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

Université Saad Dahlab Blida

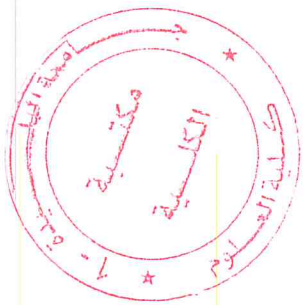
N° D'ordre :.....

Faculté des sciences

Département d'informatique

Mémoire Présenté par :

ALLOU Mohamed Amine



En vue d'obtenir le diplôme de master

Domaine : Mathématique et informatique

Filière : Informatique

Spécialité : Informatique

Option : Génie Systèmes informatique

Thème

BSO pour l'extraction des itemsets fréquents à partir des données incertaines

Soutenu le : 13/10/2016

Mme N. REZOUG	Président
Mr A.KAMECH	Examinateur
Mme ZAHRA FATMA ZOHRA	Promotrice

Promotion : 2015 / 2016

Remerciements

Nous remercions Dieu le tout puissant de nous avoir donné le courage et la volonté d'achever ce travail et sans Lequel il n'aurait jamais été accompli.

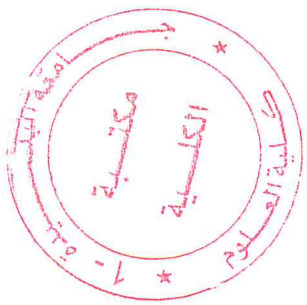
Nos remerciements les plus sincères, accompagnés de toute notre gratitude vont tout d'abord à notre promotrice M. ZAHRA FATMA ZOHRA pour nous avoir proposé ce sujet et pour ses précieux conseils et de nous avoir dirigé durant notre projet, et surtout pour la confiance qu'il nous a accordé pour la réalisation de ce projet.

Nous remercions tous les enseignants de la faculté des sciences de BLIDA et surtout ceux du département informatique.

Nous remercions les nombres de jury pour nous avoir fait l'honneur de juger notre travail.

Nous remercions également toute personne ayant contribué à notre éducation et notre formation.

Enfin, nos remerciements vont à toute personne ayant contribué, de près ou de loin, à l'aboutissement de ce travail.



Résume

L'extraction de motifs fréquents est une technique utilisée en fouille de données, elle s'appuie sur des principes relativement simples. Son objectif est de trouver des structures de données qui se répètent fréquemment dans un ensemble de données. Les itemsets représentent le type de motifs basique et le plus traité dans ce domaine. Les données réelles sont en une grande partie incertaine. En effet, on s'intéresse dans notre travail à ce genre de données. Par conséquent, notre travail consiste à proposer une méthode d'extraction des itemsets fréquents à partir des données incertaines en se basant sur une méta-heuristique connue, Optimisation par Colonies d'Abeilles (BSO, Bee Swarm Optimization) dont le but est d'optimiser le nombre des itemsets fréquents générés et de gagner en temps d'exécution et consommation de mémoire.

Mots clés : Extraction des itemsets fréquents, données incertaines, techniques d'optimisation, Optimisation par Colonies d'Abeilles.

Abstract

The extraction of frequent patterns is a technique used in data mining, it relies on relatively simple principles. Its goal is to find data structures that are frequently repeated in a data set. Itemsets represent the type of basic grounds, as addressed in this area. The actual data is in a large part uncertain. Thats why we are interested in our work to such data. So our job is to propose a method of extracting frequent itemsets from uncertain data based on a known meta-heuristic optimization by Bee Colonies (BCO, Bee Colony Optimization) whose purpose is to optimize the number of frequent itemsets generated and gain in execution time and memory consumption.

Keywords: Extraction of frequent itemsets, uncertain data, optimization techniques, optimization by Bee Colonies.

ملخص

استخراج أنماط متكررة هي التقنية المستخدمة في التنقيب عن البيانات، فإنها تعتمد على مبادئ بسيطة نسبيا. هدفها هو ايجاد هياكل البيانات التي كثيرا ما تتكرر في مجموعة البيانات. كما تناولنا تقنية التي تمثل نوع من الوحدات الأساسية في هذا المجال.

البيانات الفعلية هي جزء كبير غير مؤكد. في الواقع، ونحن مهتمون في عملنا لهذه البيانات لذلك مهمتنا هي اقتراح طريقة استخراج itemsets المتكررة من البيانات الغير مؤكدة. بناء على الكشف عن مجريات الأمور المعروفة، مستعمرات النحل الأمثل (BSO، النحلة مستعمرة الأمثل) الذي يهدف إلى تحسين عدد من itemsets المتكررة وزيادة الأداء واستهلاك الذاكرة الوقت.

Table des matières

Chapitre 4 : Solution proposé

Chapitre 5 : Test et Validation

Liste des figures

Liste des tableaux

Introduction générale

Introduction générale

1. Contexte

De nos jours, les techniques d'extraction des connaissances se sont énormément développées. Grace à cela, les bases de données volumineuses sont devenues des sources riches et fiables pour la génération et la validation de connaissances. Le Data Mining (fouille de données) est le noyau de processus d'extraction des connaissances, il couvre plusieurs domaines, l'analyse de données, les bases de données, l'apprentissage, les statistiques, les systèmes à base de règles. Il dispose d'outils performants afin de structurer et d'extraire des connaissances : la classification, la segmentation, la recherche de règle d'association, ...etc dont l'extraction des itemsets fréquents à partir des différente sources fait partie.

2. Problématique

D'autre part, l'extraction d'itemsets fréquents est une technique utilisée en fouille de données. Cette dernière s'appuie sur des principes relativement simples. Son objectif est de trouver les itemsets qui apparaissent fréquemment dans un ensemble de données. Cependant les algorithmes d'extraction des itemsets fréquents à partir d'une base de donnés traditionnelle (certaine) ne fonctionne pas avec une base de données incertaine. Cela est dû au fait qu'une base de données incertaine, plus précisément les bases de données probabilistes contiennent des données avec leurs probabilités existentielles.

L'extraction de motifs fréquents est la seule technique de Data Mining où le volume du résultat est parfois plus important que le volume des données d'entrée ce qui pose le problème de pertinence et d'interprétation des résultats ainsi que la taille de l'espace de recherche qui est d'ordre exponentiel impliquant un temps d'exécution très important.

3. Objectifs

Nous nous sommes fixés comme objectif le développement d'une méthode d'extraction des motifs fréquents a partir des données incertaines. Pour ce faire, notre travail consiste à appliquer une **Méta-heuristique bioinspirée,** basée sur les comportements des animaux (les insectes) dans le but d'extraire des itemsets fréquents à partir de données incertaines.

L'une des espèces d'insectes les plus organisées et les plus rigoureuses dans leur travail est **l'abeille**. Les abeilles possèdent une très grande capacité de communication. Et on se basant sur le principe de fonctionnement des abeilles, un algorithme appelé la Colonie

d'Abeille a été développée. C'est cet algorithme que nous avons utilisé et adapté dans notre travail.

L'algorithme d'optimisation par Colonie d'Abeille (BCO, Bee Colony Optimization) utilise un modèle probabiliste glouton pour générer des combinaisons. Ce modèle évoluant en fonction des combinaisons précédemment construites dans un processus itératif d'apprentissage. L'originalité et la contribution essentielle de BCO est de s'inspirer du comportement collectif des abeilles pour faire évoluer le modèle probabiliste.

L'objectif de notre travail est l'implémentation d'une méthode qui permet d'extraire les itemsets fréquents à partir des données incertaine en utilisant l'algorithme d'optimisation par colonies d'abeilles.

4. Organisation du mémoire

Le mémoire se réparti en cinq chapitres.

Chapitre 1 : « l'extraction des itemsets fréquents à partir des données déterministes »

Ce chapitre est consacré aux les principaux algorithmes d'extraction d'itemsets fréquents a partir des données certaines.

Chapitre 2 : « l'extraction des itemset fréquents à partir des données incertaines »

Dans ce chapitre nous allons présenter et étudier les algorithmes d'extraction à partir des donnes incertaine, en plus on va présenter le modèle probabiliste.

Chapitre 3 : «les techniques d'optimisation pour l'extraction des règles d'association»

Dans ce chapitre nous détaillerons les deux grandes techniques d'optimisation en data mining, « méta-heuristique et l'extraction des règles d'association ».

Chapitre 4 : « Solution proposée »

Ce chapitre, sera réservé pour présenter notre solution proposée et évaluation de notre application.

Chapitre 5 : « Test et Validation»

Le dernier chapitre sera pour valider la méthode adoptée, et on terminera par une conclusion générale.

Chapitre 1 :

l'extraction des itemsets fréquents à partir de données déterministes

Chapitre 1 : Extraction des itemsets fréquents à partir des données déterministes

1. Introduction

Une grande partie des algorithmes utilisés en fouille de données proviennent de l'apprentissage automatique (Machine learning). L'apprentissage statistique est un paradigme qui regroupe un ensemble de méthodes et d'algorithmes permettant d'extraire l'information pertinente à partir de données. Dans ce chapitre nous allons présenter le concept d'extraction des itemsets ainsi que, les algorithmes d'extractions des itemsets fréquents à partir des données certaines.

2. L'extraction des itemsets.

La recherche des régularités dans les bases de données est l'idée principale du data mining. Ces régularités s'expriment sous différentes formes. Dans l'analyse du panier d'achats de consommateurs, l'extraction des itemsets consiste à mettre en evidence les cooccurrences entres les produits achetés c.-à-d. Déterminer les produits (les items) qui sont « souvent » achetés simultanément. On parle alors d'itemsets fréquents. Par exemple, en analysant les tickets de caisse d'un supermarché, on pourrait produire des itemsets (un ensemble d'items) du type « le pain et le lait sont présents dans 10% des caddies » Le fichier comporte 10 observations (transactions) et 4 items (voir tableau 1.1). **[01], [02], [03].**

Tableau 1.1 : Base de transactions **[01].**

S1	S2	S3	S4
1	0	1	0
0	1	0	0
0	0	0	1
0	1	1	1
0	1	1	0
0	1	1	0
1	1	1	1
1	0	1	0
1	1	1	0
1	1	1	0

Item : Un item correspond à un produit. Nous avons 4 items (S1, S2, S3 et S4) dans notre fichier.

Support : Le support d'un item est égal au nombre de transactions dans lesquelles il apparaît.

Itemset : Un itemset est un ensemble d'items. Le support d'un itemset comptabilise le nombre de transactions dans lesquelles les items apparaissent simultanément. Un itemset peut être composé d'un singleton.

Itemset fréquent :Un itemset est dit fréquent si son support est supérieur à un seuil défini à l'avance, paramètre de l'algorithme de recherche.

Superset : Un supersetest un itemset défini par rapport à un autre itemset.

Itemset fermé (closeditemset) :Un itemset fréquent est dit fermé si aucun de ses supersets n'a de support identique. Autrement dit, tous ses supersets ont un support strictement plus faible.

Itemset maximal (maximal itemset). Un itemset est dit maximal si aucun de ses supersets n'est fréquent.

