_2)=$\frac{(4-2)}{3+3}=\frac{1}{3}$.

Calcul des centres de gravité

La détermination de la règle moyenne d'un ensemble de règles dépend de la mesure de distance utilisée. Afin d'étendre Kmeans à Kmeans-AR, on a besoin d'une formule qui calcule le centre de gravité d'un ensemble de règles d'association. Les formules classiques de calcul de centre de gravité ne peuvent pas être appliquées ici car Kmeans manipule des données et non pas des règles d'association. Pour cette raison, plusieurs formules sont étendues aux règles d'association afin de bien s'approcher au centre exact d'un ensemble de règles d'association.

Etant données m règles d'association $\{r_1, r_2, r_3..., r_m\}$ et règle-centre est la règle moyenne de ces règles.

1. CC (Classical Centers) $rgle-centre[i]=\frac{\sum_{j=1}^{m} r_j[i]}{m} \forall i \in [1..n]$.
2. FC (Frequent Centers) Dans un premier temps, la taille de la règle-centre notée l est calculée. Elle est égale à la moyenne des nombres d'items des m règles et elle est déterminée comme :

$$l=\frac{\sum_{i=1}^{m}|items(r_i)|}{m}$$

Ensuite, les items de m règles sont triés en fonction de leurs fréquences dans les m règles et seulement les l items sont insérés au vecteur appellé $Freq$ tel que :

$rgle-centre[Freq[j]]=1$ et $\forall$ i $\neq$ Freq[j] règle-centre[i]=0.

3. La formule UOI (Union Over Intersection) : Consiste de calculer l'union et l'intersection de tous les items des m règles. Puis, on soustrait le résultat de l'intersection de l'union.
   $items(rgle{-}centre) = [items(r_1) \cup items(r_2) \cup \ldots \cup items(r_m)] - [items(r_1) \cap items(r_2) \cap \ldots \cap items(r_m)]$
4. La formule IUC (Intersection Union Center) : Dans ce cas, on calcule l'intersections de tous les items de m règles, puis on fait l'union entre le résultat de l'intersection et la règle-centre de l'itération courante comme :
   $items(rgle - centre) = [items(r_1) \cap items(r_2) \cap \ldots \cap items(r_m)] \cup items(g_i)$
5. La formule UIC (Union Intersection Center) : On calcul l'union de tous les items ensuite on fait l'intersection entre le résultat obtenu et la règle-centre de l'itération courante comme :
   $items(rgle - centre) = [items(r_1) \cup \ldots \cup items(r_m)] \cap items(g_i)$.

Exemple :

- La règle-centre est initialement vide.
- l'ensemble d'items est $\{i_1, i_2, i_3\}$,

Afin de calculer le centre de gravité des trois règles suivantes : r_1 (i_1, $i_2 \rightarrow i_3$), r_2 ($i_1 \rightarrow i_3$), r_3 ($i_2 \rightarrow i_3$), chaque règle r_i est représentée en fonction de la partie antécédente et la partie conséquente comme r_1={1,1,2}, r_2={1,0,2}, r_3={0,1,2}.

Aussi, l'ensemble d'items de chaque règle peut être défini comme : items(r_1)={i_1, i_2, i_3}, items(r_2)={i_1, i_3}, items(r_3)={i_2, i_3}.

La règle-centre des trois règles r_1, r_2 et r_3 est calculée :

- $CC(rgle - centre) = \{\frac{(1+1+0)}{3}, \frac{(1+0+1)}{3}, \frac{(2+2+2)}{3}\} = \{\frac{2}{3}, \frac{2}{3}, 2\}$.
- Pour calculer la formule FC(règle-centre), on défini l tel que : $l = \frac{3+2+2}{3} = 2$. Puis, les l items fréquents sont séléctionés. Dans cet exemple, $\{r_1, r_3\}$ sont des items fréquents. Donc, $FC(rgle - centre) = \{1, 0, 1\}$.
- $UOI(rgle - centre) = [items(r_1) \cup items(r_2) \cup items(r_3)] - [items(r_1) \cap items(r_2) \cap items(r_3)] = \{i_1, i_2\}$.
- $IUC(rgle{-}centre) = [items(r_1) \cap items(r_2) \cap items(r_3)] \cup items(g_i) = \{i_3\}$.
- $UIC(rgle - centre) = [items(r_1) \cup items(r_2) \cup items(r_3)] \cap items(g_i) = \emptyset$.

5.2.2 Déscription détaillée de l'algorithme Kmeans-AR

Dans [110], nous avons proposé l'algorithme *Kmeans-AR* qui est une extension de l'algorithme Kmeans pour les règles d'association. Avant de détailler sur l'algorithme proposé, on fait un petit rappel sur le principe de Kmeans. Kmeans est l'un des algorithmes les plus simples qui résout le problème de classification non supervisée. Le but est de grouper un ensemble de données dans K clusters fixés à priori. L'idée principale consiste de définir K centres, un pour chaque cluster. Les centres doivent être placés intelligemment car le résultat de classification dépend de la location intitale de ces centres. Afin d'optimiser l'efficacité de résultat de Kmeans, il est judicieux de placer les centres autant éloignés que possible des uns aux autres. La prochaine étape est de prendre chaque point de l'ensemble de données et de le placer avec le centre le plus proche. Quand tous les points ont été traités, la première étape est terminée et le premier groupement est réalisé. A ce moment, on recalcule les K nouveaux centres en utilisant une formule de calcul des centres de gravité. Ce processus doit être répété jusqu'à où aucun point ne change de cluster. L'algorithme de Kmeans-AR suit le même principe de l'algorithme Kmeans, sauf dans le calcul de similarité et les centres de gravité, il utilise les formules déja expliquées ci-dessus.

L'algorithme Kmeans-AR

```
Debut
 Initialser un centre pour chaque cluster
    Pour chaque règle r faire
       affecter r au group i qui a le centre le plus proche
       Si (aucune règle ne change de cluster) alors
         terminer et sortir
         Sinon
          Calculer le nouveau center pour chaque cluster
       fin_si
    fin_pour
Fin
```

Distance	r_1	r_2	r_3	r_4	r_5
G_1	0	1	3	3	1
G_2	1	0	4	2	1
G_3	3	4	0	2	4

TABLE 5.1 – Les distances entre les centres et toutes les règles pour la première itération

Comme dans l'algorithme Kmeans, la complexité de l'algorithme Kmeans-AR est O($K \times n \times NBR \times Nb-iter$), tel que K est le nombre des clusters, NBR est le nombre de règles, n est le nombre d'items et $Nb-iter$ est le nombre d'itérations de l'algorithme Kmeans-AR.

5.2.3 Illustration de l'algorithme Kmeans-AR

Considérons l'ensemble d'items I=$\{i_1, i_2, i_3, i_4, i_5\}$ et les règles générées suivantes :

- $r1 : i_1, i_2 \Rightarrow i_3$.
- $r2 : i_2, i_3 \Rightarrow i_1, i_4$.
- $r3 : i_2 \Rightarrow i_5$.
- $r4 : i_1 \Rightarrow i_4$.
- $r5 : i_1, i_2, i_3 \Rightarrow i_4$.

Pour K=3, et en utilisant la distance Ratio et la formule UOI pour le calcul des centres de gravité, on obtient le scénario suivant :
Initiallement :
$G_1 = r_1, G_2 = r_2, G_3 = r_3$ et $C_1 = \{r_1\}, C_2 = \{r_2\}, C_3 = \{r_3\}$.
La Table 5.2.3 présente les distances de la première itération entre les centres et toutes les règles. Ensuite, les premières clusters sont obtenus
$C_1 = \{r_1, r_5\}$, $C_2 = \{r_2\}$, $C_3 = \{r_3, r_4, r_5\}$.
Puis, Les centres de gravité sont calculés pour les nouveaux clusters. Le résultat final de l'algorithme est : $C_1 = \{r_1, r_5, r_2\}$, $C_2 = \{r_4\}$, $C_3 = \{r_3\}$.

5.3 L'algorithme d'extraction des meta-règles

Le processus d'extraction des règles d'associations délivre un nombre important de règles. Il est le temps de traiter ces règles en trouvant des structures cachées et des relations entre ces règles. Comme on a vu dans le deuxième chapitre, les meta-règles sont extraitent à partir d'un ensemble de règles représentées sous forme de transactions [65]. Les incovenient principales de ce travail sont : D'une part, le nombre des règles générées est plus grand que le nombre de transactions, ceci provoque un temps d'exécution considérable avec l'utilisation de l'algorithme Apriori. Dans ce cas, il est plus approprié d'appliquer les méthodes approchées que les méthodes exactes. D'autre part, la découverte des meta-règles, est basée seulement sur l'extraction des règles de taille 2, c'est à dire une règle de la partie antécédente et une autre règle de la partie conséquente. Cette contrainte affaiblit la structure des meta-règles et provoque la perte des connaissances qui seront utiles par la suite dans la réduction des règles d'association.

Dans cette section, on propose une nouvelle approche de découverte de meta-règles permettant l'extraction des meta-règles les plus pertinentes parmi toutes les meta-règles. Pour bien réaliser cette approche, les algorithmes déja proposés du quatrième chapitre seront utilisés. Par conséquent, différentes extensions de ces algorithmes peuvent être considérées.

Avant d'appliquer le processus d'extraction des meta-règles, les règles d'association ont besoin d'être transformées à la base de transactions. Pour cette raison, on propose dans un premier temps un processus de transfomration qui permet de transformer n'importe qu'elle règle en forme transactionale, la prochaine section détaillera le proccesus de transformation.

5.3.1 Transformation des règles d'association en forme transactionale

Contrairement aux règles d'association dont le but est d'extraire des relations entre les données, le but de l'extraction des meta-règles est de trouver les différentes relations entre les règles. Alors, cette extraction est située à un niveau plus élévé d'abstraction. Afin de découvrir les meta-règles, une transformation des

règles d'association en forme transactionale est nécessaire. La transformation est faite en fonction de la partie conséquente de la règle. Chaque règle est considérée comme un item et les règles qui partagent la même partie conséquente appartiennent à la même transaction. Par conséquence, le nombre transactions est égal au nombre de conséquences possibles.

On considère $AR = \{r_1, r_2, ..., r_n\}$ un ensemble de règles d'association, $I = \{I_1, I_2, ..., I_n\}$ un ensemble d'items et $T = \{T_1, T_2, ..., T_m\}$ la forme transactionale des règles d'association. La transformation peut être formulée comme :

$$\begin{cases} \text{I=AR} \\ r = x \Rightarrow\ y\ \in\ AR, \Rightarrow\ y\ \in\ T. \end{cases} \tag{5.7}$$

L'algorithme de transformation des règles en forme transactionale

```
Debut
 Initialser T à vide
    Pour chaque règle r: X=>Y dans AR faire
      existe=faux
      Pour chaque transaction t dans T et existe=faux faire
        Si (Y=t) alors
          exist=vrai
          insérer r dans t
        fin_si
      fin_pour
      Si (exist=faux) alors
          créer une nouvelle  entrée dans T nommée par Y
      fin_si
    fin_pour
Fin
```

Exemple Considérons I l'ensemble d'items $\{i_1, i_2, i_3, i_4\}$ et AR les règles d'association suivantes :

$r_1\ i_1, i_2 \Rightarrow i_3$

$r_2\ i_4 \Rightarrow i_3$

$r_3\ i_1 \Rightarrow i_2$

$r_4\ i_4 \Rightarrow i_1$
$r_5\ i_4 \Rightarrow i_1, i_2, i_3$
$r_6\ i_4 \Rightarrow i_2, i_3$

Donc on aura sept parties conséquentes (i_1), (i_2), (i_3), (i_1, i_2), (i_1, i_3), (i_2, i_3), (i_1, i_2, i_3). Alors, on obtient septs différentes transactions $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$. Tel que :
t_1 : r_4 , r_5 car r_4 et r_5 ont la même partie conséquente (i_1).
t_2 : r_3, r_5, r_6 car r_3, r_5 et r_6 ont la même partie conséquente (i_2).
t_3 : r_1, r_2, r_5, r_6 car r_1, r_2 ,r_5 et r_6 ont la même partie conséquente (i_3).
De même pour les transactions de t_4 à t_7.

5.3.2 Principe de l'algorithme Meta-règles

Aprés la transformation des règles d'association en forme transactionale, l'étape d'extraction des meta-règles est réalisée en appliquant un algorithme d'extraction des règles d'association sur la base de transactions T déja trouvées par le processus de transformation.

A cause de la grande taille de T, les méthodes exactes prennent un temps considérable. Afin de résoudre cet inconvenient, une extension des algorithmes d'extraction des règles déja proposées du quatrième chapitre est effectuée. Pour celà, quatres algorithmes peuvent être considérés. Les deux algorithmes séquentiels (*BSO-MR* et *HBSO-MR*), ils sont l'extension de BSO-ARM et HBSO-ARM réspectivement. Deux autres algorithmes parallèles peuvent être étendus appelés réspectvement (SEGPU-MR qui est l'extension de l'algorithme SEGPU, et MEGPU-MR qui est l'extension de l'algorithme MEGPU). Chaque solution S de ces quatres algorithmes représente une meta-règle notée mr qui est un vecteur de NBR éléments tel que NBR est le nombre de toutes les règles possibles. La solution S peut être définie comme suit :

1. $S[i] = 0$ si la règle r_i n'appartient pas à la meta-règle mr.
2. $S[i] = 1$ si la règle r_i appartient à la partie antécédente de la meta-règle mr.

3. $S[i] = 2$ si la règle r_i appartient à la partie conséquente de la meta-règles mr.

Aprés la représentation de la solution, on applique le même processus des algorithmes proposés dans le quatriéme chapitre.

5.4 La réduction des règles d'association

Le règles trouvées par le processus d'extraction des règles d'association sont inutiles par l'utilisateur. La réduction du nombre des règles est devenue une challenge pour la communauté d'extraction des règles d'associations. En se basant sur les algorithmes d'extraction des meta-règles déja dévéloppés dans la Section 5.3, on propose dans cette section, une nouvelle approche de réduction des règles d'association.

5.4.1 Formulation du problème de réduction des règles

Définition1 : Considérons l'espace des règles RS qui représente l'ensemble de règles, on défini la mesure Relevant comme une application de RS à l'ensemble des nombres réels R. Cette fonction permet de mesurer la qualité d'une règle donnée r et peut être calculée grâce à la formule suivante :
Relevant(r)=α Lift(r)+β Leverage(r)+ γ Conviction(r).
tel que
α, β $\gamma \in [0-1]^3$.