Itemset générateur (generator itemset) : Un itemset A est dit générateur s'il n'existe aucun itemset B tel que B $\subset$A et que SUP(B) = SUP(A). Autrement dit, l'itemset est générateur si tous ses sous itemsets ont un support strictement supérieur. **[01]**

3. Les algorithmes d'extraction des itemsets fréquent

Dans cette section, nous allons présenter les lignes directrices des algorithmes les plus importants parus dans la littérature, en nous focalisant essentiellement sur l'étape de découverte des itemsets. Un premier sur vol de ces algorithmes permet de les classer selon la technique adoptée pour l'exploration de l'espace de recherche, à savoir « Tester-et-générer » et « Diviser-pour-régner ».

- **La technique « Tester-et-générer » :** les algorithmes parcourent l'espace de recherche par niveau. A chaque niveau k, un ensemble de candidats de taille k est généré. Cet ensemble de candidats est, généralement, élagué par la conjonction d'une métrique statistique (e.g. le support) et des heuristiques basées essentiellement sur les propriétés structurelles des itemsets.

- **La technique « Diviser-pour-régner » :**les algorithmes essaient de diviser le contexte d'extraction en des sous-contextes et d'appliquer le processus de

découverte des itemsets récursivement sur ces sous-contextes. Ce processus de découverte repose sur un élagage du contexte basé essentiellement sur l'imposition d'une métrique statistique et d'heuristiques introduites.

La technique la plus utilisé c'est la technique « Tester-et-générer » mais dans cette technique sur une base de données dense provoque le problème de goulot d'étranglement, à savoir la génération d'un nombre prohibitif de candidats, et sont très gourmands en ressources mémoire et aussi le temps d'exécution est un peu long. Pour pallier à cette faiblisse de la technique précédente, des auteurs ont proposé la technique « Diviser-pour-régner » dans cette technique on va éviter le problème de goulot d'étranglement et de perde de l'espace mémoire et du temps d'exécution moins long que la technique précédente. Bref, les algorithmes qui sont implémenté dans la technique Divisé-pour-régner sont plus performant qui sont implémenté en Tester-et-générer.

3.1. Les algorithmes de type « Tester-et-générer »

Dans cette sous-section, nous allons passer en revue l'algorithme le plus connu dans la littérature opérant selon la technique « Tester-et-générer ». Avant d'aborder la description de l'algorithme, nous allons présenter une structure générale ou générique des algorithmes entrants dans cette catégorie.

Notons que la structure générique, présentée par l'algorithme 1.1, indique globalement les différentes étapes que suit un algorithme pour l'extraction des itemsets **[01]**.

3.1.1. Algorithme Apriori

L'algorithme Apriori **[03,04]** ont été proposés indépendamment en 1994 dans le domaine de l'apprentissage des règles d'association introduit par *Agrawal et al* dans « *Mining Association Rulesbetween Sets of Items in Large Databases*» **[05]**.Ces algorithmes, qui procèdent de la même manière, sont des algorithmes itératifs de recherche des itemsets fréquents par niveaux. Cela signifie que durant la $k^{ème}$ itération, un ensemble d'itemsets candidats de taille k est généré et un balayage du contexte est réalisé afin de supprimer les candidats infréquents. L'ensemble des k-itemsets fréquents ainsi générés est utilisé lors de l'itération $k + 1$ suivante pour générer les candidats de taille $k + 1$. Ces algorithmes réalisent donc μ itérations afin de

déterminer tous les itemsets fréquents dans l'ordre croissant de leurs tailles, μ étant la taille des plus grands itemsets fréquents.

Le pseudo-code de l'algorithme est présenté dans l'algorithme 1.1. Les notations utilisées sont présentées dans la table 1.2.

C_k	Ensemble de *k*-itemsets candidats (itemsets potentiellement fréquents). Chaque élément de cet ensemble possède deux champs : *itemset* et *support*.
F_k	Ensemble de *k*-itemsets fréquents. Chaque élément de cet ensemble possède deux champs : *itemset* et *support*.

Tableau 1.2- Notions utilisées dans l'algorithme Apriori.

Procédure Apriori-Gen(F_{k-1}) La procédure Apriori-Gen reçoit un ensemble F_{k-1} de (*k*-1)- itemsets fréquents comme paramètre. Elle retourne un ensemble C_k de *k*-itemsets candidats qui est un sur-ensemble de l'ensemble des k-itemsets fréquents.

Algorithme 1.1 :Algorithme Apriori.

Entrée : contexte β ; seuil minimale de support *minsupport* ;
Sortie : ensembles F_k des *k*-itemsets fréquents ;

1) $F_1 \leftarrow$ {1-itemsets fréquents} ;
2) **Pour** (k ←2 ; $F_{k-1} \neq \emptyset$; k++) **faire**
3) $C_k \leftarrow$Apriori-Gen(F_{k-1}) ;
4) **Pour chaque** objet o c **β faire**
5) $C_o \leftarrow$Subset (C_k ,o) ;
6) **Pour chaque** candidat c c C_o **faire** *c.support* ++ ;
7) **Fin pour**
8) $F_1 \leftarrow$ {c c C_k| *c.support* ≥ *minsupport* ;
9) **Fin pour**
10) **Retourner** U_k F_k ;

3.2. Les algorithmes de type « diviser-pour-régner»

3.2.1. Algorithme FP- growth

L'algorithme FP-tree apporte ainsi une solution au problème de la fouille de motifs fréquents dans une grande base de données transactionnelle. En stockant l'ensemble des éléments fréquents de la base de transactions dans une structure compacte, on supprimer la nécessité de devoir scanner de façon répétée la base de

transactions. De plus, en triant les éléments dans la structure compacte, on accélère la recherche des itemsets.**[06]**

Cette méthode s'appelle FP-tree (*FrequentPattern growth*). Elle consiste d'abord à compresser la base de données en une structure compacte appelée FP-tree (*Frequent Pattern tree*), puis à diviser la base de donnees ainsi compressée en sous projections de la base de données appelées bases conditionnelles.**[07]**

Algorithme 1.3 :Algorithme FP-Growth

1. Balayer la base de transactions T une première fois
   - Créer L, la liste des items fréquents avec leur support
   - Trier L en ordre décroissant du support
2. Créer l'arbre N contenant une racine étiquetée « Null »
3. Procédure FP-Growth(Fp-tree, Null)
   a) Si FP-tree contient un seul chemin P alors
      - Pour chaque combianaison β de P faire
      - Générer l'itemset BUα de support = minimum de supports des nœuds β
   b) Sinon pour chaque a dans l'index de FP-tree faire
      - Générer l'itemset $\beta = a_i$ U de support = ai.support
      - Construire l'itemset condition de base de β
      - Construire $Fp\text{-tree}_\beta$
      - Si $FP\text{-tree}_\beta \neq 0$
        - Fp-growth($FP\text{-tree}_\beta$,β)

4. Conclusion

L'étude approfondie des algorithmes d'extraction d'itemsets fréquents à partir des données certaines. Dans le prochain chapitre nous allons présenter les déférents algorithmes utilisés pour extraire les motifs fréquents à partir des données incertaines.

Chapitre 2 : l'extraction des itemsets fréquents à partir de données incertaines

Chapitre 2 : Extraction des itemsets fréquents à partir de données incertaines

1. Introduction

En informatique, les données incertaine est la notion de données qui contient le bruit qui fait dévier des valeurs correctes. À l'ère de Big Data, l'incertitude ou de la véracité des données est l'une des caractéristiques déterminantes. Les données sont en constante augmentation dans le volume, la variété, de la vitesse et de l'incertitude. Dans ce chapitre on va présenter les algorithmes qui permettent d'extraire des itemset fréquents a partir des données incertaines.

2. Les donnes incertaines

Une valeur sera déclarée incertaine si la validité de la donnée reste douteuse au stade de validation indiquée dans l'information.

Dans la mesure du possible, la qualification douteuse doit être une étape transitoire de la validation de la donnée et doit être réservé à des avancements intermédiaires de la validation. **[08]**

C'est pour cela les algorithmes d'extractions des itemsets fréquents a partir des données incertaines utilisent deux mesures pour calculer les itemsets fréquents :

- Expected Support
- Probabiliste Support

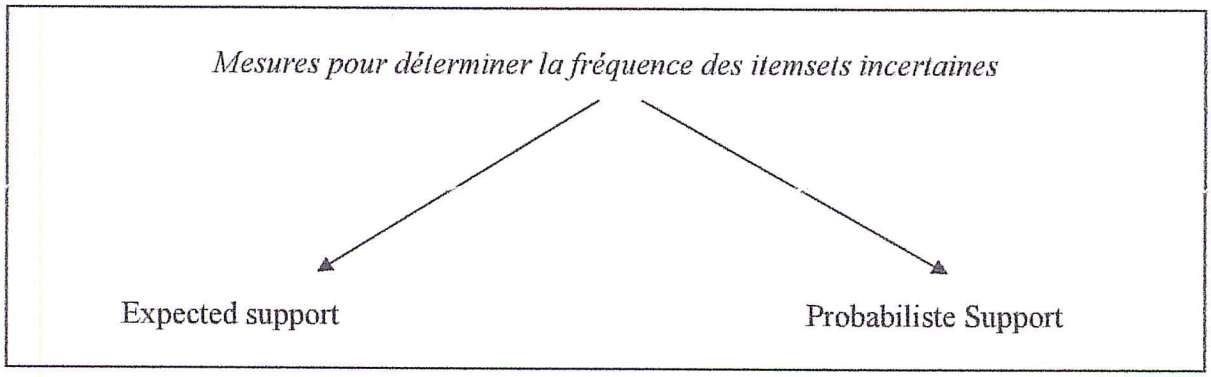


Figure 2.1: Les mesures de calcul des itemsets fréquentes.

2.1. Expected support

l'itemset X est un fréquent si son expected support supérieur ou égale le seuil minsup.

$$expSup\ (X) >= minsup \qquad (2.1)$$

Définition :

Expected support d'un itemests X dans la base de données incertaine est la somme de produit probabilité P (X, t j) de X dans la transaction t j sur tous les n transactions dans la base de données.

Donc l'expression de la fonction Expected Support est présentée comme suit :

$$expSup(X) = \sum_{j=1}^{n} P(X, t_j) = \sum_{j=1}^{n} \left(\prod_{x \in X} P(x, t_j) \right) \qquad (2.2)$$

Lorsque des éléments x ∈ X dans chaque transaction t j sont indépendants.

2.2. Probabiliste Support [09]

Un itemset X est un fréquent probabiliste si son existence dans une transaction *minsup* est supérieure ou égale le seuil d'utilisateur spécifique *minprob*

$$P(sup(X) >= minsup) >= minprob \qquad (2.3)$$

Dans les bases de données de transactions incertaines, le support d'un ou plusieurs éléments ne peut pas être représenté par une valeur unique, mais plutôt, doit être représenté par une distribution de probabilité discrète.

Soit T est la base de données de transaction et W est l'ensemble des éléments possibles de T, le $P_i(X)$ est la probabilité de support d'un itemset X tel que X a le soutien i.

$$P_i(X) = \sum_{w_j \in W, (S(X, w_j) = i)} P(w_j) \qquad (2.4)$$

Où S (X; wj) est le support de X dans un élément wj.

Le support probabiliste d'un itemset X dans une base de données de transaction incertaine T est donné par les probabilités de support de X ($P_i(X)$) pour toutes les valeurs possibles de soutien i $\in\{0,\ldots..,|T|\}$.

$$\sum_{0\leq i\leq|T|} P_i(X) = 1.0. \quad (2.5)$$

Soit I l'ensemble de tous éléments possibles et T représente la base de données incertaines, où une transaction tj ∈ T est un ensemble des éléments incertains, à savoir, tj ⊆ I. Contrairement à la base de données précise traditionnelle, chaque item xi ∈ tj est associée à une probabilité existentielle P (xi, tj) ∈ (0,1], ce qui dénote la probabilité que xi est réellement présent dans tj. Pour un itemset X ⊆ tj, basée sur l'hypothèse commune que les éléments de X sont indépendante la probabilité existentielle P (X ⊆ tj) =∏ x∈XP (x, tj). le soutien attendu (expected support) de X est alors la somme de la probabilité existentielle sur toutes les transactions. [10, 11,12],

$$expSup\ (X) = \sum tj \in TP\ (X \subseteq tj). \quad (2.6)$$

La probabilité fréquente P (X) de X peut être calculé par sommer Pi (X) pour tout i ≥ minsup :

$$P(X) = \sum_{i=minsup}^{|T|} P_i(X) \quad (2.7)$$

où Pi (X) enregistre la probabilité de X qui se produit exactement dans une transaction:

$$P_i(X) = \sum_{\substack{S\subseteq T \\ |S|=i}} \left(\prod_{t\in S} P(X\subseteq t) \prod_{t\in T-S} (1 - P(X\subseteq t)) \right) \quad (2.8)$$

Si P (X) ≥ minprob, alors X est un itemset fréquent probabiliste.

On donne (i) une base de données de transaction incertaine T, (ii) un seuil minimum de support minsup et (iii) un seuil minimum de la probabilité frequente minprob, le problème d'extraction de motifs fréquents probabilistes est de trouver tous et seuls les motifs fréquents probabilistes; à savoir, trouver tous les X tel que P (X) ≥ minprob.

3. Les algorithmes d'extraction des itemsets

Lorsqu'on manipule les données incertaines, on peut utiliser par exemple U-Apriori, UF-growth, FP- growth et PUF- growth qu'ils sont les algorithmes d'extraction les plus connus

où utilisent l'UF-arbre, l'UFP-arbre et le PUF-arbre respectivement.ces arbres peuvent être large et ainsi diminuent la performance d'extraction.les chercheurs proposés des divers algorithmes comme U-Apriori, UH-manier, UF- growth etc. dans cette section nous allons présentés une analyse pour ces algorithmes.

Il existe deux approches des algorithmes d'extraction des itemsets fréquents à partir des données incertaines **[11]** :

Tableau 2.1 : Algorithmes d'extraction des itemstes fréquents

Algorithmes basés sur «Tester-et-générer»	Algorithmes basés sur « diviser-et-régner»
U-Apriori	UF-growth UFP- growth PUF-growth UH - mine

3.1. Les algorithmes de type « Tester-et-générer »

3.1.1. Algorithme U-Apriori :

U-Apriori est la version incertaine de Apriori algorithme Pour faire face au problème des itemsets fréquentes dans les données incertaines. Apriori a été modifié à l'algorithme U-Apriori. R.Agrawa la été le premier qui avait proposé cette Apriori algorithme **[12].**

Étape-1: Scannez la base de données de transaction pour obtenir le support S de chaque1-itemset, comparer avec Soutien minimum (MS), et d'obtenir un ensemble de fréquentes1-itemsets, L1.

Etape-2: Utiliser la propriété Apriori à tailler les ensembles non fréquentés k-point de cet ensemble.

Étape-3: Scannez la base de données de transaction pour obtenir le support S de chaque k-point de l'ensemble candidat dans le dernier set, comparer S avec MS, et obtient un ensemble de fréquentes k-point ensembles, Lc.

Étape-4: Pour chaque article fréquente ensemble l, générer tous sous-ensembles non vides de l.

Étape-5: Pour toute partie non vides de l, la sortie prononcer "s =>(l-s)" si la confiance C de la règle, "S =>(l-l)".

Algorithme 2.1 : Algorithme U-Apriori

Entrée: contextβ ; seuil minimal de support minsupport;

Sortie: ensembles F_k des k-itemsets frequents;

1) $F_1 \leftarrow$ **{1-itemsets frequents};**
2) **Pour (k** $\leftarrow$**2;** $F_{k-1} \neq \emptyset$ **; k++) faire**
3) $C_k \leftarrow$**Apriori-Gen(**F_{k-1}**);**
4) **Pour chaque objet o faire;**
5) $C_0 \leftarrow$**Subset(**C_k**,o);**
6) **Pour chaque candidat** c_k , $c_{k+1} \in C_0$ **et** c_k **x** $c_{k+1} \geq$ **SM**
7) **fairec.support++;**
8) **Fin pour**
9) $F_k \leftarrow$**{c** $\in C_k$ **| c.support** $\geq$ **minisupport};**
10) **Fin pour**
11) **Retourner**C_kF_k**;**

C_k Ensemble de *k*-itemsets candidats (itemsets potentiellement fréquents). Chaque élément de cet ensemble possède deux champs : *itemset* et *support*.

F_k Ensemble de *k*-itemsets fréquents. Chaque élément de cet ensemble possède deux champs : *itemset* et *support*.

3.2. Les algorithmes de type « diviser-et-régner»

3.2.1. Algorithme UH-mine

L'algorithme UH-Mine fonctionne comme suit:

Étape -1: élaguer la base de données initiale pour supprimer les itemsets non fréquents

Etape -2: diviser la base de données élaguée en égale Morceaux.

Étape -3: utiliser l'Algorithme UH-Mine pour extraire les itemsets fréquents, UH-Mine maintient une H-Struct, qui contient des pointeurs à les éléments de transaction.

Étape -4: jointer les résultats;

Étape -5: scanner la base de données élaguée une autre fois pour éliminer les faux positifs et obtenir les chiffres réels.

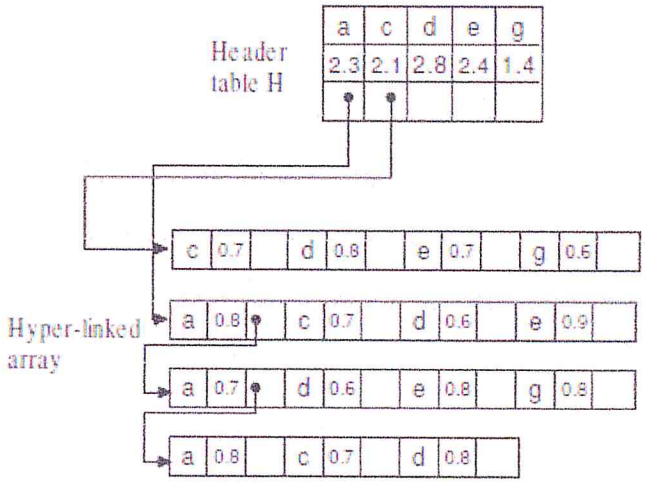


Figure 2.2: La structure L'algorithme UH-Mine **[13]**

3.2.2. L'Algorithme UF-growth

Pour représenter les données incertaines efficacement, on a proposé l'UF-arbre qu'il est une variante de FP-arbre. Chaque nœud dans l'UF-arbre stocker un item, son support attendu et le nombre d'occurrence de tel support pour tel item.

L'algorithme proposé construit UF-arbre comme suit :

Entré : une base de données, un minimal de support minsup

Sortie : UF-arbre, arbre des itemsets fréquents

Etape 1 : scanner la base de données et accumuler les supports attendus de chaque item pour trouver les items fréquents où le support attendu ≥ minsup (minimal support).

Etape 2 : trier ces items fréquents en ordre décroissant des supports attendus accumulés.

Etape 3 : scanner la base de donnés une autre fois et insérer chaque transaction dans l'UF-arbre, de la même façon que dans la construction de FP-arbre, sauf pour le suivant :

Fusionner une nouvelle transaction avec le nœud fils de la racine d'UF-arbre si le même item et le même support attendu existe dans la transaction et le nœud fils.

Avec un tel processus de construction de l'arbre, UF-arbre possède une bonne propriété que le nombre d'occurrences d'un nœud est au moins la somme des chiffres d'occurrence de tous ses nœuds fils **[12]**.

3.2.3. Algorithme UFP-growth

UFP- growth étend l'algorithme initial FP- growth. La construction de FP-arbre a besoin deux analyses de l'ensemble de données. La première analyse collecte les items fréquents et ses supports. Et dans la deuxième analyse, chaque transaction est logée dans la structure de FP-arbre. Les motifs fréquents sont générés de manière récursive de FP-arbre. Afin d'adapter cet algorithme :

Première analyse :

- Calculer le support attendu exacte de chaque ensemble des items par le calcul de leur support à la somme des probabilités existantes dans chaque transaction

Deuxième analyse :

- Chaque transaction est instanciée n fois, selon la probabilité existante des items dans la transaction.
- Insérer les itemsets dans la structure de FP-arbre.
- L'algorithme extrait les itemsets fréquents de la même manière que l'algorithme FP-growth.

3.2.4. L'Algorithme PUF-growth

Pour réduire la taille de l'UF-arbre et UFP-arbre, Leung a proposé la structure d'arbre des itemsets fréquents incertains préfixe capé (PUF-arbre), dans lequel les informations importantes sur les données incertaines est capturé afin qu'on extrait les itemsets incertains à partir de l'arbre. Le PUF-arbre est construit en tenant compte de la limite supérieure de la valeur de probabilité existante pour chaque item quand générer un k-itemset(k>1).

La limite supérieure d'un item x_r dans une transaction t_j est l'item plafond (préfixe) de x_r dans t_j, tel que défini ci-dessous,

Définition. L'item plafond (préfixe) $l^{cap}(x_r, t_j)$ d'un item x_r dans une transaction

$t_j = x_1, \ldots .., x_r, ..., X_h\}$, où $1 < r < h$, est défini comme un produit de $P(x_r, t_j)$ et la plus haute valeur de probabilité existante M des items de x_1 à x_{r-1} dans tj (ie, dans le préfixe propre de x_r dans t_j):

$$I^{cap}(x_h,t_j) = \begin{cases} P(x_r,t_j) \times M \text{ if } h > 1 \\ P(x_1,t_j) \quad \text{if } h = 1 \end{cases}, \text{ where } M = max_{1 \le q \le r-1} P(x_q,t_j) \tag{2.9}$$

Le plafond de support attendu $\text{expSup}^{cap}(X)$ d'un motif $X = \{x_1, \ldots, x_k\}$ (où $k > 1$) est définie comme la somme de tous les items plafonds de x_k dans toutes les transactions qui contiennent X:

$$\text{expSup}^{cap}(X) = \sum_{j=1}^{n} \{(I^{cap}(x_k,t_j) | X \subset t_j)\} \tag{2.10}$$

Dans des autres termes, le plafond de support attendu d'un itemset satisfait à la propriété de fermeture vers le bas.