Le support et la confiance de la règle sont insufissants pour évaluer une règle donnée. Pour celà, d'autres mesures sont nécessaires comme lift, leverage, et conviction [4]. Toutes ces mesures dépendent du support et de la confiance de la règle. Afin de bien évaluer les règles obtenues, la fonction Relevant combine toutes ces mesures avec les poids α, β, et γ.

Définition2 : On définit la mesure Distinct comme une application de RS^{NBR} à R :
Distinct($r_1, r_2, ..., r_{NBR}$)=$\sum_{i=1}^{NBR} \sum_{j=1}^{NBR} D(r_i, r_j)$.
Tel que

D(r_i,r_j) : mesure la distance entre les règles r_i et r_j qui est développée dans [91].

Afin de trouver les règles pertinentes qui décrivent l'espace des règles, le problème de réduction des règles peut être vu comme suit :
Trouver un sous ensemble de l règles $\{r_1, r_2, r_3, ..., r_l\}$ noté $R_{sub} \subseteq RS$ qui maximise la fonction Distinct des règles inclusent dans R_{sub} et maximise aussi la fonction Relevant de chaque règle dans R_{sub}. En d'autres termes, le sous ensemble qui maximise la fonction f tel que :

$$f_{max}(R_{sub}) = \frac{Distinct(R_{sub}) + \sum_{i=1}^{l} Relevant(r_i)}{l}. \tag{5.8}$$

5.4.2 L'approche de réduction des règles d'association

Propriétée de réduction des règles : On considère la meta-règle mr : $X \Rightarrow Y$ où X et Y qui contiennent un ensemble de règles. Si la confiance de mr est 100% alors on peut dire que Y forme la partie des règles représentatives à partir de mr et X forme la partie des règles déductives à partir de mr.
preuve
On a $mr : X \Rightarrow Y$, la confiance de mr est 100%, celà signifie que si l'ensemble des règles de la partie antécédente apparaît alors il est sûr que les règles de la partie conséquente apparaît. De plus, les parties conséquentes des règles composant X sont incluses dans les parties conséquentes des règles qui composent Y. En outre, on peut garder les règles composant Y et on supprime les règles qui composent X. En d'autres termes, Y peut être appelé l'ensemble des règles représentatives à partir de mr et X est l'ensemble des règles déductives à partir de mr.

Le but de l'approche de réduction des règles est de réduire l'espace des règles d'association, c'est à dire garder juste les règles représentatives de cet espace. Les différentes relations et dépendances trouvées par l'approche de meta-règles permettent l'application de la propriétée de réduction des règles d'association de telle sorte que pour chaque meta-règle forte (a une confiance de 100%), on garde l'ensemble des règles de sa partie conséquente et on supprime les règles de sa partie antécédente. La Figure 5.1 décrit clairement l'approche proposée.

le problème qui se pose dans cette approche est la complexité de l'étape de

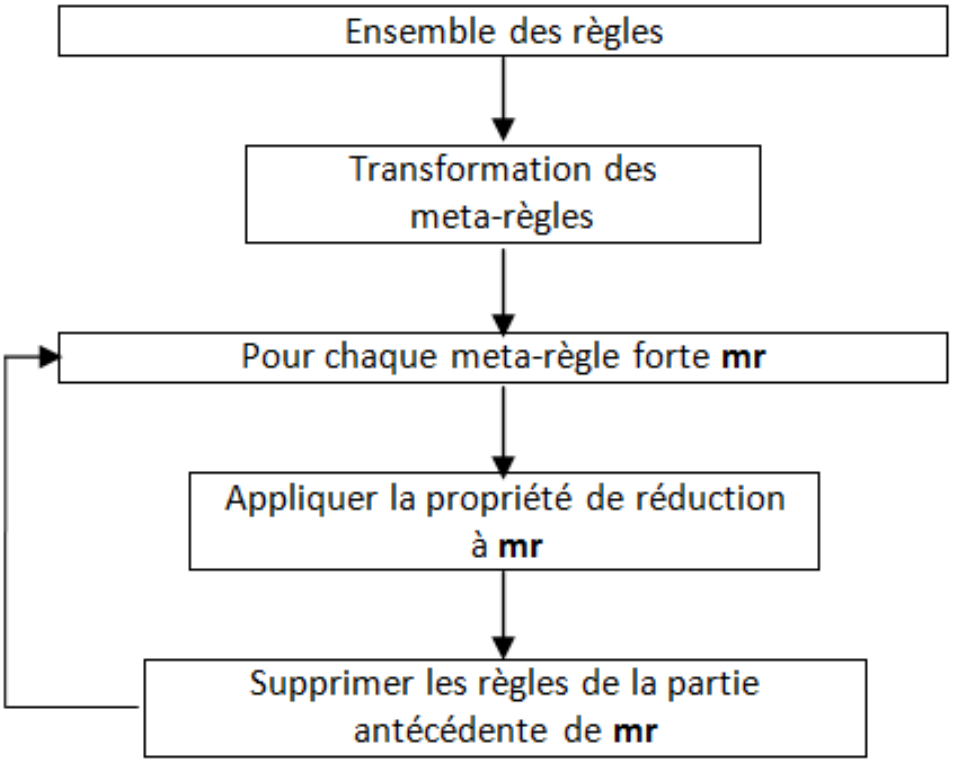


FIGURE 5.1 – L'orgranigrame général de l'approche Pruning-Rules

génération des meta-règles, pour celà, trois algorithmes (PRSBSO, PRCSBSO, PRPBSO) vont être proposés dans le but de réduire le temps de génération des meta-règles.

- **L'algorithme PRSBSO** Dans l'lagorithme PRSBSO (Pruning Rules with Sequential Bees Swarm Optimization), l'extraction de meta-règles se fait en utilisant l'algorithme BSO-MR ou bien l'algorithme HBSO-MR. Ensuite, on applique pour chaque règle forte la proporiété de réduction des règles d'association afin de garder juste les règles représentatives. BSO-MR et HBSO-MR réduisent considérablement le temps de calcul de l'extraction des meta-règles. Cependant, la complexité de calcul reste assez importante lorsqu'il s'agit d'un grand nombre de règles. Afin de réduire encore une fois la complexité de l'étape d'extraction des meta-règles, deux autres algorihmes seront proposés.
- **L'algorithme PRCSBSO** Dans l'algorithme PRCSBSO (Pruning Rules with Clustering and Bees Swarm Optimization), l'espace est divisé en plusieurs groupes, dans chaque groupe, il y'a les règles les plus similaires. Le groupement des règles d'association se fait en utilisant l'algorithme Kmeans-AR. Ensuite, Pour chaque groupe, on applique l'algorithme des meta-règles sur l'ensemble de ces règles avec l'algorithme BSO-MR ou bien l'algorithme HBSO-MR. C'est à dire pour chaque groupe, on applique l'algorithme précédent de PRSBSO. On ne peut rien à dire sur la qualité des règles obtenues c'est pour cette raison qu'on doit vérifier cet aspect dans la partie expérimentale.
- **L'algorithme PRPBSO** Dans l'algorithme PRPBSO (Pruning Rules with Parallel Bees Swarm Optimization), l'extraction des meta-règles est faite en utilisant l'algorithme SEGPU-MR ou bien MEGPU-MR sur la platforme GPU. D'abors, la transformation des règles en forme transactionale se fait en CPU. Ensuite, on applique l'algorithme SEGPU ou bien MEGPU dans GPU. Les meta-règles fortes obtenues sont envoyées en CPU. A la fin, la propriété de réduction des règles d'association s'effectue en CPU pour chaque meta-règle forte retournée.

  On a montré dans le quatrième chapitre que la qualité des solutions trouvées en appliquant l'algorithme BSO-ARM est la même qu'en appliquant

SEGPU et MEGPU. L'extraction des meta-règles de l'algorithme PRPBSO est basée sur le principe de SEGPU et MEGPU, et l'extraction des meta-règles est faite en se basant sur le principe de l'algorithme PRSBSO. Donc, la qualité des meta-règles obtenues par cet algoirthme est la même que la qualité des meta-règles obtenues par l'algorithme PRSBSO. Par conséquent, dans la partie expérimentale, on doit comparer PRPBSO et PRSBSO en terme du temps d'exécution et non pas en terme de la qualité des règles résultantes.

5.4.3 Exemple de l'approche de réduction des règles d'association

Une illustration de l'approche Pruning-Rules est donnée dans l'exemple suivant qui contient 15 différentes règles :

- $r_1 : A \Rightarrow B, C, D.$
- $r_2 : A \Rightarrow D, E.$
- $r_3 : A \Rightarrow D.$
- $r_4 : E \Rightarrow B, C.$
- $r_5 : E \Rightarrow C.$
- $r_6 : F \Rightarrow G, H.$
- $r_7 : F \Rightarrow H, I.$
- $r_8 : F \Rightarrow H.$
- $r_9 : I \Rightarrow G.$
- $r_{10} : H \Rightarrow I.$
- $r_{11} : K \Rightarrow L, M.$
- $r_{12} : K \Rightarrow M, N.$
- $r_{13} : K \Rightarrow N.$
- $r_{14} : L \Rightarrow M, N.$
- $r_{15} : M \Rightarrow L, N.$

Par exemple, si on veut utiliser l'algorithme PRCSBSO, tout d'abord, l'algorithme Kmeans-AR est appliqué pour grouper ces règles. La Table 5.2 présente les clusters obtenus en fixant K à 3.

Cluster C_i	Règles dans C_i
C_1	r_1,r_2,r_3,r_4,r_5
C_2	r_6,r_7,r_8,r_9,r_{10}
C_3	r_{11},r_{12},r_{13},r_{14},r_{15}

TABLE 5.2 – Les groupes obtenus par Kmeans-AR

Identifiant de la transaction t_i	Règles dans t_i
B	r_1,r_4
C	r_1,r_4
D	r_1,r_2,r_3
E	r_2

TABLE 5.3 – La base des règles transactionales de premier cluster

Ensuite, pour chaquer cluster, on transforme les règles en forme transactionale. A titre d'exemple, la base des règles transacionales pour le cluster C_1 est décrit dans la Table 5.3. La première transaction représente les règles qui partagent l'item B dans leurs parties conséquentes. Le même processus est fait pour les trois transactions qui restent.

Aprés cette étape, l'algorithme des meta-règles est appliqué afin d'obtenir les différentes relations entre les règles. La Table 5.4 présente l'ensemble des meta-règles trouvées pour le premier cluster ainsi que la confiance de chaque meta-règle.

A partir de cet table, les meta-règles qui ont une confiance de 100% sont extraites. On assume mr_2 et mr_6 sont ces meta-règles. Pour mr_2, on garde r_1 et on supprime r_6 et pour mr_6, on garde r_1 et on supprime r_3. Donc, les règles supprimées pour le premier cluster sont : r_3 et r_4. Ce processus est répété pour les deux autres clusters restants.

meta-règle	La partie antécédente	la partie conséquence	La confiance
mr_1	r_1	r_4	66%
mr_2	r_4	r_1	100%
mr_3	r_1	r_2	33%
mr_4	r_2	r_1	50%
mr_5	r_1	r_3	33%
mr_6	r_3	r_1	100%

TABLE 5.4 – Les meta-règles résultantes

5.5 Expérimentations et résultats

Afin de valider l'algorithme de Kmeans-AR et l'approche de meta-règles, les règles sont générées en appliquant l'algorithme Apriori à la base de transactions IBMQuest [89]. Cependant, les approches de réduction des règles d'association sont évaluées en utilisant les règles qui sont détaillées dans [65].

5.5.1 Expérimentation de l'algorithme Kmeans-AR

Dans cette première expérimentation, le but est de choisir les meilleurs paramètres de l'algorithme Kmeans-AR qui sont utilisés par la suite dans l'évaluation des approches de Pruning-Rules. Kmeans-AR dépend de trois paramètres : Les formules appliquées dans le calcul de similarité et les centres de gravité ainsi que le nombre de clusters K. Dans cette expérimentation, K est réspectivement fixé à(20, 40, 60, 80, et 100) et pour chacune de ces valeurs, les formules de calcul de similarité et les centres de gravité sont variées afin de déterminer le temps d'exécution et la fonction MMSD (voir equation 5.9) qui minimise la moyenne des distances de chaque règle à son centre. Les résultats sont rapportés dans la Table 5.5.

$$MMSD_{min} = \sum_{i=1}^{K} \sum_{j=1}^{|C_i|} D(r_{i,j}, g_i). \tag{5.9}$$

Tel que
C_i : Le i^{me} cluster.
$r_{i,j}$: Le j^{me} règle dans le i^{me} cluster.
g_i : Le centre de cluster C_i.
D($r_{i,j}$, g_i) : La distance qui sépare entre la règle $r_{i,j}$ et le center g_i.

Les collonnes de 1 à 3 de la Table 5.5 représentent réspecitvement les formules de distance, le nombre des clusters, et le résultat de la fonction MMSD. Les collonnes de 4 à 8 représentent les temps d'exécutions pour un nombre de clusters

Distances	K	MMSD	CC	FC	UOI	IUC	UIC
Manhattan	20	17.01	0.12	0.07	0.09	0.07	0.10
Chebyshev		17.01	0.09	0.12	0.08	0.12	0.06
Euclidian		17.01	0.56	0.31	0.20	0.21	0.22
Euclidian_squared		17.01	0.59	0.24	0.35	0.60	0.57
Dist_items		17.01	0.18	0.27	0.30	0.22	0.16
Ration Distance		17.01	0.08	0.07	0.10	0.10	0.16
Manhattan	40	16.69	0.32	0.31	0.28	0.27	0.16
Chebyshev		16.69	0.34	0.15	0.37	0.13	0.210
Euclidian		16.69	0.47	0.46	0.87	0.42	1.18
Euclidian_squared		16.69	0.91	0.87	0.48	0.62	0.63
Dist_items		16.69	0.42	0.43	0.15	0.20	0.17
Ration Distance		16.69	0.20	0.22	0.34	0.16	0.16
Manhattan	60	16.55	0.20	0.19	0.28	0.21	0.20
Chebyshev		16.55	0.23	0.20	0.18	0.28	0.44
Euclidian		16.55	0.64	1.14	0.89	1.44	1.09
Euclidian_squared		16.55	0.75	1.26	1.10	1.56	1.21
Dist_items		16.55	0.54	0.23	0.25	0.27	0.25
Ration Distance		16.55	0.22	0.25	0.20	0.68	0.41
Manhattan	80	19.29	0.35	0.32	0.36	0.39	0.35
Chebyshev		19.29	0.45	0.48	0.42	0.46	0.46
Euclidian		19.29	0.22	0.25	0.27	0.28	0.21
Euclidian_squared		19.29	0.24	0.32	0.35	0.36	0.24
Dist_items		19.29	0.45	0.39	0.37	0.36	0.38
Ration Distance		19.29	0.46	0.48	0.49	0.37	0.35
Manhattan	100	19.75	0.44	0.40	0.42	0.46	0.44
Chebyshev		19.75	0.47	0.39	0.32	0.45	0.40
Euclidian		19.75	0.42	0.35	0.36	0.35	0.38
Euclidian_squared		19.75	0.75	0.65	0.69	0.72	0.75
Dist_items		19.75	0.37	0.36	0.32	0.33	0.30
Ration Distance		19.75	0.31	0.29	0.27	0.27	0.25

TABLE 5.5 – L'algorithme Kmeans-AR avec différentes formules de calcul de similrité et les centres de gravité en terme de temps d'exécution (Sec)

donné et avec différentes valeurs de calcul de similarité et des centres de gravité qui ont été détaillées dans la Section 5.2).

Premièrement, on note que le nombre de clusters K influence sur la qualité des clusters (MMSD). En effet, la valeur de MMSD diminue tout en augmentant le nombre de clusters jusuq'à la valeur $K = 60$ ensuite la valeur de MMSD augmente encore en variant K de 60 à 100. Par conséquent, la meilleure valeur de K choisie pour les tests ultérieures de Pruning-Rules est 60.

Deuxièment, on remarque que les formules de calcul de mesure de similarité et les centres de grvité n'influencent pas sur la valeur MMSD. En effet, cette valeur dépend seulement du K et elle est fixe en variant ces deux paramètres pour une valeur de K donnée. Pour celà, la seule façon de choisir les meilleurs formules de ditances et des centres de gravité est de calculer le temps d'exécution de chaque valeur de ces deux paramètres. Comme il est mentionné dans la table, pour $K = 60$, Kmeans-AR est plus rapide en utilisant les formules Chebyshev et UOI (0.18 Sec).

Aprés cette première expérimentation, les meilleurs paramètres de l'algorithme Kmeans-AR sont : La distance Chebyshev, la formule UOI et $K = 60$.

5.5.2 Expérimentation de l'algorithme de meta-règles

Le but de cette expérimentation est de séléctionner les meilleurs paramètres de l'approche de meta-règles. Pour celà, plusieurs tests ont été établis sur les algorithmes BSO-MR et HBSO-MR. Les algorithmes SEGPU-MR et MEGPU-MR seront évalués ensuite dans la section suivante.

Dans les approches de Pruning-Rules, on s'intéresse seulement à l'ensemble de meta-règles qui ont 100% de confiance. Par conséquent, dans cette deuxième expérimentation, on évalue les algorithmes (BSO-MR et HSO-TS-MR) en calculant le nombre de meta-règles qui ont 100% de confiance avec les trois stratégies de détermination des régions et avec différents nombres d'abeilles(k).

Les Figures. 5.2-(1), (2), et (3) présentent le nombre des meta-règles fortes (meta-règles qui ont 100% de confidence) des deux algorithmes (BSO-MR et HBSO-MR) avec les stratégies modulo, next et syntactic. L'axe des X des figures (1) et (2) représente les valeurs de Flip pour les stratégies next et modulo et il

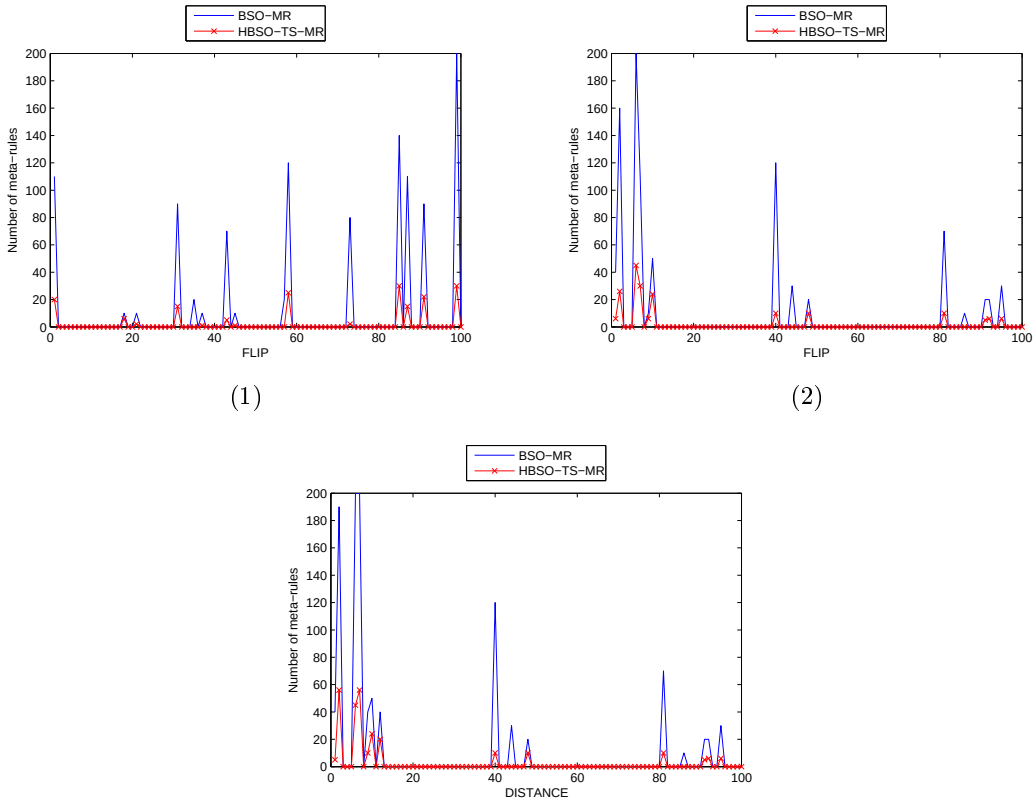
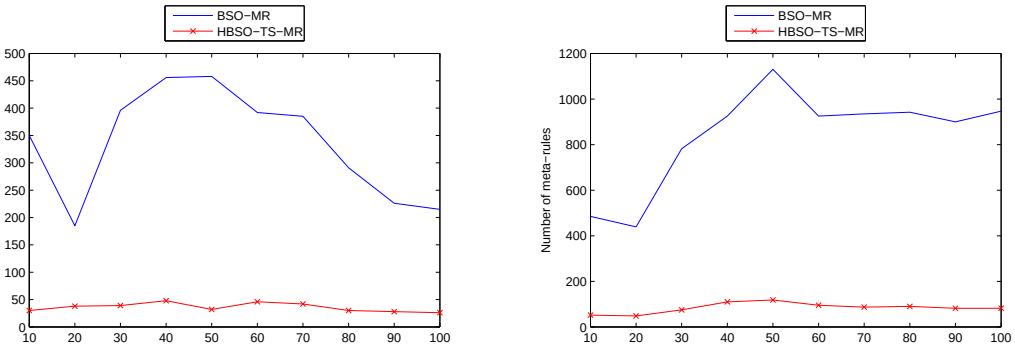


FIGURE 5.2 – Nombre des meta-règles en utilisant les stratégies modulo (1) et next (2) avec différentes valeurs de Flip, et utilisant la stratégie syntactic (3) pour différentes valeurs de Distance.

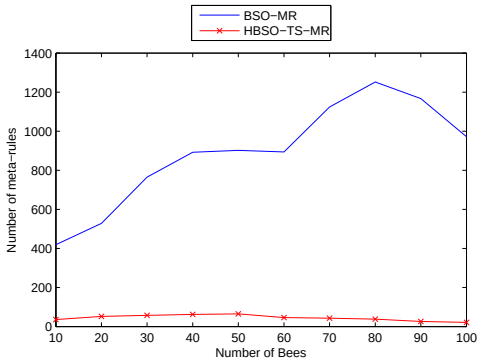
représente les valeurs de Distance pour la stratégie syntactic dans la figure (3). Flip et Distance varient de 1 à 100. Les courbes de la couleur bleu réprésentent le nombre des meta-règles fortes obtenues par BSO-MR et les courbes rouges représentent celles qui sont obtenues par HBSO-MR.

Premièrement, on note que BSO-MR est plus performant que HBSO-MR en terme du nombre des meta-règles fortes. Cependant, malgré que BSO-MR donne un nombre important des meta-règles fortes, HBSO-MR peut trouver des meta-règles utiles (voir Section. **??** pour plus de détailles). En effet, le nombre maximum des meta-règles trouvées par BSO-MR est 200, tandis que, il ne dépasse pas 60 avec HBSO-MR. De plus, les meilleures valeurs de Flip et Distance sont choisies à partir de ces figures :



(1)

(2)



(3)

FIGURE 5.3 – Nombre des meta-règles en utilisant les stratégies modulo (1) et next (2) et syntactic (3) avec différents nombres d'abeilles

- A partir de la figure (1), la valeur de Flip de la stratégie modulo de BSO-MR est 99 et dans HBSO-MR est égale à 85.
- A partir de la figure (2), la valeur de Flip de la stratégie next des deux algorithmes est égale à 6.
- A partir de la figure (3), la valeur de Distance de la stratégie syntactic des deux algorithmes est égale à 7.

La prochaine étape est d'adjuster le nombre d'abeilles des deux approches. Les paramètres Flip et Distance sont choisis selon les tests précédents.

Les Figures 5.3-(1), (2), et (3) présentent le nombre des meta-règles fortes avec les stratégies modulo, next et syntactic des algorithmes (BSO-MR et HBSO-MR). L'axe des X représente le nombre d'abeilles qui est varié entre 10 et 100.

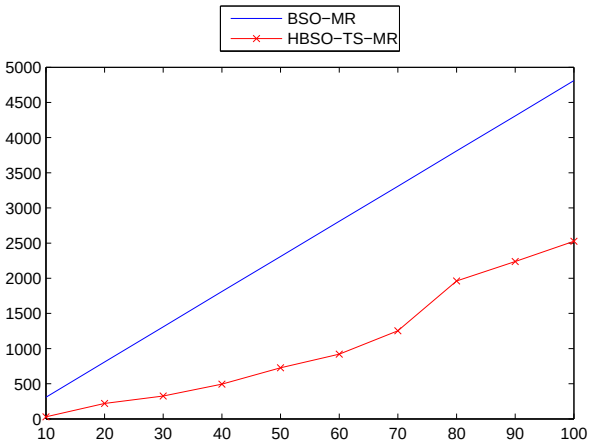


FIGURE 5.4 – Nombre des meta-règles en utilisant la stratégie sytactic avec différents nombres d'itérations

Les courbes bleus représentent le nombre des meta-règles fortes obtenues par BSO-MR et les courbes rouges représentent celles de l'algorithme HBSO-MR.

Ces expérimentations confirment l'efficacité de BSO-MR par rapport à HBSO-MR en terme de nombre des meta-règles. En effet, le nombre des meta-règles obtenues par BSO-MR est plus grand que celui obtenues par HBSO-MR avec n'import quel nombre d'abeilles et n'importe quelle stratégie utilisée.

En plus, les meilleures valeurs de k sont déterminées selon ces figures :

- A partir de la figure (1), dans l'algorithme BSO-MR k est égal à 50 et dans HBSO-MR il est égal à 40.
- A partir de la figure(2), k dans les deux algorithmes est égal à 50.
- A partir de la figure(3), k est égal à 80 dans BSO-MR et à il est à 50 dans HBSO-MR.

On remarque aussi que la stratégie syntactic génère un nombre important de règles d'association comparant avec les deux autres stratégies. Pour celà, elle est utilisée dans les prochaines expérimentations.

La Figure 5.4 présente le nombre des meta-règles pour les deux algorithmes en utilisant la stratégie syntactic avec différents nombres d'itérations. L'axe des X représente le nombre d'itérations qui est varie entre 10 et 100. Les courbes bleus représentent le nombre des meta-règles obtenues par BSO-MR et les courbes

rouges sont obtenues par HBSO-MR.

En augmentant le nombre d'itération de 10 à 100, le nombre de meta-règles des deux algorithmes augmentent. Celà est dû par le fait que dans chaque itération des nouvelles meta-règles qui sont générées par la colonie des abeilles de chaque algorithme.

De plus, la pente de la courbe de BSO-MR est plus grande que celle de HBSO-MR car le nombre des meta-règles générées par BSO-MR est plus grand que par HBSO-MR.

La Table 5.6 résume les meilleures valeurs des paramètres des algorithmes (BSO-MR et HBSO-MR) qui sont obtenues aprés ces expérimentations.

	La stratégie de détermination des régions	Distance	k	IMAX
BSO-MR	syntactic	7	80	100
HBSO-TS-MR	syntactic	7	50	100

TABLE 5.6 – Résumé des paramètres selectionnés des deux algorithmes

5.5.3 Expérimentation de l'approche Pruning-Rules

Dans un premier temps, on va valider l'approche de Pruning-Rules avec le travail de Berrado [65] qui utilise l'aspect de meta-règles pour la réduction de l'espace des règles. Pour celà, on compare l'algorithme PRSBSO par rapport à l'algorithem de Berrado, dans l'extraction des meta-règles, on utilise l'algorithme BSO-MR qu'on a vu son efficacité par rapport à l'algorithme HBSO-MR en terme de nombre de meta-règles fortes. Ensuite, on compare les deux autres algorithmes (PRCSBSO et PRPBSO) réspectivement avec l'algorithme PRSBSO.

Les résultats de l'algorithme PRSBSO

On propose maintenant le résultat de l'algorithme PRSBSO en le comparant avec le travail de Berrado qui, aussi utilise le principe de meta-règles. Les paramètres utilisés dans l'algorithme PRSBSO sont fixés comme suit :

- Le nombre d'itération IMAX de BSO-MR est égal à 100.
- Le paramètre Flip est égal à 99.
- Le nombre d'abeilles est égal à 80.