Exemple :

TID	Transactions	Transactions tries
T1	{*a*: 0.2, *b*: 0.2, *c*: 0.7, *f* :0.8}	{*a*:0. 2, *c*: 0.7, *f*: 0.8}
T2	{*a*: 0.5, *c*: 0.9, *e*: 0.5}	{*a*: 0.5, *c*: 0.9, *e*: 0.5}
T3	{*a*: 0.3, *d*: 0.5, *e*: 0.4,}	{*a* :0.3, *e*: 0.4, *f*: 0.5,}
T4	{*a*: 0.9, *b*: 0.2, *d*: 0.1}	{*a* :0.9, *e*: 0.5, *d*: 0.1}

Tableau 2.2 : Une base de données de transactions avec minsup = 0,5

Considérons une base de données incertaine avec quatre transactions présentées dans la deuxième colonne du tableau 2.2 **[1]**. L'item plafond de c dans t1 peut calculer comme suit :

$I^{cap}(c,t_1) = 0.7 \times \max\{P(a, t_1), P(b, t_1)\} = 0.7 \times \max\{0.2,0.2\} = 0.7 \times 0.2 = 0.14$

$I^{cap}(f,t1) = 0.8 \times \max\{P(a, t_1), P(b, t_1), P(c, t_1)\} = 0.8 \times \max\{0.2,0.2,0.7\} = 0.8 \times 0.7 = 0.56$

4. Etapes de construction de PUF-arbre

- Scanner la base de données et trouver les itemsets fréquents distincts dans la base de données
- Construire un principal tableau I-liste pour stocker les itemsets fréquents dans un ordre cohérant (par exemple l'ordre canonique) pour faciliter la construction de l'arbre.
- Construire PUF-arbre avec la deuxième analyse de la base de données de la même façon que FP-arbre **[12]**

- Lors de l'insertion d'un item de transaction, d'abord calculer son item plafond puis l'insérer dans le PUF arbre selon l'ordre d'I-liste.
- Si ce nœud existe déjà dans le chemin, mettre à jour son item plafond en ajoutant l'item plafond calculé à l'item plafond existant

5. Nombre de faux positifs:

Les deux UFP-arbre et PUF arbres sont compacts, leurs algorithmes correspondants génèrent des faux positifs. Par conséquent, leurs performances globales dépendent du nombre de faux positifs générés. PUF-croissance réduit le nombre de faux positifs lorsqu'on le comparé avec UFP-croissance. La principale raison de cette amélioration est que les limites supérieures de support attendu des itemsets dans clusters ne sont pas serré que les limites supérieures fournies par PUF-croissance. Dans un UFP-arbre, si un père a plusieurs fils, chaque fils utilisera des valeurs plus élevées de cluster dans le père pour générer le support attendu total. Si le nombre total de valeurs de probabilité existantes de ce fils est encore inférieur à celui de la plus haute valeur du cluster du père, le support attendu du chemin avec ce père et le fils sera élevé. Cela conclus plus de faux positifs dans long terme.

Dataset	minsup	UFP-growth[8]	PUF-growth[1]
u10k5L_80_90	0.1	61.20%	22.55%
u100k10L_50_60	0.07	89.32%	25.99%

Tableau 2.3 : Comparaison des faux positifs (en termes de modèles n totaux) [13]

6. Comparaison

N°	Algorithmes	Avantage	Inconvénient
1	U Apriori	Cet algorithme permet de réduire la taille de l'ensemble des candidats.	Il scanne la base de données plusieurs fois et cette performance est affectée.
2	UF-growth	Cet algorithme utilise UF-arbres pour extraire les itesmsets fréquents de bases de données incertaines dans deux scans de base de données.	Il contient un chemin d'arbre distinct pour chaque élément distinct, probabilité existantes paire.
3	UFP-growth	Cet algorithme scanne la base de données deux fois, Où les nœuds d'items ont les mêmes valeurs de probabilité existantes sont fusionner dans un méga-	Il contient un chemin d'arbre distinct pour chaque élément distinct, probabilité existantes paire.

		nœud, le résultat est méga-nœud dans l'UFP-arbre	
4	UH-mine	Cet algorithme stocke tous les items fréquents dans chaque transaction de base de données dans un hyper-structure appelée UH-struct.	Problème de cout élevé de calcul lorsqu'on calcule les supports attendus
5	PUF-growth	Cet algorithme permet d'extraire les motifs fréquents avec la construction d'une base de données prévue pour chaque motif fréquent potentiel et de manière récursive extraient ses extensions fréquentes potentielles.	Il crée des faux positifs.

Tableau 2.4 : Comparaison entre les algorithmes [14]

7. Conclusion

Dans ce chapitre, l'étude approfondie des algorithmes d'extraction d'itemsets fréquents de données incertaines est faite et a identifié des nombreuses forces et des faiblesses de chacun. Des nouvelles variantes d'algorithmes existants sont comparées avec des algorithmes et des résultats d'extractions classiques des prestations et des limites importantes. Cette comparaison peut aussi tomber dans diverses questions d'optimisation qui conduiront à une meilleure performance. L'efficacité des algorithmes d'extraction est plus obstacle, mais encore il est nécessaire de développer des méthodes pour obtenir d'excellents résultats .pour cela dans le prochain chapitre nous allons présenter les techniques d'optimisations pour l'extraction des règles d'association.

Chapitre3 :

Techniques d'optimisation pour l'extraction des règles d'association

Chapitre 3 : Techniques d'optimisation pour l'extraction des règles d'association

1. Introduction

La technologie d'extraction de données a émergé comme un moyen d'identifier les modèles et les tendances à partir de grandes quantités de données. Plusieurs techniques sont développées pour optimiser l'extraction des motifs fréquents à partir des données. Dans ce chapitre on va étudier les techniques d'optimisations pour choisir la meilleure technique qui nous aidera à extraire les motifs fréquents.

2. Les techniques d'optimisation

Dans ce section on va étudier les catégories des techniques d'optimisation.

2.1. Meta-heuristique

Une méta-heuristique est une méthode générique pour la résolution de problèmes combinatoires difficiles, apparue à partir des années 1980, permettent de trouver une solution de bonne qualité en un temps de calcul en général raisonnable.

Il existe essentiellement deux grandes familles d'approches heuristiques :
Les approches perturbatives, construisent des combinaisons en modifiant des combinaisons existantes ; les approches constructives, génèrent des combinaisons de façon incrémentale en utilisant pour cela un modèle stochastique. [15]

2.1.1. Meta-heuristique perturbative

Les approches perturbatives explorent l'espace des combinaisons E en perturbant itérativement des combinaisons déjà construites : partant d'une ou plusieurs combinaisons initiales (généralement prises aléatoirement dans E), l'idée est de générer à chaque étape une ou plusieurs nouvelles combinaisons en modifiant une ou plusieurs combinaisons générées précédemment. Ces approches sont dites "basées sur les instances" .Les approches perturbatives les plus connues sont les algorithmes génétiques, et la recherche locale.

2.1.1.1.Algorithmes génétiques

Les algorithmes génétiques s'inspirent de la théorie de l'évolution et des règles de la génétique qui expliquent la capacité des espèces vivantes à s'adapter à leur environnement par la combinaison des mécanismes suivants :

Chapitre 3 : Techniques d'optimisation pour l'extraction des règles d'association

La sélection naturelle fait que les individus les mieux adaptés à l'environnement tendent à survivre plus longtemps et ont donc une plus grande probabilité de se reproduire ;

La reproduction par croisement fait qu'un individu hérite ses caractéristiques de ses parents, de sorte que le croisement de deux individus bien adaptés à leur environnement aura tendance à créer un nouvel individu bien adapté à l'environnement. [15]

La mutation fait que certaines caractéristiques peuvent apparaître ou disparaître de façon aléatoire, permettant ainsi d'introduire de nouvelles capacités d'adaptation à l'environnement, capacités qui pourront se propager grâce aux mécanismes de sélection et de croisement.

Les algorithmes génétiques reprennent ces mécanismes pour définir une méta-heuristique. L'idée est de faire évoluer une population de combinaisons, par sélection, croisement et mutation, la capacité d'adaptation d'une combinaison étant ici évaluée par la fonction objective à optimiser.

A-Initialisation de la population : en général, la population initiale est générée de façon aléatoire, selon une distribution uniforme assurant une bonne diversité des combinaisons.

Algorithme 1: Algorithme génétique

Initialiser la population avec un ensemble de combinaisons de E
tant que *critères d'arrêt non atteints* **faire**
    Sélectionner des combinaisons de la population
    Créer de nouvelles combinaisons par recombinaison et mutation
    Mettre à jour la population
retourner *la meilleure combinaison ayant appartenu à la population*

Sélection : cette étape consiste à choisir les combinaisons de la population qui seront ensuite candidates pour la recombinaison et la mutation. Il s'agit là de favoriser la sélection des meilleures combinaisons, tout en laissant une petite chance aux moins bonnes combinaisons.

Recombinaison (croisement) : cette étape vise à créer de nouveaux individus par un mélange de combinaisons sélectionnées. L'objectif est de conduire la recherche dans une nouvelle zone de l'espace où de meilleures combinaisons peuvent être trouvées.

Mutation : cette opération consiste à modifier de façon aléatoire certains composants des combinaisons obtenues par croisement.

B-Mise à jour de la population : cette étape détermine quelles nouvelles combinaisons doivent devenir membre de la population et quelles anciennes combinaisons de la population

doivent être remplacées. La politique de mise-à-jour est essentielle pour maintenir une diversité appropriée de la population, pour prévenir une convergence prématurée du processus de recherche, et pour permettre à l'algorithme de toujours découvrir de nouvelles zones prometteuses dans l'espace de recherche. Ainsi, les décisions sont souvent prises selon des critères liés à la fois à la qualité et à la diversité. Par exemple, une règle bien connue de mise à jour basée uniquement sur la qualité consiste à remplacer la pire des combinaisons de la population, tandis qu'une règle fondée sur la diversité consiste à substituer les anciennes combinaisons de la population par de nouvelles combinaisons similaires, conformément à une mesure de similarité donnée. D'autres critères comme l'age peuvent aussi être considérés.

Critères d'arrêt : le processus d'évolution est itéré, de génération en génération, jusqu'à ce qu'un critère d'arrêt soit atteint. Il peut s'agir d'un nombre maximum de générations, un nombre maximum d'évaluations, un nombre maximum de générations sans améliorer la meilleure solution, une qualité de solution atteinte ou encore une diversité de population inférieure à un seuil donné.

2.1.1.2. Recherche locale

Une recherche locale explore l'espace des combinaisons de proche en proche, en partant d'une combinaison initiale et en sélectionnant à chaque itération une combinaison voisine de la combinaison courante, obtenue en lui appliquant une transformation élémentaire.