Le nombre de règles	PRSBSO	L'algorithme de Berrado
371	**702**	517
797	**408**	147
156	**592**	314
1215	**1770**	1421
1442	**3204**	2698
1374	**1214**	953
1707	**1044**	945

TABLE 5.7 – Nombre de meta-règles fortes

Le nombre de règles	PRSBSO	L'algorithme de Berrado
371	**216**	246
797	**300**	316
156	**198**	211
1215	**452**	502
1442	**502**	567
1374	**138**	120
1707	**202**	300

TABLE 5.8 – Nombre des règles restantes

Le nombre de règles	PRSBSO	L'algorithme de Berrado
1000	**422**	400
2000	**1425**	1365
3000	**2352**	2105
4000	**2880**	2409
5000	**3115**	3110
6000	**3752**	3325
7000	**4403**	4091
8000	**5196**	4895
9000	**5965**	5263
10000	**6208**	6110

TABLE 5.9 – La qualité des règles obtenues

Les Table 5.7 et Table 5.8 présentent réspectivement le nombre des meta-règles obtenues et le nombre des règles restantes par notre algorithme et l'algorithme de Berrado avec différents nombre des règles en entrées.

A partir de la Table 5.7, on remarque que avec n'importe quel nombre des règles en entrées, le nombre des meta-règles obtenu par notre algorithme est plus grand que celui obtenu par l'algorithme de Berrado. En effet, PRSBSO, génère toutes les meta-règles qui ont 100% de confiance. Ce qui n'est pas le cas pour l'algorithme de Berrado qui génère des meta-règles de taille 2 seulement. Ces résultats valident notre étude dans la Section 5.3.

A partir de la Table 5.8, on peut aussi remarquer que le nombre des règles résultantes qui sont obtenues par l'algorithme Berrado est grand par rapport au nombre de règles obtenues par notre algorithme. Celà est dû par le fait que dans l'algorithme Berrado, il y'a beacoup d'informations manquantes avec la génération d'un nombre réstreint de meta-règles. Malgré que notre algorithme est meilleur que l'algorithme de Berrado en terme de nombre des règles résultantes, en revanche aucune information ne peut être extraire concernant la qualité de ces règles.

Pour évaluer la qualité des règles résultantes par les deux algorithmes, un autre test a été effectué. La Table 5.9 montre la qualité des règles résultantes par les deux algorithmes avec différents nombres des règles en entrées en utilisant l'equation 5.8. Cette table décrit que l'algorithme PRSBSO surpasse l'algorithme de Berrado en terme de qualité des règles résultantes avec différent nombres des règles en entrée. En effet, Dans PRSBSO, tout type de règles peut être trouvé, par conséquent, cet algorithme résume trés bien l'espace des règles. Cependent, dans l'algorithme de Berrado, seulement les règles de taille 2 sont utilisées donc, uniquement les règles similaires sont trouvées. Par conséquent, ce dernier algorithme ne couvre pas toutes les règles possibles.

PRSBSO vs PRCSBSO

La Table 5.10, présente les rapports des algorithmes PRCSBSO et PRSBSO des temps d'exécutions et la qualité des règles calculée par l'équation 5.8. D'aprés cette table, on remarque que les règles obtenues par les deux algorithmes ont

Nombre de règles	(PRCSBSO/PRSBSO) (Sec)	(PRCSBSO/PRSBSO) (Qualité)
100	0.42	1.02
200	0.72	1.05
300	0.73	1.01
400	0.78	1.00
500	0.79	1.00
600	0.80	0.98
700	0.86	0.92
800	1.00	1.02
900	1.05	1.05
1000	1.20	1.06

TABLE 5.10 – PRCSBSO Vs PRSBSO en terme de terme de temps d'exécution (Sec) et la qualité des règles trovuée

presque la même qualité, ce résultat est réalisé grâce à la manière de groupement des règles de départ. En effet, juste les règles (de même cluster qui ont une forte dépendance) sont traitées et n'est pas toutes les règles.
Concenrnant le temps d'exécution, on remarque que PRCSBSO est meilleur que PRSBSO, tout en augmentant le nombre de règles de 100 à 800. Cependant, à partir de 900 règles, PRSBSO est meilleur que PRCSBSO, on peut expliquer ceci par le fait que par l'utilisation de Kmeans, PRCSBSO consomme beaucoup de temps dans la phase de groupement des règles.

PRSBSO vs PRPBSO

La Table 5.11, présente le rapport des temps d'exécutions entre l'algorithme PRSBSO et sa version paralléle. En augmentant le nombre de règles de 100 à 1000, PRPBSO est meilleur que PRSBSO, de plus, lorsque le nombre de règles est 1000, le temps d'exécution de PRPBSO est 35 fois meilleur que l'algorithme PRSBSO.

Nombre de règles	PRSBSO/PRPBSO
100	1.20
200	2.65
300	5.66
400	6.78
500	8.65
600	9.65
700	10.25
800	15.23
900	22.32
1000	35.25

TABLE 5.11 – PRPBSO Vs PRSBSO en terme de terme de temps d'exécution (Sec)

5.6 Conclusion

Dans ce chapitre, on a proposé deux approches de fouille de règles ainsi que l'application de ces approches pour la réduction de l'espace des règles. La première approche appelée Kmeans-AR est une extension de l'algorithme Kmeans dans le but d'améliorer la manière de groupement des règles d'association. La deuxième approche applique le même processus d'extraction des règles d'association sur l'ensemble des règles. Le but de cette approche est de créer des dépendances entre les règles représentées sous forme de meta-règles. Ainsi, une extension des algorithmes proposés dans le quatrième chapitre est établie.

Dans la dernière partie de ce chapitre, on a vu l'utilisation des deux approches proposées dans la réduction de l'espace des règles d'association. Pour celà, on a proposé trois algorithmes, PRSBSO, et PRPBSO utilisant juste la deuxième approche qui est l'extraction des meta-règles, tandis que PRCSBSO, est une sorte d'hybridation entre le groupement des règles et l'extraction des meta-règles.

Chapitre 6

L'extraction des règles d'association pour le problème SAT

6.1 Introduction

SAT est le premier problème qui a été montré NP-complet et, considéré par plusieurs chercheurs. Plusieurs approches ont été dévelopées pour résoudre le problème SAT. Un travail récent a été proposé pour résoudre d'une manière efficace une instance SAT. Il divise le problème SAT d'une grande taille en plusieurs sous problèmes de petite taille. Ensuite, il résout chaque problème apart afin de trouver la solution globale du problème de départ.
La division de problème en sous problèmes se fait grâce aux connaissances extraites à partir de l'instance SAT. En effet, il applique un processus de fouille à partir de l'instance en entrée. La téchnique utilisée est la classification non supervisée, il groupe les différentes clauses d'une instance SAT afin de trouver un ensemble de clusters des clauses. Par conséquent, chaque cluster est vu comme un sous problème par la suite dans l'étape de résolution. Cependant, la méthode de classification proposée de cette approche est naive, en effet, on peut avoir la même variable dans deux clusters différents. Afin de bien grouper les clauses et de minimiser le nombre de variables communes, nous proposons dans ce chapitre un algorithme de classification non supervisée basé sur l'extraction des règles d'association. L'application de l'algorithme BSO-ARM déja proposé au quatrième

chapitre est établie.

6.2 Le problème SAT

6.2.1 Déscription du problème

Etant donnée un ensemble de variables booléennes $V = \{v_1, v_2, ..., v_n\}$ et une fonction t qui assigne deux valeurs possibles (Vraie ou Fausse) de chaque variable v_i. Un litérale est une variable qui apparait avec ou sans l'opérateur de négation. Une clause est une disjunction d'un ensemble de litéraux. Une instance SAT est une conjonction de clauses.
Le problélme SAT est défini formellement par les données et la question suivantes :

- Données : m clauses définies sur n litéraux.
- Question : Existe il une instanciation des variables dont lequel les clauses sont toutes vraies. ?

Une clause est vraie si elle contient au moins un litérale avec la valeur vraie ou bien un litérale négatif avec la valeur fausse.
Exemple
Considérons l'ensemble des variables suivantes : $V = \{v_1, v_2, v_3, v_4\}$ et l'ensemble des clauses suivantes : $C = \{C_1, C_2, C_3\}$ définies comme :
$C_1 = v_1, v_2, -v_4\}$
$C_1 = v_2, -v_3, v_4\}$
$C_1 = -v_1, -v_2, v_3\}$
Le signe (-) exprime la négation et le symbole (,) est l'opérateur de disjunction.
Une solution possible de cette instance est {Vraie, Vraie, Vraie, Vraie} car :
Vraie ou Vraie ou Fausse= Vraie
Vraie ou Fausse ou Vraie= Vraie
Fausse ou Fausse ou Vraie=Vraie

6.2.2 Les méthodes de résolution classique pour le probéme SAT

Dans la litérature, il existe plusieurs approches de résolution pour le problème SAT. La première catégorie des approches apppelée, les approches complètes, car elles sont capables de trouver la solution optimale si elle existe ou prouver qu'elle n'existe pas. La solution optimale d'un problème SAT est la satisfaction de toutes les clauses d'une instance donnée. Les approches complètes les plus utilisées sont :

- DPP (Davis-Puntnam Procedure) : DPP [92], est l'un des algorithmes complets qui est largement utilisé pour la résolution du problème SAT. L'idée de cet algorithme est la construction d'un arbre binaire en utilisant la récursivité et le retour en arriére. Ainsi, la taille de l'espace de recherche est considérablement réduit par rapport à d'autres méthodes en éliminant des variables à partir de l'instance SAT. En effet, dans chaque itération, on séléctionne une variable donnée v et on lui affecte les deux valeurs possibles (Vraie ou Fausse) en eliminant toutes les clauses qui contiennent v ou bien $-v$.
- DPLL (Davis-Puntnam-Logemann-Loveland) DPLL [93] est une version améliorée de l'algorithme DPP. Loveland et Logemann introduisent une règle de division qui consiste d'affecter les deux valeurs booléennes de chaque variable. Ensuite, il résout d'une manière récusrive les sous-problèmes résultants par cette affectation. Quand une clause est satisfaite, elle est supprimée à partir de l'instance. Cette opération est appelée la règle de simplificiation. Ce processus est répété jusqu'à la fin des traitements des clauses.

Quand la taille d'instance à résoudre est trés importante, les algorithmes exacts peuvent engendrer une explosion combinatoire durant leurs exécutions. Par conséquent, les approches incomplètes ont été conçues afin de traiter une solution proche de la solution optimale, en d'autres terme, de satisfaire le maximum des clauses d'une instance donnée.
Plusieurs approches incomplètes ont été dévellopées pour résoudre le problème SAT. Parmi ces approches, on peut citer :

- GSAT [94] est une recherche locale aléatoire, Il commence par séléctionner aléatoirement des valeurs des variables. Ensuite, Il construit un certain nombres de flips qui réduit le nombre des clauses unsatisfiables. Ce processus est répété jusqu'à l'obtention de la solution optimale ou alors l'atteinte d'un nombre limité d'itérations.
- Walksat [95] est une version améliorée de GSAT. Une probabilité brute est introduite afin de réaliser des sauts aléatoires dans l'espace de solutions. En effet, avec la probabilité d'une valeur p, la variable déja choisie est assignée aléatoirement, alors que avec une probabilité d'une valeur $1-p$, la variable qui maximise le nombre de clauses satisfiables est séléctionnée. D'autres part, Plusieurs méthodes basées sur l'intelligence par essaims comme ACO [96] et BSO [86] ont été adaptées pour le problème SAT.

6.3 L'algorithme BSO-DM

6.3.1 Principe

L'algorithme BSO-DM est proposé par Drias et d'autres dans [97], il est composé de deux étapes : La classification non supervisée des clauses et la résolution des clusters.

- **La classification non supervisée des clauses :** Un cluster contient un ensemble de clauses et il est représenté par un ensemble de variables. L'insertion des clauses se fait d'une manière séquentielle. On commence par la création de premier cluster qui contient la première clause. Pour chaque prochaine clause, on vérifie l'existance de ses variables dans les clusters déja crées. S'il n'existe aucune variable alors on crée un nouveau cluster avec cette clause et ses variables deviennent les variables représentatives de ce nouveau cluster. Dans le cas où les variables de cette clause inclusent dans les clusters déja crées, ladite clause sera insérée dans le cluster qui partage un maximum de variables avec elle. Ce processus doit être répété jusqu'au traitement de toutes les clauses. La complexité de cet algorithme est $O(n \times m)$ tel que n est le nombre des variables et m est le nombre de

toutes les clauses.

- **La résolution des clusters :** Les clusters obtenus par la première étape sont considérés comme des sous-problèmes SAT. De cette manière, on peut facilement les résoudre. Chaque cluster des clauses peut être traiter séparament. Pour chaque cluster, on applique l'algorithme de BSO développé pour le problème SAT en 2005 [86]. Dans la résolution des clusters on ne considère pas toute l'instance de SAT mais juste les clauses formant chaque cluster.

6.3.2 Exemple

On considère les clauses suivantes avec les cinq variables $\{v_1, v_2, v_3, v_4, v_5\}$:
$C_1 = v_1, v_2, v_3$
$C_2 = v_4, v_5$
$C_3 = v_2, v_3, v_4$
$C_4 = v_2, v_4, v_5$
Initialement, on affecte la première clause au premier cluster, ainsi, les variables de cette clause deviendront des variables représentatives de ce premier cluster :
Cluster(1)=$\{C_1\}$
represent(1)=$\{v_1, v_2, v_3\}$
Ensuite, les trois prochaines clauses seront placées de la manière suivante :
Pour C_2 : il y'a aucune variable de C_2 qui appartient à represent(1), donc on crée un nouveau cluster pour C_2 et on note : Cluster(2)=$\{C_2\}$ et represent(2)=$\{v_4, v_5\}$
Pour C_3 : on calcule le nombre de variables communes entre C_3 et réspectivement represent(1) et represent(2).
Avec represent(1), C_3 partage deux variables (v_2 et v_3) par contre, avec represent(2), C_3 partage une seule variable (v_4) donc on insère C_3 au premier cluster.
Pour C_4, on suit le même principe que pour C_3, sauf ici, la clause C_4 partage deux variables avec represent(2) et une seule avec represent(1), alors, C_4 est insérée dans le deuxième cluster.

Par conséquent, les clusters finals sont :
Cluster(1)=$\{C_1, C_3\}$
Cluster(2)=$\{C_2, C_4\}$
Aprés cette étape, on aura deux sous problèmes SAT :
Le premier sous-problème contient deux clauses C_1 et C_3 et trois variables (v_1, v_2, v_3).
Le deuxième sous-problème contient deux autres clauses C_2 et C_4 et deux variables $\{v_4, v_5\}$.
Par la suite, ces deux sous-problèmes seront résolus en utilisant l'algorithme BSO adapté à SAT.