Algorithme 2: Recherche locale

Générer une combinaison initiale e ϵ E
tant que *critères d'arrêt non atteints* **faire**
    Choisir e' ϵ v(e)
    e ←e'
retourner *la meilleure combinaison construite*

Fonction de voisinage : l'algorithme de recherche locale est paramétré par une fonction de voisinage v : E ! P(E) définissant l'ensemble des combinaisons que l'on peut explorer à partir d'une combinaison donnée. Étant donnée une combinaison courante eϵE, v(e) est un ensemble de combinaisons que l'on peut obtenir en appliquant une modification "élémentaire" à e. On peut généralement considérer différents opérateurs de modification comme, par exemple, changer la valeur d'une variable ou échanger les valeurs de deux variables. Chaque opérateur

de modification différent induit un voisinage différent, qui peut contenir un nombre plus ou moins grand de combinaisons plus ou moins similaires à la combinaison courante. Ainsi, le choix de l'opérateur de voisinage a une influence forte sur les performances de l'algorithme. Il est généralement souhaitable que l'opérateur de voisinage permette d'atteindre la combinaison optimale à partir de n'importe quelle combinaison initiale de E, ce qui revient à imposer que le graphe orienté associant un sommet à chaque combinaison de E et un arc (ei; ej) à chaque couple de combinaisons telles que ej ev(ei), admette un chemin depuis n'importe lequel de ses sommets jusque la combinaison optimale.

Génération de la combinaison initiale : la combinaison à partir de laquelle le processus d'exploration commence est souvent générée de façon aléatoire. Elle est parfois générée en suivant une heuristique constructive gloutonne .Lorsque la recherche locale est hybridée avec une autre méta-heuristique, comme par exemple les algorithmes génétiques ou les algorithmes par colonies de fourmis, la combinaison initiale peut être le résultat d'un autre processus de recherche.

Choix du voisin : à chaque itération de la recherche locale, il s'agit de choisir une combinaison dans le voisinage de la combinaison courante. Ce choix d'une combinaison voisine est appelé "mouvement". Il existe un très grand nombre de stratégies de choix. On peut par exemple sélectionner à chaque itération le meilleur voisin, c'est-à-dire, celui qui améliore le plus la fonction objectif ou bien le premier voisin trouvé qui améliore la fonction objectif. De telles stratégies "gloutonnes" (aussi appelées "montées de gradients") risquent fort d'être rapidement piégées dans des optima locaux, c'est-à-dire, sur des combinaisons dont toutes les voisines sont moins bonnes. Pour s'échapper de ces optima locaux, on peut considérer différentes méta-heuristiques, par exemple, pour n'en citer que quelques-unes :

— la marche aléatoire (random walk) ,qui autorise avec une très petite probabilité de sélectionner un voisin de façon complètement aléatoire ;

— le recuit simulé (simulated annealing, qui autorise de sélectionner des voisins de moins bonne qualité selon une probabilité qui décroit avec le temps ;

— la recherche taboue (tabu search) ,qui empêche de boucler sur un petit nombre de combinaisons autour des optima locaux en mémorisant les derniers mouvements effectués dans une liste taboue et en interdisant les mouvements inverses à ces derniers mouvements ;

— la recherche à voisinage variable, qui change d'opérateur de voisinage lorsque la combinaison courante est un optimum local par rapport au voisinage courant.

Répétition du processus de recherche locale : une recherche locale peut être répétée plusieurs fois à partir de combinaisons initiales différentes. Il peut s'agir de combinaisons initiales indépendantes, générées aléatoirement. On parle alors de recherche locale à plusieurs points de départ (multi-start local search). Il peut également s'agir de combinaisons initiales obtenues en perturbant une combinaison issue d'un processus de recherche locale précédent. On parle alors de recherche locale itérée (iterated local search). On peut par ailleurs faire plusieurs recherches locales en parallèle, en partant de différentes combinaisons initiales, et redistribuer régulièrement les combinaisons courantes en supprimant les moins bonnes et dupliquant les meilleures selon le principe "aller avec les meilleurs" (go with the winner)

2.1.2. Méta-heuristiques constructives

Les approches constructives construisent une ou plusieurs combinaisons de façon incrémentale, c'est-à dire, en partant d'une combinaison vide, et en ajoutant des composants de combinaison jusqu'à obtenir une combinaison complète. Ces approches sont dites "basées sur les modèles", dans le sens où elles utilisent un modèle, généralement stochastique, pour choisir à chaque itération le prochain composant de combinaison à ajouter à la combinaison en cours de construction. Il existe différentes stratégies pour choisir les composants à ajouter à chaque itération, les plus connues étant les stratégies gloutonnes aléatoires, les algorithmes par estimation de distribution, et la méta-heuristique d'optimisation par colonies de fourmis, et la méta-heuristique d'optimisation par colonies d'abeilles.**[15]**

2.1.2.1.Optimisation par colonies de fourmis

Il existe un parallèle assez fort entre l'optimisation par colonies de fourmis (Ant Colony Optimization ;ACO) et les algorithmes par estimation de distribution . Ces deux approches utilisent un modèle probabiliste glouton pour générer des combinaisons, ce modèle évoluant en fonction des combinaisons précédemment construites dans un processus itératif d'apprentissage. L'originalité et la contribution essentielle d'ACO est de s'inspirer du comportement collectif des fourmis pour faire évoluer le modèle probabiliste. Ainsi, la probabilité de choisir un composant est définie proportionnellement à une quantité de phéromone représentant l'expérience passée de la colonie concernant le choix de ce composant. Cette quantité de phéromone évolue par la conjugaison de deux mécanismes : un

mécanisme de renforcement des traces de phéromone associées aux composants des meilleures combinaisons, visant à augmenter la probabilité de sélection de ces composants ; et un mécanisme d'évaporation, visant à privilégier les expériences récentes par rapport aux expériences plus anciennes. **[15]**

Algorithme 3: Optimisation par colonies de fourmis

Initialiser *les traces de phéromone à* T_0
tant que les conditions d'arrêt ne sont pas atteintes **faire**
    Construction de combinaisons par les fourmis
    Mise à jour de la phéromone
retourner *la meilleure combinaison construite*

Structure phéromonale : La phéromone est utilisée pour biaiser les probabilités de choix des composants de combinaisons lors de la construction gloutonne aléatoire d'une combinaison. Un point crucial pour la performance de l'algorithme réside donc dans le choix de la structure phéromonale, c'est-à-dire, dans le choix des données sur lesquelles des traces de phéromone seront déposées. Au début de l'exécution d'un algorithme ACO, toutes les traces de phéromone sont initialisées à une valeur T_0 donnée.

Construction de combinaisons par les fourmis : A chaque cycle d'un algorithme ACO, chaque fourmi construit une combinaison selon un principe glouton aléatoire similaire.

Partant d'une combinaison vide, ou d'une combinaison contenant un premier composant de combinaison choisi aléatoirement ou selon une heuristique donnée, la fourmi ajoute un nouveau composant de combinaison à chaque itération, jusqu'à ce que la combinaison soit complète. A chaque itération, le prochain composant de combinaison est choisi selon une règle de transition probabiliste : étant donnés un début de combinaison S, et un ensemble C de composants de combinaison pouvant être ajoutés à S, la fourmi choisit le composant $i \in C$ selon la probabilité :

$$Ps(i)=\frac{[T_s(i)]^{\alpha}.[\mu_s(i)]^{\beta}}{\sum_{i\in C}[T_s(i)]^{\alpha}.[\mu_s(i)]^{\beta}} \qquad (3.1)$$

Où $T_s(i)$ est le facteur phéromone associé au composant de combinaison i par rapport au début de combinaison S (la définition de ce facteur dépend de la structure phéromone choisie), $\mu_s(i)$ est le facteur heuristique associé à i par rapport au début de combinaison S (la définition de ce facteur dépend du problème), et α et β sont deux paramètres permettant de

moduler l'influence relative des deux facteurs dans la probabilité de transition. En particulier, si $\alpha = 0$ alors le facteur phéromone n'intervient pas dans le choix des composants de combinaison et l'algorithme se comporte comme un algorithme glouton aléatoire pur. A l'inverse, si $\beta = 0$ alors seules les traces de phéromone sont prises en compte pour définir les probabilités de choix.

Mise à jour de la phéromone : Une fois que chaque fourmi a construit une combinaison, les traces de phéromone sont mises à jour. Elles sont tout d'abord diminuées en multipliant chaque trace par un facteur $(1-- \rho)$, où $\rho \in [0; 1]$ est le taux d'évaporation. Ensuite, certaines combinaisons sont "récompensées" par un dépôt de phéromone. Il existe différentes stratégies concernant le choix des combinaisons à récompenser. On peut récompenser toutes les combinaisons construites lors du dernier cycle, ou bien seulement les meilleures combinaisons du cycle, ou encore la meilleure combinaison trouvée depuis le début de l'exécution. Ces différentes stratégies influent sur l'intensification et la diversification de la recherche. En général, la phéromone est déposée en quantité proportionnelle à la qualité de la combinaison récompensée. Elle est déposée sur les traces de phéromone associées à la combinaison à récompenser ; ces traces dépendent de l'application et de la structure phéromone le choisie.

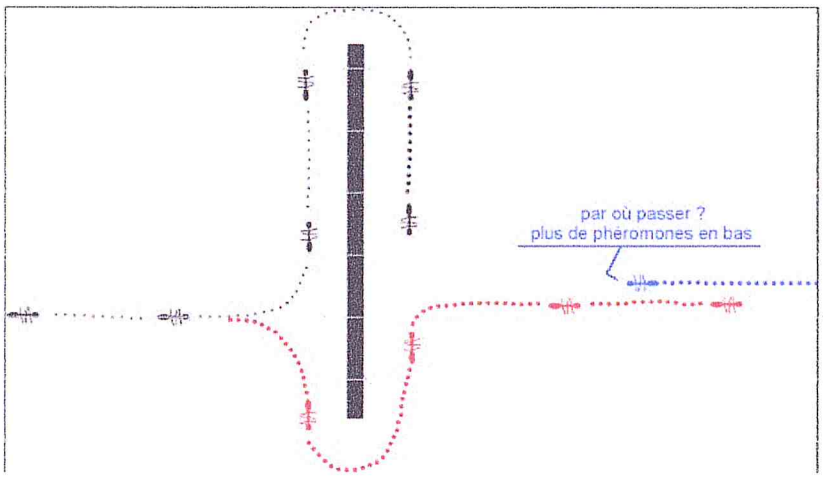


Figure 3.1: la structure phéromone le choisie.

Les fourmis rouge ont pris un plus court chemin, plus ont franchi l'obstacles, donc plus de phéromones ont été déposées du côté rouge que du côté noir. La fourmi bleue venant de l'autre sens choisira donc le côté rouge.