6.3.3 Limite

Le problème de cette approche est dans l'étape de classification non supervisée des clauses. A la fin de cette étape, on peut avoir des clusters qui ne sont pas totalement disjoints, en d'autres termes, on peut avoir des variables incluses dans deux clusters différents. Donc lors de la résolution des clutsers, on peut tomber dans le cas où la même variable peut avoir deux valeurs possibles dans deux clusters différents. En effet, si on prend l'exemple précédent, la variable v_2 incluse dans C_1 et C_2 donc si lors de la résolution, on affecte la variable v_2 la valeur *Vraie* pour le premier cluster et la valeur *Fausse* pour le deuxième cluster, alors on aura une contradiction et une fausse résolution de la solution gloable.
Ce problème est dû par le fait que dans la première étape, on utilise un algorithme naîf pour grouper les clauses.
Pour celà, et afin de réduire le nombre de variables communes, on applique l'extraction des règles d'association pour grouper les différentes clauses d'un problème SAT donné. On a vu que la complexité de l'algorithme précédent de classification non supervisée des clauses est $O(n \times m)$, donc si on applique l'algorithme APRIORI à l'étape de classification non supervisée des clauses, on aura un autre problème qui est le temps de calcul. Par conséquent, l'exploitation des algorithmes déja proposés dans le quatrième chapitre peut réduire le problème de classification non supervisée des clauses. Dans la prochaine section, on détaillera l'adapatation de notre algorithme BSO-ARM pour le problème de classification non supervisée

des clauses.

6.4 L'algorithme BSO-ARM-CC

Le but de l'algorithme BSO-ARM-CC (Bees Swarm Optimisation for Association Rules Mining in Clustering Clauses), est de grouper les différentes clauses du problème SAT, afin de réduire le nombre de variables communes entre les différents clusters. Avant de voir le principe de l'algorithme, une transformation formelle de problème SAT au problème d'extraction des règles d'association est nécessaire.

6.4.1 Transformation formelle

Considérons un ensemble de clauses C=$\{c_1, c_2, ..., c_m\}$ et un ensemble de variables V=$\{v_1, v_2, ...v_n\}$. La base de transactions des clauses T=$\{t_1, t_2, ..., t_m\}$ qui contient n différents items I=$\{I_1, I_2, ...I_n\}$ est définie commme suit :
$\forall(t_i \in T), (c_i \in C), t_i = c_i$
$\forall(I_i \in I), (v_i \in V), v_i \in C_j \Rightarrow (I_i = 1) \in T_j$
$\forall(I_i \in I), (v_i \in V), v_i \notin C_j \Rightarrow (I_i = 0) \in T_j$
En effet, chaque clause d'une instance SAT peut être vu comme une transaction, et chaque item comme une variable. Si la variable v_i appartient à la clause C_j alors l'item I_i prend la valeur 1 dans la transaction t_j, sinon il prend la valeur 0 dans la transaction t_j.

6.4.2 Principe

Dans un premier temps, la transformation d'une instance SAT à une base de transactions des clauses est faite. Ensuite, on applique l'algorithme BSO-ARM qui est proposé dans le quatrième chapitre afin d'extraire des règles d'association à partir de la base des transactions des clauses. On garde juste les règles qui ont une confiance supérieure à $Minconf$. Soit R=$\{r_1, r_2, ..., r_k\}$ est l'ensemble de ces règles telle que chaque règle $r_i : X_i \Rightarrow Y_i$ avec X_i et Y_i sont des ensembles de variables SAT. On trie l'ensemble R selon le support de ces règles avec un ordre décroissant des règles. On dispose aussi une liste temporaire nommée *List-temp*

qui contient toutes les clauses d'une instance SAT donnée. Initialement, on récupère la première règle $r_1 : X_1 \Rightarrow Y_1$, on crée un cluster pour l'ensemble des clauses contenant les variables qui composent X_1 et Y_1 en même temps. Les variables représentatives du premier cluster sont des variables de X_1 et Y_1. Ensuite, pour chaque règle r_i de R, on vérifie si X_i et Y_i n'appartiennent pas en même temps aux variables représentatives à un cluster déja créé, alors on construit un cluster pour les clauses de X_i et Y_i et on considère les variables de X_i et Y_i comme des variables représentatives de ce cluster. Ce processus est répété pour les K règles de R ou bien jusqu'à ce que la liste temporaire des clauses *List-temp* devienne vide, en d'autres termes, toutes les clauses sont regroupées.
Pour éviter que la même clause appartient à deux clusters différents, chaque fois qu'une clause est insérée dans un cluster, elle sera supprimée de *List-temp*.

6.4.3 Exemple

Considérons l'exemple précédent, initialement la *List-temp* contient toutes les clauses $\{C_1, C_2, C_3, C_4\}$. Tout d'abord, on transforme l'ensemble de clauses à un ensemble de transactions $\{t_1, t_2, t_3, t_4\}$ comme :
t_1=1,1,1,0,0 par ce que les trois premières variables appatiennent à C_1 et les deux dernières variables n'appartiennent pas à C_1.
t_2=0,0,0,1,1.
t_3=0,1,1,1,0.
t_4=0,1,0,1,1.
Ensuite, on applique l'algorithme BSO-ARM à l'ensemble de transactions de t_1 à t_4, on considère dans le processus de fouille juste les règles de confiance qui est supérieure (soit Minconf=100%).
Supposons que les règles obtenues triées par leurs supports sont :
$r_1 : v_5 \Rightarrow v_4(Conf = 100\%, Supp = 50\%$.
$r_2 : v_3 \Rightarrow v_2(Conf = 100\%, Supp = 50\%$.
$r_3 : v_1 \Rightarrow v_2(Conf = 100\%, Supp = 25\%$.
On commence par la première règle, on crée le premier cluster avec les clauses C_2 et C_4 car v_5 et v_4 appartiennent en même temps à C_2 et C_4. On considère

les variables v_4 et v_2 comme des règles représentatives de premier cluster et on supprime C_2 et C_4 de la liste *List-temp*, plus formellement on aura :
Cluster(1)=$\{C_2, C_4\}$
represent(1)=$\{v_4, v_5\}$
List-temp=$\{C_1, C_3\}$
Pour la deuxième règle, on répète le même processus, et on aura :
Cluster(2)=$\{C_1, C_3\}$
represent(2)=$\{v_3, v_2\}$
List-temp=$\emptyset$
La liste *List-temp* est vide, alors on s'arrête sans traiter la règle r_3. L'état final des clusters est :
Cluster(1)=$\{C_2, C_4\}$
Cluster(2)=$\{C_1, C_3\}$
represent(1)=$\{v_4, v_5\}$
represent(2)=$\{v_3, v_2\}$
Aprés cette étape, comme dans l'algorithme BSO-DM, on résoud chaque cluster à part.

6.5 Expérimentation

Plusieurs tests ont été faits dont le but de valider l'efficacité de l'algorithme BSO-ARM-CC. L'algorithme est implémenté en C++ dans Pentium (I3,4GO). Les premiérs tests ont été réalisés seulement sur l'algorithme BSO-ARM-CC, pour règler les paramètres de la métaheuristique BSO comme : Le nombre d'abeilles, le nombre d'itérations et le minimum de confiance. Aprés cette étape, on va faire une comparaison entre l'algorithme proposé dans ce chapitre et l'algorithme BSO-DM.

6.5.1 réglage de paramètres de BSO-ARM-CC

Les tables 6.1, 6.2, 6.3 présentent le nombre de règles obtenues par notre algorithme BSO-ARM-CC, avec différents nombres de clauses (de 100 à 1000), différents nombres de variables (5, 10, 15) et avec différentes valeurs de minimum

confiance(20%, 50%, 80%, 100%).
D'aprés les tables, on remarque que le nombre de règles diminue tout en ajoutant au minimum confiance où il ne dépasse pas les dix règles lorsque le nombre de variables est égal à 5 et le minimum de confiance est à 100%.
On peut dire aussi que le nombre de règles fortes (Minconf=100%) est raisonnable avec n'importe quel nombre de clauses et de variables. Pour celà, on considère Minconf=100% pour tout le reste d'expérimentations.

La Table 6.4 présente le nombre de règles fortes délivrées par notre algorithme

m	20%	50%	80%	100%
100	46	46	13	9
200	45	32	10	10
300	47	35	10	10
400	48	30	10	10
500	43	31	10	10
600	46	46	9	9
700	43	35	10	10
800	48	30	10	10
900	45	33	10	10
1000	43	39	10	10

TABLE 6.1 – Nombre de règles générées avec $n = 5$

m	20%	50%	80%	100%
100	221	174	47	36
200	146	69	23	20
300	136	70	22	20
400	168	116	22	20
500	155	110	20	20
600	177	130	20	20
700	144	119	20	20
800	161	139	20	20
900	180	128	20	20
1000	180	135	20	20

TABLE 6.2 – Nombre de règles générées avec $n = 10$

en fixant le nombre de clauses à 1000 et le nombre de variables à 15 et en variant le nombre maximal d'itération de 5 à 100. D'aprés cette table, on remarque que le

m	20%	50%	80%	100%
100	1392	1058	601	601
200	1256	892	236	236
300	936	637	141	141
400	432	265	75	75
500	449	250	80	80
600	427	252	52	52
700	415	251	45	39
800	417	235	44	38
900	436	274	52	38
1000	396	243	46	37

TABLE 6.3 – Nombre de règles générées avec $n = 15$

IMAX	nombre de règles
5	5
10	13
20	13
30	13
40	13
50	13
60	13
70	13
80	13
90	13
100	13

TABLE 6.4 – Le nombre de règles fortes avec différents nombre d'itérations

nombre de règles fortes augmente jusuqu'à ce que le nombre d'itérations atteigne les 10 itérations. Pour celà, on choisit IMAX=10 pour les prochains tests. De même, la Table 6.5 montre que le nombre de règles fortes augmente en augmentant le nombre d'abeilles jusuq'au 20 abeilles où il stablise à 16 règles fortes.
Par conséquent, les paramètres de l'algorithme BSO-ARM-CC choisis aprés ces expérimentations sont :
Le minimum confiance est égal à 100%.
Le nombre d'itérations maximal est égal à 10.
Le nombre d'abeilles est égal à 20.

K	nombre de règles
1	1
5	5
10	10
20	16
30	16
40	16
50	16
60	16
70	16
80	16
90	16
100	16

TABLE 6.5 – Le nombre de règles fortes avec différents nombre d'abeilles

6.5.2 comparaison de BSO-ARM-CC avec BSO-DM

m	BSO-DM	BSO-ARM-cc
100	67	14
200	121	21
300	177	32
400	237	38
500	284	50
600	349	58
700	410	71
800	456	75
900	479	80
1000	507	85

TABLE 6.6 – La qualité des clusters avec $n = 5$

Les tables présentent la qualité des clusters trouvés par les deux algorithmes (BSO-DM et BSO-ARM-CC) avec différents nombres de clauses et de variables. La qualité de clusters est calculée en utilisant l'equation 6.1, il s'agit de minimiser la distance entre les clauses de même cluster. La distance entre deux clauses est le nombre total de variables moins le nombre de variables communes entre ces

m	BSO-DM	BSO-ARM-cc
100	329	3
200	459	22
300	560	44
400	657	62
500	781	72
600	877	89
700	980	108
800	1055	110
900	1134	128
1000	1251	139

TABLE 6.7 – La qualité des clusters avec $n = 10$

m	BSO-DM	BSO-ARM-cc
100	492	0
200	995	7
300	1493	3
400	1994	12
500	2494	13
600	2993	65
700	3104	68
800	3151	140
900	3269	167
1000	3469	173

TABLE 6.8 – La qualité des clusters avec $n = 15$

deux clauses.

$$MADP_{min} = \sum_{l=1}^{K} \sum_{i=1}^{|Clusterl|} \sum_{j=1}^{|Clusterl|} D(C_i, C_j). \tag{6.1}$$

Tel que
D(C_i, C_j)=n-$vc_{i,j}$.
$vc_{i,j}$= nombre de variables communes entre C_i et C_j.

D'aprés les tables, on remarque une large différence entre la qualité des clusters des deux algorithmes. En effet, les clusters obtenus par notre algorithme sont bien collérés, de telle sorte qu'on observe une forte dépendance entre les clauses de

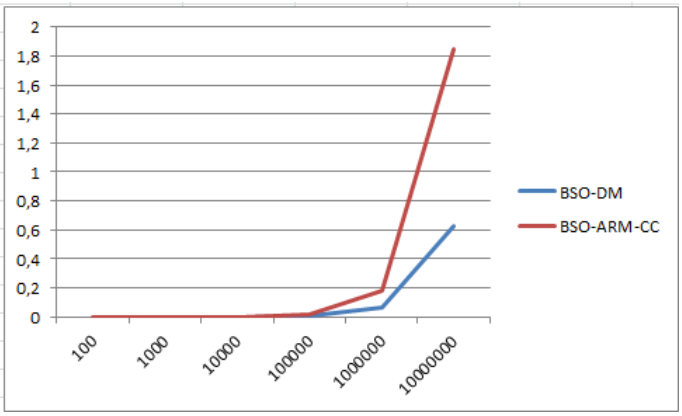


FIGURE 6.1 – Le temps d'execution de groupement des clauses(Sec)

même cluster. On peut expliquer celà par l'information supplémentaire introduite par l'ensemble des règles de l'algorithme BSO-ARM.
La Figure 6.1 montre que le temps d'exécution de notre algorithme et l'algorithme BSO-DM est presque le même en augmentant le nombre de clauses de 100 à $10 millions$. En effet, à 10 millions de règles, il aura 1 Seconde de différence entre les deux algorithmes. Ces résultats montrent la nécessité d'utiliser les règles d'association dans le processus de groupement des clauses pour le problème de satisfiabilité.

6.6 Conclusion

Dans ce chapitre, on a vu l'application de l'algorithme BSO-ARM dans le problème de satisfiabilité (SAT). Les règles obtenues par BSO-ARM sont utilisées pour le groupement des clauses de SAT. Chaque règle forte peut former un cluster de clauses où chaque cluster sera traité par la suite indépendemment.
On a vu aussi que notre algorithme de groupement des clauses est meilleur en terme de la qualité des clusters obtenus par rapport à l'algorithme BSO-DM, tandis que le temps d'exécution des deux algorithmes est le même avec un nombre important de clauses en entrée.
L'application de l'algorithme HBSO-ARM dans l'étape d'extraction des règles est une perspective de ce chapitre.

Chapitre 7

La réduction des règles d'association pour le problème de recherche d'informations

7.1 Introduction

La recherche d'informations est un processus qui permet de trouver un ensemble d'informations pertinentes d'une requête donnée et à partir d'un corpus donné. Avec l'apparition de web, la quantité d'informations est de plus en plus volumineuse. Par conséquent, les méthodes de recherche d'informations classiques deviennent de plus en plus paralysées. Plusieurs approches ont été dévelopées afin de réduire la complexité de calcul de recherche d'informations, A titre d'exemple : la recherche d'informations distribuée basée sur le concept des systèmes multi-agents, la recherche d'informations personalisée qui a pour but de restreindre l'espace de recherche selon le profil de l'utilisateur, la recherche d'informations intelligente qui utilise des approches basées sur des métaheuristiques.
Une autre approche est proposée pour réduire la complexité de problème de la recherche d'informations est l'exploration de l'espace des informations en utilisant un processus de fouille de ces informations. La connaissance extraite peut être vu comme une étape de prétraitement dans le processus de recherche d'informations. Dans ce chapitre, nous allons proposer une nouvelle téchnique basée sur l'extrac-

tion des règles d'associations pour optimiser le temps de réponse des différentes requêtes à des informations représentées par un ensemble de documents. L'application de l'algorithme PRSBSO déja proposé dans le cinquième chapitre est établie.

7.2 La recherche d'informations classique

7.2.1 Concepts de Base

Le problème de recherche d'informations consiste de trouver à partir d'une large collection, les documents qui répondent à une requête donnée exprimée par l'utilisateur. Les concepts suivants sont dans le corp du processus de recherche d'informations [98], [99].

- Document : Il peut être un text, une page web, un video ou d'autres sources d'informations, un document est généralement représenté par un ensemble de termes ou mots clés.
- Requête : Elle représente les besoins de l'utilisateur exprimés par un formalisme adopté par le processus de recherche d'informations.
- L'appariment : Il calcule la similarité entre un document et une requête. En outre, deux évaluations sont largement utilisées pour tester un modèle de recherche d'informations, la précision est la fraction des documents extraits qui sont pertinents et le rappel est la fraction des documents pertinents qui sont extraits.
- L'indexation : C'est une étape importante dans le processus de recherche d'informations, elle consiste de transformer la collection des documents et les requêtes de l'utilisateur à un format uniforme afin de les accédées plus rapidement et d'une manière efficace. Les documents et les requêtes sont représentés en fonction de modèle utilisé. Il exitste plusieurs modèles de recherche d'informations comme le modèle bolléen, le modèle vectoriel et le modèle probabilistique. Le modèle le plus utilisé dans la litérature est le modèle vectoriel, Dans ce modèle, les documents et les requêtes sont représentés par un vecteur de termes pondérés. Chaque poids dénote l'importance

de son terme dans le document ou bien dans la requête. Les vecteurs des documents sont construit durant la phase d'indexation.
Considérons le vecteur d'un document suivant :
$(t_1, t_2, .., t_n)$
tel que t_i est le i^{me} terme dans ce document. Ensuite, pour chaque terme, on considère une structure qui contient tous les docuemnts qui l'apprtiennent. Le poids d'un terme dans un document est associé avec ce document dans une liste. La collection complète C des documents est représentée par un fichier qui contient tous les docuemnts, la structure est indéxée par le nombre de documents.
$C = (d_1, d_2, ..., d_m)$
Chaque élément de C pointe sur une liste qui contient tous les termes des documents avec réspectivement leurs poids. En outre, la requête est modélisée exactement comme le document.

- La pondération : Le poids d'un terme dans un document est calculé en utilisant la formule (tf*idf) tel que tf représente la fréquence du terme dans le document et idf est la fréquence inverse généralement calculée comme : log(m/df) tel que m est le nombre de documents et df est le nombre de documents contenant le terme. tf indique l'importance de terme dans le document, alors que idf exprime le degré de discrimination de cet terme. De cette façon, un terme ayant une valeur élevée de tf *idf est en même temps important dans le document et moins fréquent dans les autres documents, le poids de la requête est calculé de la même manière.
- La fonction de similarité : La similarité entre un document d et une requête q est calculée en utilisant la formule suivante :
$\sum a_i \times b_i$ a_i et b_i sont les poids de terme t_i réspecitivement dans le document et dans la requête.
On peut aussi trouver d'autres formules dans la litérature comme cosinus, et jaccard [98]
- Le fichier inverse : On considère le vecteur des termes suivants$(t_1, t_2, ...t_n)$
Pour chaque terme, on considère une liste qui contient tous les documents dont ce terme l'appartient. Le poids d'un terme dans un document est associé avec ce document dans la liste. Le fichier contenant toutes les listes

des termes est appelé le fichier inverse.

7.2.2 Le processus classique de la recherche d'informations

Le processus classique de la recherche d'information est le suivant [100] : Premièrement, tous les documents sont indéxés, dans cette étape, on extrait tous les items d'un document avec ses fréquences réspecitvent dans chaque document. Puis, la base des documents est transformée en fichier inverse dont chaque terme t_i est représenté par une liste de documents. Ensuite, pour chaque couple (terme=t_i, document=d_j), on applique la formule de tf*idf et on la met dans la position (i,j). On transforme de la même manière la requête de l'utilisateur et on calcule la similarité entre la requête et chaque document dans la collection. On trie les valeurs obtenues tout en gardant les documents les plus similaires à la requête. Cependant, quand on considère un nombre important de documents comme dans le cas du web, le coût de cette approche devient important et incontrôllable. Afin de bien gérer le temps d'exécution des différentes requêtes, plusieurs approches ont été propsées, des approches basées sur le parallélisme, des approches basées sur les métaheuristiques et d'autres sur l'extraction des connaissances.

Parmi les approches d'extraction des connaissances, il y'a les approches basées sur l'extraction des règles d'association. Dans la section suivante, on détaillera sur ces approches dans le contexte de la recherche d'informations.

7.3 L'extraction des règles pour le problème de recherche d'information

Dans [101], Beil et d'autres développent le premier algorithme d'extraction des règles d'association pour le problème de recherche d'informations. Il est appelé HFTC (Hierarchical Frequent Term-based Clustering). Tout d'abord, il extrait les itemsets fréquents en utilisant l'algorithme Apriori, les itemsets sont modélisés par les termes dans la base des documents. Ensuite, il considère les itemsets les plus fréquents comme des clusters où chaque itemsets fréquent est un cluster dont il contient uniquement les documents qui vérifient cet itemset.

Fung et d'autres [102] proposent une nouvelle approche appelée FIHC (Frequent Itemset-based Hierarchical Clustering). Ils utilisent les itemsets fréquents pour construire un arbre hiérarchique de la base des documents. Ils prouvent qu'en utilisant les itemsets fréquents pour la classification des documents peut réduire le temps de calcul des requêtes de l'utilisateur.

Yu et d'autres [103] présentent un autre algorithme d'extraction des règles appelé TDC afin d'améliorer la qualité de classification des documents. Cet algorithme génère dynamiquement les différents topics de la base des documents avec seulement les itemsets fréquents fermés, celà permet de réduire efficacement le temps de calcul en comparant avec l'algorithme FIHC. TDC utilise un arbre pour construire hiérarchiquement les liens de chaque itemset de taille k avec tous les sous-ensembles de niveau *k-1*. Cette approche peut produire une forte précision, mais elle influence négativement sur la qualité des clusters, celà est dû par le faite qu'elle peut générer trop de chevauchement quand les termes d'un ensemble de documents sont fortements liés.

Dans [104], les auteurs proposent un nouveau algorithme pour la recherche des textes, une requête est modélisée par un ensemble de concepts dont les relations entre les différents concepts dans la même requête sont déterminées en appliquant le processus d'extractions des règles d'association. Dans [105], une fonction d'apprentissage pour trier les documents est proposée. Dans un premier temps, un ensemble de documents d'apprentissage est construit pour une requête donnée. Ensuite, le processus d'extraction des règles d'association est appliqué sur cet ensemble d'apprentissage. La partie conséquente de chaque règle représente les scores des documents qui contiennnent les termes de la partie antécédente de cette règle.

Un autre travail intéressant est présenté dans [106], les auteurs proposent un algorithme appelé LATRE. Il extrait des règles d'association à partir d'un ensemble de données d'apprentissage. Ces règles prévoient des mots clés d'un objet donné. Il est vu comme une étape de prétraitement de la recherche d'informations. Il fournit une téchnique pour réduire l'espace des règles en faisant une projection entre l'ensemble initial d'apprentissage et l'information fourni dans les documents tests. De plus, un ensemble d'étiquettes pertinentes est associé à chaque document ce qui diminue parfaitement le temps de réponse des différentes requêtes.

Cependant, le processus d'extraction des règles d'association génère un nombre important de règles, ce qui pénalise la phase de prétraitement de ces algorithmes. Par conséquent, nous proposons dans la section suivante une adaptation de l'algorithme PRSBSO (proposé dans le cinquième chapitre) dans le but de réduire le nombre de règles pour la recherche d'informations.

7.4 L'algorithme PRSBSO-IR

L'intention première de cette partie est de concevoir un nouveau algorithme de recherche d'informations (PRSBSO-IR) tout en appliquant l'algorithme de réduction de l'espace des règles (PRSBSO) déja proposé en cinquième chapitre. PRSBSO-IR permet d'exploiter les règles générées par le processus d'extraction des règles d'association. Il est composé principalement de trois phases : La transformation, l'extraction et la réduction des règles d'association, et le calcul des similarité.

7.4.1 La transformation

Un document d_i est représenté par l'ensemble de ces termes comme : $d_i = < t_1, t_2, ..., t_n >$, on peut transformer la base des documents à une base de transactions dont chaque transaction est un document et chaque item est un terme. De plus, chaque terme i dans le document j est écrit avec sa forme binaire dans la base de transactions, il est égal à 1 si d_j contient le terme t_i, et il est égal à 0 sinon. L'idée principale de cette phase peut être résumée dans l'algorithme suiviant :

L'algorithme de transformation

```
Debut
 Extraction de tous les termes de tous les documents
 Créer la table de document (n*m) (documents * terms)
    Pour i = 1 à n faire
       Pour j = 1 à m faire
        Mettre 1 dans table[i,j] si le document i contient le term j
```

```
        Mettre 0 sinon;
      fin_pour
    fin_pour
Fin
```

La complexité de cet algorithme au pire des cas est O(n^2).

7.4.2 l'extraction et la réduction des règles d'association

Premièrement, la base de transactions créée dans la phase précédente est divisée en plusieurs paquets comme il est présenté dans la Figure 7.1. Puis, pour chaque paquet, on applique le processus d'extraction et de réduction des règles d'association. En d'autres termes, on applique l'algorithme PRSBSO pour chaque paquet de transactions des documents. On sauvegarde par la suite les règles obtenues pour chaque paquet traité. Les règles obtenues aprés cette phase sont de la forme : $ensembledetermes \Rightarrow ensembledetermes$.

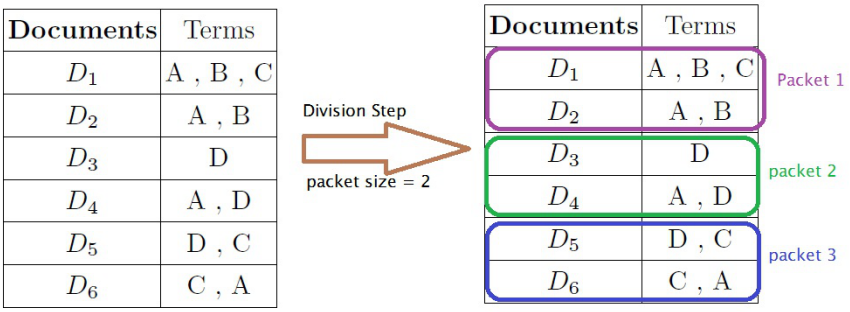


FIGURE 7.1 – Division par paquets

7.4.3 Calcul de similarité

Pour chaque nouvelle requête, une fonction de similarité entre cette dernière et les règles obtenues de chaque paquet.
L'idée est de choisir le paquet de documents qui partage un maximum de termes avec la requête. La fonction de similarité est donnée comme suit :
Score(request, $packet_i$)= $\sum \bigcap terms(requte, packet_i)$.
Tel que $terms(requte, packet_i)$ est le nombre de termes communs entre la requête

Documents	Termes
d_1	A , B , C
d_2	A , B
d_3	D
d_4	A , D
d_5	D , C
d_6	C , A

TABLE 7.1 – Table de documents

et le i^{me} paquet. Ensuite, on choisit le paquet qui maximise cette fonction et on applique le processus classique de recherche d'information afin de trouver l'ensemble de documents qui répondent à la requête demandée.

7.4.4 Exemple de l'algoritme PRSBSO-IR

La Table 7.1 présente un exemple de documents et termes. En fixant la taille de chaque paquet à 3 et un minimum support est égal à 50%. Les règles suivantes sont extraites à partir des trois paquets en appliquant l'algorithme PRSBSO :

A partir du premier paquet, on obtient trois règles :

$A \rightarrow B$.
$A \rightarrow C$.
$B \rightarrow C$.
A partir du deuxième paquet, on obtient :
$A \rightarrow D$.
A partir du troisième paquet, on aura :
$D \rightarrow C$.
$C \rightarrow A$.

A l'arrivée d'une requête donnée par exemple cette requête : $req =< A, B >$, on calcule la similarité entre la requête et chaque paquet :
$Score(req, packet1) = 2 + 1 + 1 = 4$.
$Score(req, packet2) = 1$.
$Score(req, packet3) = 1$.

On remarque que la valeur maximale est donnée par le premier paquet. Par

conséquent, on applique l'algorithme classique de recherche d'informations seulement pour les documents du premier paquet c'est à dire on recherche dans le premier et le deuxième document. Contrairement à l'algorithme classique précédent où tous les documents doivent être explorés.