2.1.2.2. Optimisation par colonies d'abeilles

Définition

L'algorithme de colonie d'abeille artificielle (Artificial Bee Colony ABC en Anglais) est l'un des plus récemment introduit dans la base des Algorithmes basés essaim. ABC simule le comportement intelligent de recherche de nourriture d'un essaim d'abeilles. L'algorithme de colonie d'abeille artificielle a été introduit par Karaboga **[16].**

Dans l'algorithme ABC, l'emplacement de la source de nourriture représente la solution possible au problème, et la quantité du nectar de cette source correspond à une valeur objective dite fitness. Le nombre des abeilles employeuses ou des abeilles spectateurs représente le nombre de solution dans la population.

Comportement des abeilles :

Les abeilles sont des insectes sociaux. Elles sont obligées de vivre en colonie très organisée, formée d'ouvrières, de faux-bourdon et d'une seule reine, et où chacune a un travail bien précis à faire.

L'abeille est capable, par la danse ou par la production de substances chimiques appelées « phéromone », de communiquer aux autre abeilles l'endroit où elle a découvert de la nourriture. Elle **danse en rond** (Figure 3.3) quand elle a trouvé du pollen à faible distance (moins de 25 mètres). Elle utilise une danse très compliquée dite la **danse frétillante** (Figure 3.4), ou danse en huit, si la nourriture se trouve à moins de 10 kilomètres. La direction de la nourriture est exprimée par rapport à la position du soleil. La distance est exprimée par le nombre et la vitesse des tours effectués par l'abeille sur elle-même. Afin de survivre à l'hiver, les abeilles doivent recueillir et stocker environ 15 à 50 Kg de nectar.

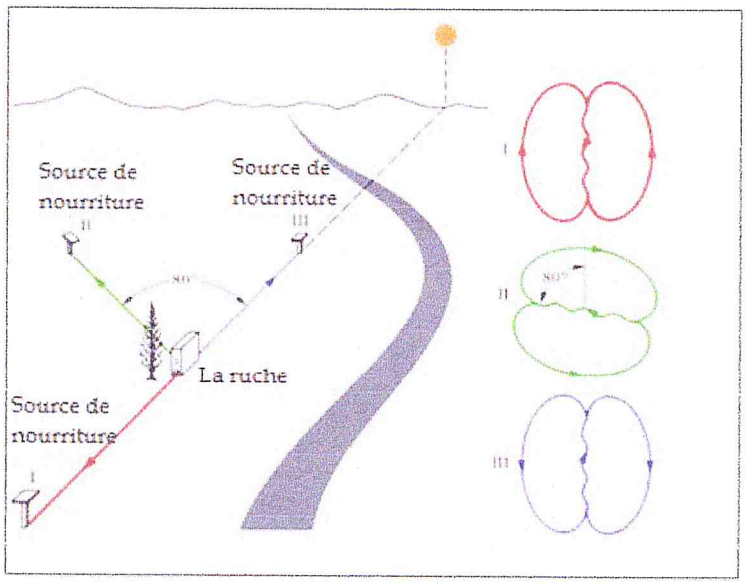


Figure 3.2 : La danse en rond qu'effectue l'abeille en fonction de la direction de la source de nourriture.

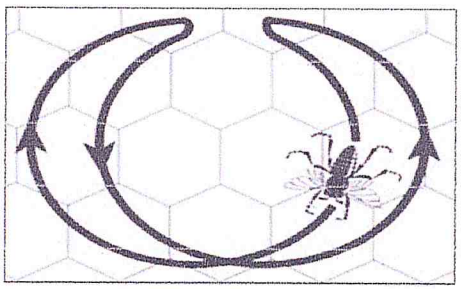


Figure 3.3 : la danse frétillante, appelée aussi en huit

Dans cet algorithme, l'emplacement de la source de nourriture représente la solution possible au problème, et la quantité du nectar de cette source correspond à une valeur objective dite **fitness**.

Les butineuses sont attribués aux différentes sources de nourriture de façon à maximiser l'apport total de nectar. La colonie doit **optimiser** l'efficacité globale de la collecte. La répartition des abeilles est donc en fonction de nombreux facteurs tels que la quantité du nectar et la distance entre la source de nourriture et la ruche. Ce problème et similaire à la répartition des serveurs d'hébergement web, qui était en fait un des premiers problèmes résolus en utilisant les algorithmes d'abeilles par NAKRANI et TOVEY en 2004.

Le nombre des butineuses actives ou inactives représente le nombre de solution dans cette populations.

Dans la première étape, l'algorithme génère une population initiale de SN solutions distribuées de façon aléatoire. Chaque solution x_i $(i = 1, 2, \dots, SN)$ qui est initialisée par les éclaireuses, et représente un vecteur de solution au problème d'optimisation. Les variables que contient chaque vecteur doivent être optimisées.

Après l'initialisation, la population des solutions est soumise à des cycles répétés $C = 1, 2, \dots, C_{max}$, ces cycles représentent des processus de recherches faits par les butineuses actives, inactives et les éclaireuses.

Les butineuses actives recherchent dans le voisinage de la source précédente x_i de nouvelles sources v_i ayant plus de nectar, Elles calculent ensuite leur fitness. Afin de produire une nouvelle source de nourriture à partir de l'ancienne, on utilise l'expression ci contre :

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \qquad (3.2)$$

Où $k \in \{1, 2, \dots, BN\}(BN \text{ est le nombes des butineuses actives})et\ j \in \{1, 2, \dots, SN\}$ sont des indices choisis au hasard. Bien que k est déterminé aléatoirement, il doit être différent de i. ϕ_{ij} est un nombre aléatoire appartenant à l'intervalle [-1, 1], il contrôle la production d'une source de nourriture dans le voisinage de x_{ij}.

Après la découverte de chaque nouvelle source de nourriturev_{ij}, un mécanisme de sélection gourmande est adopté, c'est-à-dire que cette source est évaluée par les abeilles artificielles, sa performance est comparée à celle de x_{ij}. si le nectar de cette source est égale ou meilleur que celui de la source précédente, celle-ci est remplacée par la nouvelle. Dans le cas contraire l'ancienne est conservée.

Pour un problème de minimisation, La fitness est calculée suivant cette formule :

$$fit_i(\overrightarrow{x_i}) = \begin{cases} \frac{1}{1+f_i(\overrightarrow{x_i})} & si\ f_i(\overrightarrow{x_i}) \geq 0 \\ 1 + abs\left(f_i(\overrightarrow{x_i})\right) & si\ f_i(\overrightarrow{x_i}) > 0 \end{cases} \qquad (3.3)$$

Telle que $f_i(\overrightarrow{x_i})$est la valeur de la fonction objectif de la solution $\overrightarrow{x_i}$.

A ce stade, les butineuses inactives et les éclaireuses qui sont entrain d'attendre au sein de la ruche. A la fin du processus de recherche, les butineuses actives partagent les informations sur le nectar des sources de nourriture ainsi que leurs localisations avec les autre abeilles via la danse frétillante. Ces dernières évaluent ces informations tirées de toutes les butineuses actives, et choisissent les sources de nourriture en fonction de la valeur de probabilitéP_i associée a cette source, et calculée par la formule suivante :

$$P_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \qquad (3.4)$$

Où fit_i est la fitness de la solution i, qui est proportionnelle à la quantité du nectar de la source de nourriture de la position i).

La source de nourriture dont le nectar est abandonné par les abeilles, les éclaireuses la remplacent par une nouvelle source. Si durant un nombre de cycle prédéterminé appelé « limite » une position ne peut être améliorée, alors cette source de nourriture est supposée être abandonnée.

Toutes ces étapes sont résumées dans l'algorithme de la Figure 3.4.

```
Entrée : S, W, O

Sortie : la meilleure solution

  1- Initialiser la population avec S+W solutions aléatoires
  2- Evaluer la fitness de la population
  3- Tant que le critère d'arrêt n'est pas satisfait faire
  4-      Recruter O butineuses inactives et attribuer
  5-      chacune à un membre de la population
  6-      Pour chaque butineuse inactive affectée à un membre
  7            n de la population faire
  8-         Effectuer une itération de l'algorithme de
  9-         recherche de nouvelle source
  10-     Fin pour
  11-     Evaluer la fitness de la population
  12-     Si un membre de la population ne s'est pas amélioré
  13-        au cours des itérations faire
  14-        Sauver la solution et remplacer la par une
  15-        solution aléatoire
  16-     Trouver S solutions aléatoires et remplacer les S
  17-     membres de la population qui ont la mauvaise fitness
  18- Fin Tant que
  19- Retourner la meilleure solution
```

Figure 3.4 : algorithme de l'optimisation par colonie d'abeille :
S : nombre de butineuse éclaireuse.
W : nombre de butineuse active.
O : nombre de butineuse inactive

2.2.l'extraction des règles d'association

L'extraction des règles d'association (Association Rules mining ARM) est l'un des plus importants et bien étudié techniques de tâches d'exploration de données. L'extraction des règles d'association à partir d'une grande base de données, a été une tâche importante dans le domaine de l'exploration de données pour découvrir, des associations intéressantes cachés qui se produisent entre des différents éléments de données. De nos jours, ARM est largement utilisé dans de nombreux différents domaines tels que les réseaux de télécommunication, marché et gestion des risques, le contrôle des stocks mobiles l'exploitation minière, l'exploitation minière de graphique, l'exploitation minière de l'éducation, etc

Plusieurs algorithmes d'optimisations a été développés dans l'extraction des règles d'association. Dans ce chapitre on va citer deux algorithmes, BSO ARM basé sur le méta-heuristique algorithme « colonie d'abeille », TS ARM basé sur le méta-heuristique algorithme « recherche local, tabu search ».

2.2.1. Algorithme BSO-ARM

Cet algorithme, basé sur la méta-heuristique optimisation par colonie d'abeille (bees swarm optimisation) : [17]

Algorithm 1: BSO Algorithm

```
Sref← The solution found by InitBee.
while i < Max-Iter and not stop do
    Insert Sref in taboo list.
    (SearchArea(Sref).
    Assign a solution from SearchArea to each bee.
    for each bee k do
        Built-Search-Area(bee k ).
        Store the result in the table Dance.
    end for
(Choose the new reference solution Sref.
end while
```

2.2.2. Algorithme recherche local (TS- ARM)

Cet algorithme basé sur le méta-heuristique algorithme ,recherche local (tabu search TS) :[18]

Algorithm 2: TS Algorithm

```
l ← Desired maximum tabu list length
n ← number of tweaks desired to sample the gradient
S ← some initial candidate solution
Best ← S
L ← {} a tabu list of maximum length l " Implemented as first in, first-out queue
Enqueue S into L
repeat
  if Length(L) > l then
    Remove oldest element from L
    R ← Tweak(Copy(S))
    for n−1 times do
        W ← Tweak(Copy(S))
        if W ∈/ L and (Quality(W) > Quality(R) or R ∈ L) then
            R ← W
            if R ∈/ L and Quality(R) > Quality(S) then
                S ← R
                Enqueue R into L
  if Quality(S) > Quality(Best) then
        Best ← S
until Best is the ideal solution or we have run out of time
return Best.
```

3. Discussion et comparaison

La variété des méta-heuristiques pose naturellement le problème du choix de celle qui sera la plus efficace pour résoudre un nouveau problème. Evidemment, cette question est complexe et se rapproche d'une quête fondamentale en intelligence artificielle, à savoir la résolution automatique de problèmes. Nous n'abordons dans cette discussion que quelques éléments de réponse. En particulier, le choix d'une méta-heuristique dépend de la pertinence de ses mécanismes de base pour le problème considéré, c'est-à-dire, de leur capacité à générer de bonnes combinaisons :

- Pour les algorithmes génétiques AG, l'opérateur de recombinaison doit être capable d'identifier les bons motifs qui doivent être assemblés et hérités par recombinaison.