7.5 Expérimentation

Plusieurs expérimentations ont été élaborées pour valider l'efficacité de l'algorithme PRSBSO-IR. L'algorithme est implémenté en C++ dans Pentium (I3,4GO). Les premiérs tests ont été faits afin de règler le nombre de paquets de l'algorithme PRSBSO-IR. Par la suite, une comparaison entre PRSBSO-IR et l'algorithme classique de recherche d'informations est établie.
Pour évaluer l'algorithme PRSBSO-IR, on le compare avec l'algorithme classique de recherche d'informations. En effet, on calcule son efficacité par rapport à l'algorithme classique. En d'autres termes, on vérifie le score des documents retournés par PRSBSO-IR et les documents retournés par l'algorithme classique. Plus Formellement, on suppose que l'algorithme classique et notre algorithme donnent deux vecteurs (Classic-Rel et PRSBSO-IR-Rel) comme :
Classic-Rel[i]= le score de i^{me} document de la collection est obtenu par l'algorithme classique.
PRSBSO-IR-Rel[i]=le score de i^{me} document de la collection est obtenu par l'algorithme PRSBSO-IR.
En outre, l'efficacité de l'algorithme PRSBSO-IR est calculée comme suit :
$Efficacit = (Nbr_1/Nbr_2) \times 100$.
Nbr_1 : le nombre de documents qui ont le même score non nul dans les deux vecteurs.
Nbr_2 : le nombre total des documents qui ont un score non nul.
Par exemple, si on considère une collection de 5 documents et les deux vecteurs :
Classic-Rel={0, 2, 5, 0,6}
PRSBSO-IR-Rel={0, 2, 4, 0, 6}
l'efficacité est égale au :
$Efficacit = (2/3) \times 100$=66%.
La Table 7.2 montre le temps d'exécution en secondes et l'efficacité de l'al-

Paquets	Temps d'exécution(Sec)	Efficacité(%)
1	0.092	100
2	0.053	100
3	0.031	99
4	0.023	75
5	0.018	60
6	0.016	50
7	0.014	43
8	0.012	38
9	0.01	33
10	0.009	30

TABLE 7.2 – le temps d'exécution (Sec) et l'efficacité (%)de l'algorithme PRSBSO-IR

N	classical-IR (Sec)	PRSBSO-IR (Sec)	Efficacité(%)
100	0.001	0.005	100
1000	0.002	0.001	100
10000	0.0009	0.005	100
100000	0.092	0.048	100
1000000	0.941	0.473	100

TABLE 7.3 – PRSBSO-IR Vs Classical-IR

gorithme PRSBSO-IR en fonction du nombre de paquets. D'aprés la table, on remarque qu'il y'a un compromis entre le temps d'exécution et l'efficacité. En effet, en augmentant le nombre de paquets, le temps d'exécution dimunue aussi. Celà est dû par le fait qu'en augmentant le nombre de paquets, on aura peu de documents à traiter ce qui diminue parfaitement le temps de recherche des documents pertinents. De contre partie, on perd les documents des autres paquets ce qui influe dans l'efficacité.
On remarque aussi que l'efficacité est égale à 100% lorsque le nombre de paquets est égal à 2. Pour celà, on choisit le nombre de paquets égal à 2 pour la prochaine expérimentation.

La Table 7.3 présente le temps d'exécution des deux algorithmes Classical-IR et PRSBSO-IR ainsi l'efficacité de PRSBSO-IR en fonction du nombre de documents dans la collection et avec un nombre de paquets qui est égal à 2.

D'aprés cette table, on remarque que quelque soit la taille de la collection, l'efficacité de notre algorithme est égale à 100% grâce aux informations ajoutées par le processus d'extraction et de réduction des règles d'association. On remarque aussi que le temps d'exécution de notre algorithme est meilleur que l'algorithme classique. On peut expliquer celà par le faite que dans l'algorithme classique, toute la collection est considérée lors de l'exécution des requêtes tandis que notre algorithme considère juste une partie de documents.

7.6 Conclusion

Dans ce chapitre, on a vu l'intégration de l'algorithme PRSBSO dans une application pratique tel que la recherche d'informations. La collection des documents est divisée en plusieurs paquets puis un processus d'extraction et de réduction des règles d'association est appliqué pour chaque paquet, a la fin on considère que le paquet des documents qui maximise le score avec la requête dans le processus de recherche des documents pertinents.
L'algorithme PRSBSO-IR est meilleur en terme de temps d'exécution par rapport à l'algorithme classique de recherche d'informations, cependant, son efficacité diminue tout en augmentant le nombre de paquets. Pour celà et comme perspective de ce chapitre, on prévoit d'implémenter une approche efficace de division de la collection afin de bien regrouper les documents les plus similaires dans le même paquet.

Conclusion générale

Bilan

Durant cette thèse, nous avons traité une représentation de connaissances de type règles d'association. Dans un premier temps, nous avons fait un état de l'art sur les algorithmes d'extraction et fouille des règles d'association. La complexité de calcul des algorithmes d'extraction des règles d'association existants nous a conduit à penser deux nouvelles approches basées sur les essaims d'abeilles.

Dans la première approche, la détermination des régions de chaque abeille est réalisée par une simple recherche locale, tandis que pour la deuxième approche, le calcul de voisinage est déterminé avec la recherche tabou pour que chaque abeille explore profondément sa région de recherche. Ces deux approches sont plus performants par rapport à l'état de l'art des algorithmes d'extraction des règles d'association en terme de temps du calcul.

Cependant, en traitant une base de transactions comme Webdocs qui contient plus d'un million de transactions, ces approches se bloquent aprés $15 jours$ de déroulement. Pour celà et grâce à la téchnologie GPU, nous nous sommes finalement arrivés à fouiller les données de WebDocs dans seulement 10 jours. En effet, Nous avons bénéficie du calcul intensif de GPU pour évaluer les solutions générées dans CPU, en proposant deux autres approches. La première approche traite solution par solution en GPU, alors que la deuxième approche traite un ensemble de solutions à la fois en GPU.

Nous avons proposé aussi deux algorithmes de fouille des règles. Le premiér

algorithme appelé Kmeans-AR, c'est une extension de l'algorithme Kmeans dans le but est d'améliorer la maniére de groupement des règles d'association. Le deuxième algorithme extrait des meta-règles à partir de l'ensemble des règles délivrées. Ces meta-règle aident à résumer et à déduire l'espace des règles. Nous avons testé nos algorithmes de fouille des règles sur des benchmarks connus, nous avons constaté que nos algorithmes sont plus performants par rapport aux algorithmes existants en terme de qualité des règles obtenues et temps d'exécution en se basant sur les approches propsées pour l'extraction des règles d'association.

Afin de bien valider les approches d'extraction et de fouille des règles d'association, nous avons élargie nos approches dans des applications pratiques.

D'une part, nous avons utilisés l'algorithme BSO-ARM pour le groupement des clauses de SAT où chaque règle de 100% de confiance peut former un cluster de clauses. Notre approche des règles d'association est plus efficace que l'algorithme BSO-DM déja proposé dans la littérature en terme de qualité des clusters obtenus. Le temps d'exécution des deux algorithmes est presque le même tout en augmentant le nombre de clauses à grouper.

D'autre part, nous avons intégré notre algorithme de fouille et réduction des règles (PRSBSO) dans le problème de recherche d'informations. Le but de l'algorithme est d'extraire le petit ensemble de règles représentatives d'un paquet de documents. Ensuite, lors de l'exécution d'une requête donnée, seulement le paquet qui maximise le score avec la requête est utilisé. Cette approche donne de meilleurs résultats en comparaison avec l'approche classique de recherche d'informations en terme de temps de réponse.

Perspectives

Les perspectives de cette thèse sont comme suit :

- Appliquer les algorithmes proposés dans le contexte de big data, comme dans le cas des réseaux sociaux(Twitter, Facebook, ...ect).
- Implémenter les algorithmes parallèles proposés en d'autres architectures comme les clusters et les supers-calculateurs.

– Intégrer les connaissances de domaines dans le processus de fouille des règles d'association.

Liste des Publications

Conférences Internationales

Djenouri, Y., Drias, H., Habbas, Z., Mosteghanemi, H. (2012, December). Bees Swarm Optimization for Web Association Rule Mining. In Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on (Vol. 3, pp. 142-146). IEEE.

Djenouri, Y., Drias, H., Chemchem, A. (2013, August). A hybrid Bees Swarm Optimization and Tabu Search algorithm for Association rule mining. In Nature and Biologically Inspired Computing (NaBIC), 2013 World Congress on (pp. 120-125). IEEE.

Djenouri, Y., Drias, H., Habbas, Z., Chemchem, A. (2013, April). Organizing association rules with meta-rules using knowledge clustering. In Programming and Systems (ISPS), 2013 11th International Symposium on (pp. 109-115). IEEE.

Revues Internationales

Djenouri, Y., Drias, H., Habbas, Z., Bees Swarm Optimization Using Multiple Strategies For Association Rule Mining, International Journal of Bio-Inspired Computation.

Djenouri, Y., Drias, H., Habbas, Z. Hybrid Intelligent Method for Association Rules Mining Using Multiple Strategies, International Journal of Metaheuristic and Applied Computing

References

[1] Olafsson, S., Li, X., & Wu, S. (2008). Operations research and data mining. European Journal of Operational Research, 187(3), 1429-1448.

[2] Martínez-Ballesteros, M., Nepomuceno-Chamorro, I. A., & Riquelme, J. C. (2013). Discovering gene association networks by multi-objective evolutionary quantitative association rules. Journal of Computer and System Sciences.

[3] Faria, F. F., Veloso, A., Almeida, H. M., Valle, E., Torres, R. D. S., Gonçalves, M. A., & Meira Jr, W. (2010, March). Learning to rank for content-based image retrieval. In Proceedings of the international conference on Multimedia information retrieval (pp. 285-294). ACM.

[4] Han, J., Kamber, M., & Pei, J. (2006). Data mining : concepts and techniques. Morgan kaufmann.

[5] Romero, C., Romero, J. R., Luna, J. M., & Ventura, S. (2010). Mining Rare Association Rules from e-Learning Data. In EDM (pp. 171-180).

[6] Lenca, P., Meyer, P., Vaillant, B., & Lallich, S. (2008). On selecting interestingness measures for association rules : User oriented description and multiple criteria decision aid. European Journal of Operational Research, 184(2), 610-626.

[7] Hegland, M. (2005). The apriori algorithma tutorial. Mathematics and computation in imaging science and information processing, 11, 209-262.

[8] Fang, Wenbin, et al. "Frequent itemset mining on graphics processors." Proceedings of the fifth international workshop on data management on new hardware. ACM, 2009.

[9] Agrawal, Rakesh, Tomasz Imielinski, and Arun Swami. "Mining association rules between sets of items in large databases." ACM SIGMOD Record. Vol. 22. No. 2. ACM, 1993.

[10] Han, Jiawei, Jian Pei, and Yiwen Yin. "Mining frequent patterns without candidate generation." ACM SIGMOD Record. Vol. 29. No. 2. ACM, 2000.

[11] Brin, Sergey, et al. "Dynamic itemset counting and implication rules for market basket data." ACM SIGMOD Record. Vol. 26. No. 2. ACM, 1997.

[12] Park, Jong Soo, Ming-Syan Chen, and Philip S. Yu. An effective hash-based algorithm for mining association rules. Vol. 24. No. 2. ACM, 1995.

[13] Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., ... & Steinberg, D. (2008). Top 10 algorithms in data mining. Knowledge and Information Systems, 14(1), 1-37.

[14] Savasere, A., Omiecinski, E. R., & Navathe, S. B. (1995). An efficient algorithm for mining association rules in large databases.

[15] Mueller, A. (1998). Fast sequential and parallel algorithms for association rule mining : A comparison.

[16] Zaki, Mohammed Javeed, Srinivasan Parthasarathy, and Wei Li. "A localized algorithm for parallel association mining." Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures. ACM, 1997.

[17] Amphawan, K., and A. Surarerks. "An Approach of Frequent Item Tree for Association Generation." Artificial Intelligence and Soft Computing Ninth IASTED International Conference Proceedings. 2005.

[18] Mata, J., Alvarez, J. L., & Riquelme, J. C. (2001). Mining numeric association rules with genetic algorithms. In 5th Internacional Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA (pp. 264-267).

[19] Mata, J., Alvarez, J. L., & Riquelme, J. C. (2002, March). An evolutionary algorithm to discover numeric association rules. In Proceedings of the 2002 ACM symposium on Applied computing (pp. 590-594). ACM.

[20] Yan, X., Zhang, C., & Zhang, S. (2009). Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. Expert Systems with Applications, 36(2), 3066-3076.

[21] Wang,M., Zou,Q. and Lin, C. (2011) Multi dimensions association rules mining on adaptive genetic algorithm. International conference on uncertainly reasoning on knowledge engineering IEEE

[22] Liu, D. (2010) Improved genetic algorithm based on simulated annealing and quantum computing strategy for association rule mining. Journal of software

[23] Hong, G. and Zhou, Y. (2009) An algorithm for mining association rules based on improved genetic algorithm and its application. Third international conference on genetic and evolutionary computing, IEEE computer science, pp117-120

[24] Indira, K., & Kanmani, S. (2012). Performance Analysis of Genetic Algorithm for Mining Association Rules. International Journal of Computer Science Issues(IJCSI), 9(2).

[25] Romero, C., Zafra, A., Luna, J. and Ventura, S. (2012) Association rule mining using genetic programming to provide feedback to instructors from multiple-choice quiz data. Journal of Expert System

[26] Kuo R.J., Chao C.M., Chiu Y.T. (2011) Application of particle swarm optimization to association rule mining. Journal of Applied Soft Computing, pp326 ?336.

[27] Parisa M., Behrouz M., Mahdi N., Afshin S. (2011) Multi-objective Numeric Association Rules Mining via Ant Colony Optimization for Continuous Domains without Specifying Minimum Support and Minimum Confidence. International Journal of Computer Science Issues, Vol8, No1

[28] Fariba K., Baraani A., Kamran Z. (2011) Efficient mining of association rules based on gravitational search algorithm. International Journal of Computer Science Issues, Vol8, No2

[29] Garg, R., & Mishra, P. K. (2011). Exploiting Parallelism in Association Rule Mining Algorithms. International Journal of Advancements in Technology, 2(2), 222-232.

[30] Garg, R., & Mishra, P. K. (2009, March). Some Observations of Sequential, Parallel and Distributed Association Rule Mining Algorithms. In Computer and Automation Engineering, 2009. ICCAE'09. International Conference on (pp. 336-342). IEEE.

[31] Zaki, M. J. (1999). Parallel and distributed association mining : A survey. Concurrency, IEEE, 7(4), 14-25.

[32] Agrawal, Rakesh, and John C. Shafer. "Parallel mining of association rules." Knowledge and Data Engineering, IEEE Transactions on 8.6 (1996) : 962-969.

[33] Boley, Daniel, et al. "Partitioning-based clustering for web document categorization." Decision Support Systems 27.3 (1999) : 329-341.

[34] Sun, Jinguang, and Weihao Pan. "Parallel association rule mining for image data." International Conference on Photonics and Image in Agriculture Engineering (PIAGENG 2009). International Society for Optics and Photonics, 2009.

[35] Park, Jong Soo, Ming-Syan Chen, and Philip S. Yu. "Efficient parallel data mining for association rules." Proceedings of the fourth international conference on Information and knowledge management. ACM, 1995.