- Dans le cas de la recherche locale, le voisinage doit favoriser la construction de meilleures combinaisons.
- Pour ACO, la structure phéromonale doit être capable de guider la recherche vers de meilleures combinaisons.

Pour BSO, les butineuses (actives, inactives et les éclaireuses) doit rechercher et calculer le fitness afin d'obtenir les meilleurs solutions.

Le choix d'une méta-heuristique dépend la complexité en temps, la performance, l'existence de structures de données qui permettent une évaluation incrémentale de la fonction d'évaluation après chaque application des opérateurs élémentaires de cette méta-heuristique, l'efficacité, les bonnes briques et le bon paramétrage.

4. Conclusion

Apres l'étude des techniques d'optimisations, on peut déduire que l'extraction des règles d'association permet de trouver l'association entre les itemsets fréquents. et on trouve que les algorithmes qui permettre d'extraire les itemsets fréquents a partir des données incertaines dans un temps raisonnable avec des bons résultats sont les algorithmes de base méta-heuristique, c pour cela on a choisi la technique d'optimisation méta-heuristique comme une solution pour extraire les itemsets fréquents et après l'étude d ses algorithme on a trouvé que l'optimisation par colonie d'abeille c'est la bonne solution grâce à ses avantages, alors dans le prochain chapitre on va présenter notre solution pour extraire les itemsets fréquents à partir des données incertaines.

Chapitre 4 :
Solution proposée

1. Introduction

Quand un nouveau problème se pose, on se retrouve devant une tâche difficile qui conduit à définir de nouvelle méthode de résolution car les techniques existantes se sont pas convenables ou dépassé. Et chaque nouvelle méthode s'inspire d'une méthode déjà existante. Dans ce chapitre nous allons décrire l'algorithme propose BSO-UFIM pour l'extraction des itemsets fréquents à partir de données incertaines.

2. Algorithme Colonies d'abeilles pour l'extraction des itemsets fréquents.

Notre algorithme BSO-UFIM a été inspiré de l'algorithme d'optimisation par colonies de d'abeilles proposé par Drias et al. [22]

Pseudo code de l'algorithme général d'optimisation par colonie d'abeilles (BSO-UFIM)

Début
Laisser Sref soit la solution trouvée par Abeille Initiale ;
Tanque la condition d'arrêt n'est pas satisfaite **Faire**
Début
Insérer Sref dans la liste tabou ;
Déterminer l'espace de recherche de Sref ;
Affecter une solution de l'espace de recherche à chaque abeille ;
Pour chaque Abeille K **Faire**
Début
Rechercher en commençant par la solution affectée à elle ;
Stocker le résultat dans le tableau Danse ;
Fin ;
Choisissez la nouvelle solution de référence Sref ;
Fin ;
Fin ;

Notre propre solution consiste à adapter l'algorithme décrit ci-dessous pour extraire des itemsets fréquent à partir de données incertaines.

Notre solution propose 3 étapes nécessaires a passé pour cet extraction, au début on a une base de données incertaines qui contient des itemsets.

- **1 iére étape (choix de solution initiale)**

Le début de notre algorithme proposé contient une solution Sref donnée d'une façon aléatoire et deux valeurs nécessaires pour débuter notre programme, une valeur MinSup et l'autre présente le nombre d'itération MaxIter qui sont donnée par l'utilisateur.

Tout d'abord en calcule la solution Sref qui est on la teste si elle est fréquente ou pas par rapport au minsup donné par l'utilisateur.

Si Sref est fréquente on passe à la deuxième étape sinon on choisit une autre Sref d'une façon aléatoire jusqu'on obtienne une solution fréquente.

Pour calculer la fréquentess de Sref on doit utiliser la formule de l' Expected Support.

a. Expected support

En Utilisant le modèle probabiliste, l' Expected support d'un itemset X dans un ensemble de données T de taille D, noté exp Sup (X, T) est calculé comme suit:

$$expSup(X,T) = \sum_{i=1}^{D}\left(\prod_{x\in X} P(x,T_i)\right) \qquad (4.1)$$

On compare la valeur de fréquentness de Sref avec le MinSup, si ExpSup(Sref) >MinSup alors Sref est fréquente sinon le programme choisi une autre solution Sref aléatoirement.

- **2 iéme étape (Détermination des zones de recherche)**

Après avoir choisit une solution Sref fréquente, dans la deuxième étape on utilise une méthode sur la solution Sref qu'on la appelé « stratégie de la taille des itemsets » pour déterminer les zones de recherche.

A. Détermination de solution de début de chaque zone de recherche

Après avoir une solution Sref fréquente, dans la premier étape on utilise une méthode sur la solution Sref qui s'appelle stratégie de ajoute un bit, elle serre a trouvé un espace ou bien un zone de recherche dans notre base de données.

Au début on présente Sref comme un grand vecteur de taille m qui est composé de sous vecteurs de taille n, ses sous vecteurs sont les items set de Sref, et le grand vecteur contient que des valeurs binaire 0 et 1, si un item existe dans Sref en lui donne la valeur 1 sinon le 0.

La taille du grand vecteur est le nombre de tous les items qui existe dans notre base de données.

Exemple :

1	2	3	4	5	.	.	.	.	m
0	0	0	0	1	.	.	.	.	n

A. **Génération de voisinage de chaque zone de recherche**

Sref S_0 :

0	0	0	0	1	1	1	0	0	0

0	0	0	0	1	1	1	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0

Taille = 1

0	0	0	0	1	1	1	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	1	1	0	0	0

Taille = 2

0	0	0	0	1	1	1	0	0	0
0	0	0	0	1	1	1	0	0	0

Taille =3

2) Ajouter un bit (1) avant le premier bit (1) de Sref et Après cela générer les rester solution.

0	0	0	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	0	0

Taille =4

0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	0

Taille = 5

0	1	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	1	1

Taille = 6

1	1	1	1	1	1	1	0	0	0

Taille = 7

1	1	1	1	1	1	1	1	0	0

Taille = 8

1	1	1	1	1	1	1	1	1	0

Taille = 9

1	1	1	1	1	1	1	1	1	1

Taille = 10

Chaque abeille travaille sur une solution parmi la liste des solutions que générer.

3^{iéme} étape

Elle consiste à affecter à chaque abeille une taille des solutions de la 2^{ième} étape.

Après on va calculée la fréquentness de chaque solution obtenu, et on prenant les solutions fréquentes et on supprime les solutions non fréquentes, la meilleure solution on la sauvegarde avec sa valeur de fréquentness dans tableau de « danse », il contient tous les meilleures solutions obtenu de Sref.

Dans les autres itérations on prend la meilleure solution de tableau Danse de l'itération précédente et on la considère comme une nouvelle solution Sref.

3. L'architecture de l'algorithme colonie d'abeille

L'architecture présenté dans la figure (4.1) représente les différentes étapes que suit l'algorithme de colonie d'abeilles afin d'extraire les itemsets fréquents.

Les différentes étapes sont :

- **Etape1 :** Entre les donnes incertaines et Sref initial, MinSup, Maxitération
- **Etape 2 :** Génère zone de recherche possible.
- **Etape 3 :** Orientation chaque zone avec l'abeille,et Calculer l'expected support de chaque solution.
- **Etape 4 :** Faire une comparaison entre la valeur de min-sup et Expected Support.
- **Etape 5 :** Faire une comparaison entre les solutions fréquentes ensuite prendre la meilleure solution et enregistrer dans la table danse.
- **Etape 6 :** Enregistrée la meilleure solution de chaque itération dans tabou liste.
- **Etape 7 : Test 2 :** si le meilleur solution prendre a prochain itération comme Sref sinon nous faire nouvelle fonction Random .

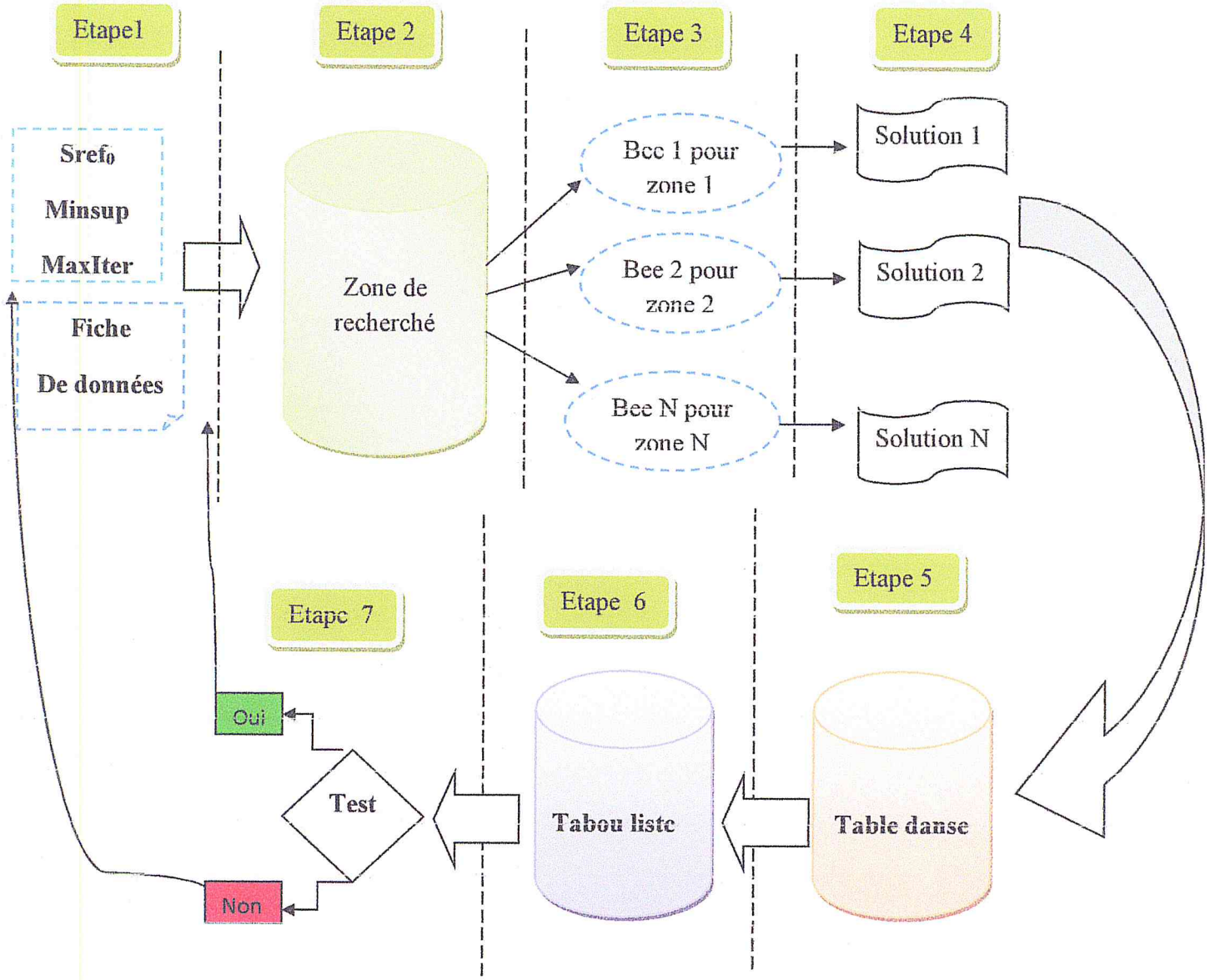


Figure 4.1 : Schéma de l'algorithme **BSO-UFIM**

4. Exemple d'application

Afin de démontrer l'efficacité de l'approche proposée (BSO-UFIM), nous l'examinons sur les données incertaines présentées dans le tableau (4.1).que contient 10 itemsets{1, 2, 3, 4, 5, 6, 7, 8, 9,10} et 7 transaction.