[36] Xue-li, S., & Tao, L. (2010, April). Association rules parallel algorithm based on FP-tree. In Computer Engineering and Technology (ICCET), 2010 2nd International Conference on (Vol. 4, pp. V4-687). IEEE.

[37] Parthasarathy, Srinivasan, et al. "Parallel data mining for association rules on shared-memory systems." Knowledge and Information Systems 3.1 (2001) : 1-29.

[38] Zaïane, Osmar R., Mohammad El-Hajj, and Paul Lu. "Fast parallel association rule mining without candidacy generation." Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on. IEEE, 2001.

[39] Yang, J., & Yang, Y. (2010, May). A parallel algorithm for mining association rules. In Networking and Digital Society (ICNDS), 2010 2nd International Conference on (Vol. 1, pp. 475-478). IEEE.

[40] Zhou, Jiayi, Kun-Ming Yu, and Bin-Chang Wu. "Parallel frequent patterns mining algorithm on GPU." Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on. IEEE, 2010.

[41] Adil, Syed Hasan, and Sadaf Qamar. "Implementation of association rule mining using CUDA." Emerging Technologies, 2009. ICET 2009. International Conference on. IEEE, 2009.

[42] Silvestri, Claudio, and Salvatore Orlando. "gpudci : Exploiting gpus in frequent itemset mining." Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on. IEEE, 2012.

[43] Orlando, Salvatore, et al. "Adaptive and resource-aware mining of frequent sets." Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on. IEEE, 2002.

[44] Cui, Qingmin, and Xiaobo Guo. "Research on Parallel Association Rules Mining on GPU." Proceedings of the 2nd International Conference on Green Communications and Networks 2012 (GCN 2012) : Volume 2. Springer Berlin Heidelberg, 2013.

[45] Zhang, Fan, Yan Zhang, and Jason Bakos. "Gpapriori : Gpu-accelerated frequent itemset mining." Cluster Computing (CLUSTER), 2011 IEEE International Conference on. IEEE, 2011.

[46] Li, Haifeng, and Ning Zhang. "Mining maximal frequent itemsets on graphics processors." Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on. Vol. 3. IEEE, 2010.

[47] Zhou, Jiayi, Kun-Ming Yu, and Bin-Chang Wu. "Parallel frequent patterns mining algorithm on GPU." Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on. IEEE, 2010.

[48] Luke, S. Essentials of meta-heuristics. pp11-52 (2011).

[49] Melab, Nordine, and El-Ghazali Talbi. "A parallel genetic algorithm for rule mining." Proceedings of the 15th International Parallel & Distributed Processing Symposium. IEEE Computer Society, 2001.

[50] Khabzaoui, Mohammed, Clarisse Dhaenens, and El-Ghazali Talbi. "Parallel Genetic Algorithms for multi-objective rule mining." MIC2005. The 6th (2005).

[51] Mishra, B. S. P., et al. "Parallel multi-objective genetic algorithms for associative classification rule mining." Proceedings of the 2011 International Conference on Communication, Computing & Security. ACM, 2011.

[52] Pei, J., Han, J., & Mao, R. (2000, May). CLOSET : An Efficient Algorithm for Mining Frequent Closed Itemsets. In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (Vol. 4, No. 2, pp. 21-30).

[53] Burdick, D., Calimlim, M., & Gehrke, J. (2001). MAFIA : A maximal frequent itemset algorithm for transactional databases. In Data Engineering, 2001. Proceedings. 17th International Conference on (pp. 443-452). IEEE.

[54] Hájek, P. (2001). The GUHA method and mining association rules. Proc. CIMA, 533-539.

[55] REYNOLDS, Alan P., RICHARDS, Graeme, DE LA IGLESIA, Beatriz, et al. Clustering rules : a comparison of partitioning and hierarchical clustering algorithms. Journal of Mathematical Modelling and Algorithms, 2006, vol. 5, no 4, p. 475-504.

[56] Gray, B., & Orlowska, M. E. (1998). CCAIIA : Clustering categorical attributes into interesting association rules. In Research and Development in

Knowledge Discovery and Data Mining (pp. 132-143). Springer Berlin Heidelberg.

[57] Strehl, A., Gupta, G. K., & Ghosh, J. (1999). Distance based clustering of association rules. Proceedings ANNIE 1999, 9, 759-764.

[58] Hahsler, M., & Chelluboina, S. (2011, May). Visualizing Association Rules in Hierarchical Groups. In 42nd Symposium on the Interface : Statistical, Machine Learning, and Visualization Algorithms (Interface 2011). The Interface Foundation of North America.

[59] Jorge, A. (2004, April). Hierarchical clustering for thematic browsing and summarization of large sets of association rules. In Proceedings of the 2004 SIAM International Conference on Data Mining.

[60] Lent, B., Swami, A., & Widom, J. (1997, April). Clustering association rules. In Data Engineering, 1997. Proceedings. 13th International Conference on (pp. 220-231). IEEE.

[61] Drias, H., Aouichat, A., & Boutorh, A. (2012). Towards Incremental Knowledge Warehousing and Mining. In Distributed Computing and Artificial Intelligence (pp. 501-510). Springer Berlin Heidelberg.

[62] MacQueen, J. (1967, June). Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, No. 281-297, p. 14).

[63] Chemchem, A., Drias, H., & Djenouri, Y. (2013). Multilevel Clustering of Induction Rules for Web Meta-knowledge. In Advances in Information Systems and Technologies (pp. 43-54). Springer Berlin Heidelberg.

[64] Dudani, S. A. (1976). The distance-weighted k-nearest-neighbor rule. Systems, Man and Cybernetics, IEEE Transactions on, (4), 325-327.

[65] Berrado, A., & Runger, G. C. (2007). Using metarules to organize and group discovered association rules. Data Mining and Knowledge Discovery, 14(3), 409-431.

[66] Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H., & Verkamo, A. I. (1994, November). Finding interesting rules from large sets of discovered association rules. In Proceedings of the third international conference on Information and knowledge management (pp. 401-407). ACM.

[67] Liu, B., Hsu, W., Wang, K., & Chen, S. (1999). Visually aided exploration of interesting association rules. In Methodologies for Knowledge Discovery and Data Mining (pp. 380-389). Springer Berlin Heidelberg.

[68] Ng, R. T., Lakshmanan, L. V., Han, J., & Pang, A. (1998, June). Exploratory mining and pruning optimizations of constrained associations rules. In ACM SIGMOD Record (Vol. 27, No. 2, pp. 13-24). ACM.

[69] Marinica, C., & Guillet, F. (2010). Knowledge-based interactive postmining of association rules using ontologies. Knowledge and Data Engineering, IEEE Transactions on, 22(6), 784-797.

[70] Mansingh, G., Osei-Bryson, K. M., & Reichgelt, H. (2011). Using ontologies to facilitate post-processing of association rules by domain experts. Information Sciences, 181(3), 419-434.

[71] ANUSHA, P., & REDDY, G. S. Interactive Postmining of Association Rules by Validating Ontologies. International Journal of Electronics, 1.

[72] Fernandes, L. A., & García, A. C. B. (2012). Association Rule Visualization and Pruning through Response-Style Data Organization and Clustering. In Advances in Artificial IntelligenceIBERAMIA 2012 (pp. 71-80). Springer Berlin Heidelberg.

[73] Watanabe, T. (2010, November). An improvement of fuzzy association rules mining algorithm based on redundacy of rules. In Aware Computing (ISAC), 2010 2nd International Symposium on (pp. 68-73). IEEE.

[74] Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., & Lakhal, L. (2001). Intelligent structuring and reducing of association rules with formal concept analysis. In KI 2001 : Advances in Artificial Intelligence (pp. 335-350). Springer Berlin Heidelberg.

[75] Hämäläinen, W. (2010). StatApriori : an efficient algorithm for searching statistically significant association rules. Knowledge and information systems, 23(3), 373-399.

[76] Liu, B., Hsu, W., & Ma, Y. (1999, August). Pruning and summarizing the discovered associations. In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 125-134). ACM.

[77] Liu, H., Liu, L., & Zhang, H. (2011). A fast pruning redundant rule method using Galois connection. Applied Soft Computing, 11(1), 130-137.

[78] Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). Swarm intelligence : from natural to artificial systems (Vol. 4). New York : Oxford university press.

[79] Dorigo, M., & Di Caro, G. (1999). Ant colony optimization : a new meta-heuristic. In Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on (Vol. 2). IEEE.

[80] Eberhart, R., & Kennedy, J. (1995, October). A new optimizer using particle swarm theory. In Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on (pp. 39-43). IEEE.

[81] Karaboga, D., & Akay, B. (2009). A survey : algorithms simulating bee swarm intelligence. Artificial Intelligence Review, 31(1-4), 61-85.

[82] Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Techn. Rep. TR06, Erciyes Univ. Press, Erciyes.

[83] Teodorovic, D., & DellOrco, M. (2005, September). Bee colony optimizationa cooperative learning approach to complex transportation problems. In Advanced OR and AI Methods in Transportation : Proceedings of 16th MiniEURO Conference and 10th Meeting of EWGT (13-16 September 2005).Poznan : Publishing House of the Polish Operational and System Research (pp. 51-60).

[84] Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S, Zaidi M (2005) The bees algorithm. Technical report, Manufacturing Engineering Centre, Cardiff University, UK

[85] Abbass, H. A. (2001). MBO : Marriage in honey bees optimization-A haplometrosis polygynous swarming approach. In Evolutionary Computation, 2001. Proceedings of the 2001 Congress on (Vol. 1, pp. 207-214). IEEE.

[86] Drias, H., Sadeg, S., & Yahi, S. (2005). Cooperative bees swarm for solving the maximum weighted satisfiability problem. In Computational Intelligence and Bioinspired Systems (pp. 318-325). Springer Berlin Heidelberg.

[87] Wedde, H. F., Farooq, M., & Zhang, Y. (2004). BeeHive : An efficient fault-tolerant routing algorithm inspired by honey bee behavior. In Ant colony optimization and swarm intelligence (pp. 83-94). Springer Berlin Heidelberg.

[88] Guvenir, H. A., & Uysal, I. (2000). Bilkent university function approximation repository. 2012-03-12]. http ://funapp. cs. bilkent. edu. tr/DataSets.

[89] Goethals, B., & Zaki, M. J. (2003). Frequent itemset mining implementations repository. This site contains a wide-variety of algorithms for mining frequent, closed, and maximal itemsets, http ://fimi. cs. helsinki. fi.

[90] Mehdi, M., Bendjoudi, A., Loukil, L., & Melab, N. (2013). Parallel GPU-accelerated metaheuristics. Designing Scientific Applications on GPUs, 183.

[91] Chemchem, A., Djenouri, Y., & Drias, H. (2013, May). Incremental induction rules clustering. In Systems, Signal Processing and their Applications (WoSSPA), 2013 8th International Workshop on (pp. 492-497). IEEE.

[92] Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. Journal of the ACM (JACM), 7(3), 201-215.

[93] Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem-proving. Communications of the ACM, 5(7), 394-397.

[94] Selman, B., Levesque, H. J., & Mitchell, D. G. (1992, July). A New Method for Solving Hard Satisfiability Problems. In AAAI (Vol. 92, pp. 440-446).

[95] Selman, B., Kautz, H., & Cohen, B. (1993). Local search strategies for satisfiability testing. Cliques, coloring, and satisfiability : Second DIMACS implementation challenge, 26, 521-532.

[96] Drias, H., & Ibri, S. (2003). Parallel ACS for Weighted Max-Sat. In Computational Methods in Neural Modeling (pp. 414-421). Springer Berlin Heidelberg.

[97] Drias, H., Hireche, C., & Douib, A. (2013, August). Datamining techniques and swarm intelligence for problem solving : Application to SAT. In Nature and Biologically Inspired Computing (NaBIC), 2013 World Congress on (pp. 200-206). IEEE.

[98] Drias, H. (2011, October). Parallel swarm optimization for web information retrieval. In Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on (pp. 249-254). IEEE.

[99] Drias, H. (2011, August). Web information retrieval using particle swarm optimization based approaches. In Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on (Vol. 1, pp. 36-39). IEEE.

[100] Salton, G., & McGill, M. J. (1986). Introduction to modern information retrieval.

[101] Beil, F., Ester, M., & Xu, X. (2002, July). Frequent term-based text clustering. In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 436-442). ACM.

[102] Fung, B. C., Wang, K., & Ester, M. (2003, May). Hierarchical document clustering using frequent itemsets. In Proceedings of SIAM international conference on data mining (pp. 59-70).

[103] Yu, H., Searsmith, D., Li, X., & Han, J. (2004, November). Scalable construction of topic directory with nonparametric closed termset mining. In Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on (pp. 563-566). IEEE.

[104] Babashzadeh, A., Daoud, M., & Huang, J. (2013). Using semantic-based association rule mining for improving clinical text retrieval. In Health Information Science (pp. 186-197). Springer Berlin Heidelberg.

[105] Veloso, A. A., Almeida, H. M., Gonçalves, M. A., & Meira Jr, W. (2008, July). Learning to rank at query-time using association rules. In Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval (pp. 267-274). ACM.

[106] Menezes, G. V., Almeida, J. M., Belém, F., Gonçalves, M. A., Lacerda, A., De Moura, E. S., ... & Ziviani, N. (2010). Demand-driven tag recommendation. In Machine Learning and Knowledge Discovery in Databases (pp. 402-417). Springer Berlin Heidelberg.

[107] Djenouri, Y., Drias, H., Habbas, Z., & Mosteghanemi, H. (2012, December). Bees Swarm Optimization for Web Association Rule Mining. In Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on (Vol. 3, pp. 142-146). IEEE.

[108] Djenouri, Y., Drias, H., & Habbas, Z., Bees Swarm Optimization Using Multiple Strategies For Association Rule Mining , International Journal of Bio-Inspired Computation.

[109] Djenouri, Y., Drias, H., & Chemchem, A. (2013, August). A hybrid Bees Swarm Optimization and Tabu Search algorithm for Association rule mining. In Nature and Biologically Inspired Computing (NaBIC), 2013 World Congress on (pp. 120-125). IEEE.

[110] Djenouri, Y., Drias, H., Habbas, Z., & Chemchem, A. (2013, April). Organizing association rules with meta-rules using knowledge clustering. In Programming and Systems (ISPS), 2013 11th International Symposium on (pp. 109-115). IEEE.

[111] Djenouri, Y., Drias, & H., Habbas, Z. Hybrid Intelligent Method for Association Rules Mining Using Multiple Strategies, International Journal of Metaheuristic and Applied Computing