Tid	Les itemsets (transaction)
1	**1**:0.69 **2**:0.72 **4**:0.80 **6**:0.75 **9**:0.85
2	**2**:0.78 **3**:0.864 **4**:0.84 **5**:0.88 **7**:0.81 **9**:0.74
3	**1**:0.84 **4**:0.72 **5**:0.79 **6**:0.83 **8**:0.71 **9**:0.71 **10**:0.88
4	**1**:0.80 **3**:0.78 **10**:0.86
5	**2**:0.84 **4**:0.44 **5**:0.13 **6**:0.02 **7**:0.83 **8**:0.12 **9**:0.79
6	**1**:0.79 **3**:0.47 **5**:0.07 **7**:0.04 **8**:0.75 **10**:0.70
7	**7**:0.70 **8**:0.83 **9**:0.82 **10**:0.82

Tableau 4.1 : les données

L'algorithme BCO-UFIM est composé de trois étapes, une étape que génère la solution $Sref_0$ et étape de parcours de la base de transactions qui calcule le expected support de l'itemset et une étape qui compare ce expected support avec le minsup choisi par l'utilisateur. Ces trois étapes sont répétées pour chaque itération k.

Etape 1 : Génère la solution initiale $Sref_0$

Sref {0101010010} => {2, 4, 6,9}

1	2	3	4	5	6	7	8	9	10
0	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**

Tableau 4.2 : codage de $sref_0$

Etape 2 : calcule Expected support de transaction à partir $Sref_0$

ExpSup{2,4,6,9}= (0.72*0.80*0.75*0.85)+(0.84*0.44*0.02*0.79) =0.3730

Etape 3 : comparer expected support avec minsup

ExpSup{2,4,6,9} >MinSup alors itemsets fréquent.

Etape 4 : Enregistrée la meilleure solution dans tabou dance.

5. Conclusion

Dans ce chapitre nous avons présenté l'algorithme de colonie d'abeille, ces fondamentaux ainsi que son principe de fonctionnement. Aussi, nous avons explicité l'approche proposée pour l'extraction d'itemsets fréquents à partir des données incertaines basées sur les colonies d'abeilles (BSO-UFIM).

Le chapitre suivant est consacré à démontrer l'efficacité de notre méthode grâce au résultat obtenue.

Chapitre 5 :

Test et Validation

Chapitre 5 : Test et Validation

1. Introduction

Après avoir décrit notre solution, nous aborderons dans ce chapitre la partie test et validation de notre application. En première partie, nous présenterons l'environnement de travail et les outils de développement utilisés. Ensuite, nous présentons notre jeu d'essai. Enfin, nous présenterons notre application ainsi que son fonctionnement et les résultats.

2. Présentation de l'environnement de travail

2.1.Systèmes d'exploitation

Nous utilisons le système Windows 7 son type est. 64-bit, sa RAM et de 4GO, son Disque dure et de 500 GO Le processeur est Intel(R) Core(TM) i3 CPU M 330 @ 2.13GHz 2.13 GHz

2.2.NetBeans

NetBeans est un projet open source ayant un succès et une base d'utilisateur très large. Sun Microsystems a fondé le projet open source NetBeans en Juin 2000 et continue d'être le sponsor principal du projet. Conçu en Java, NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC), Mac OS X . Aujourd'hui, deux projets existent:

– **EDI NetBeans** : c'est un environnement de développement, un outil pour les programmeurs pour écrire, compiler, déboguer et déployer des programmes. Il est écrit en Java, mais peut supporter n'importe quel langage de programmation. Il y a également un grand nombre de modules pour étendre l'EDI NetBeans.

– **Plateforme NetBeans** : c'est une fondation modulable et extensible utilisée comme brique logicielle pour la création d'applications bureautiques. Les partenaires privilégiés fournissent des modules à valeurs ajoutées qui s'intègrent facilement à la plateforme et peuvent être utilisés pour développer ses propres outils et solutions.[22]

3. Présentation des données utilisées pour les tests

Les fichiers de donnée que nous avons utilisée sont des fichiers qui contiennent des items avec leurs probabilités existentielles. Ces probabilités sont générés en suivant la loi Normale avec des paramètres qui permet d'avoir des probabilité supérieur à 0.6 afin d'avoir un nombre important des itemsets fréquents même avec des valeurs de MinSup élevées.

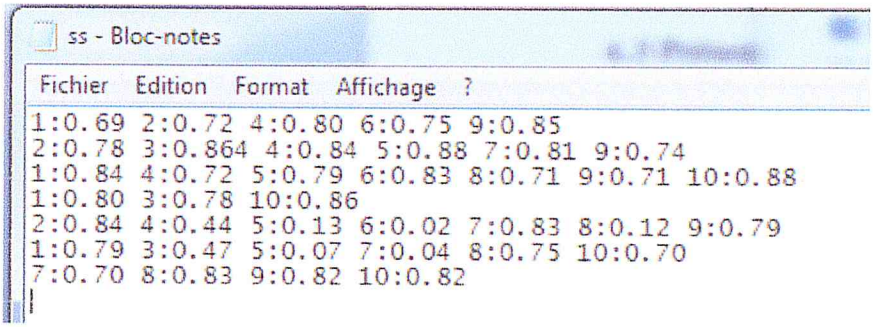


Figure 5.1: Fichier des itemsets

4. Présentation de l'interface utilisateur

Dans qui suit nous allons présenter les interfaces réalisées qui permettent à l'utilisateur d'avoir les résultats souhaités.

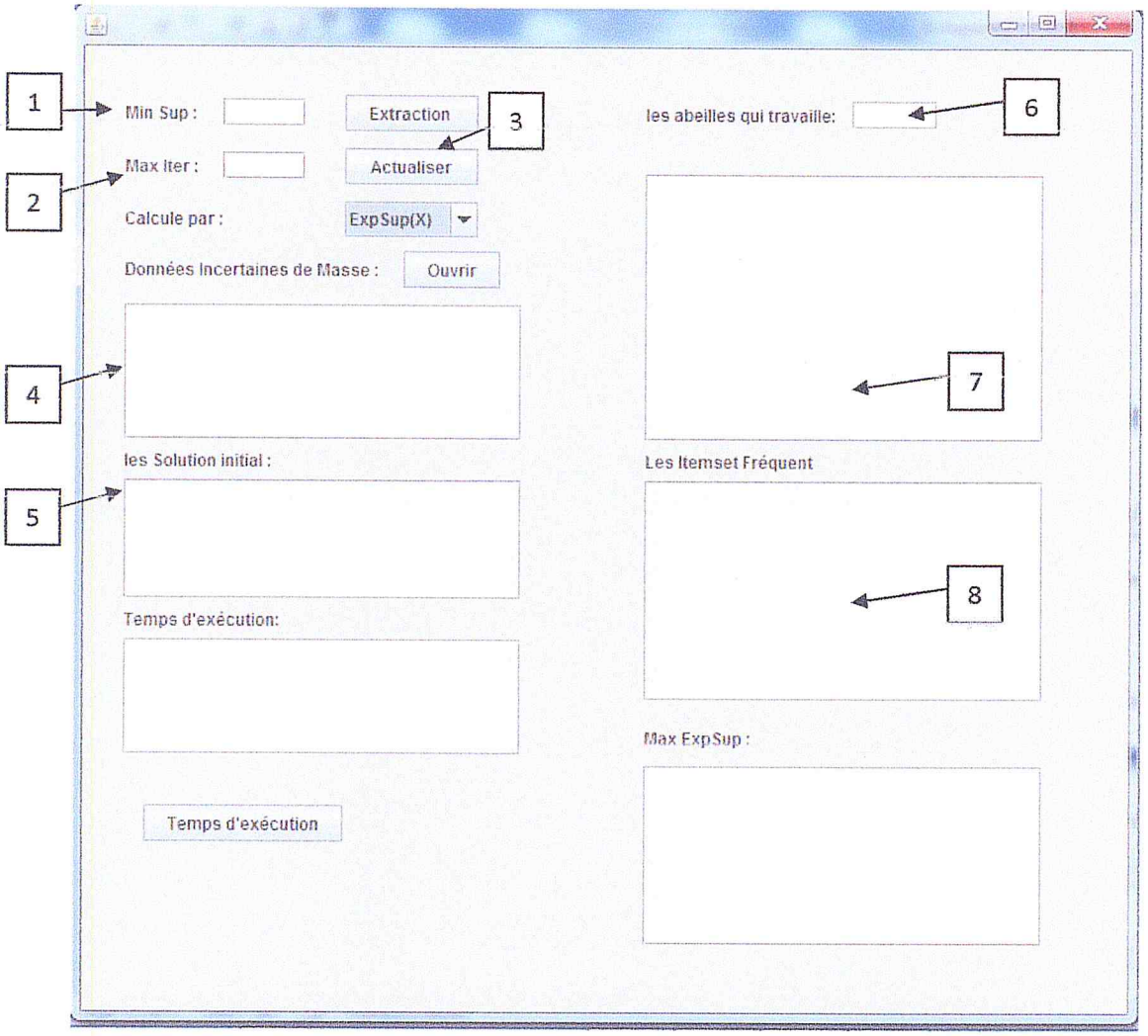


Figure 5.2: Interface pour la configuration

1 : Définir le Min-sup par utilisateur.

2 : Définir le max itération.

3 : le fichier des données (Data-set)

4 : le Sref initiale.

5 : Temps d'exécution de chaque d'abeille.

6 : Nombre d'abeille que travaille pour chaque itération.

7 : les itemsets fréquent (table dance).

8 : la solution de max ExpSup que va définie le sref de prochaine itération.

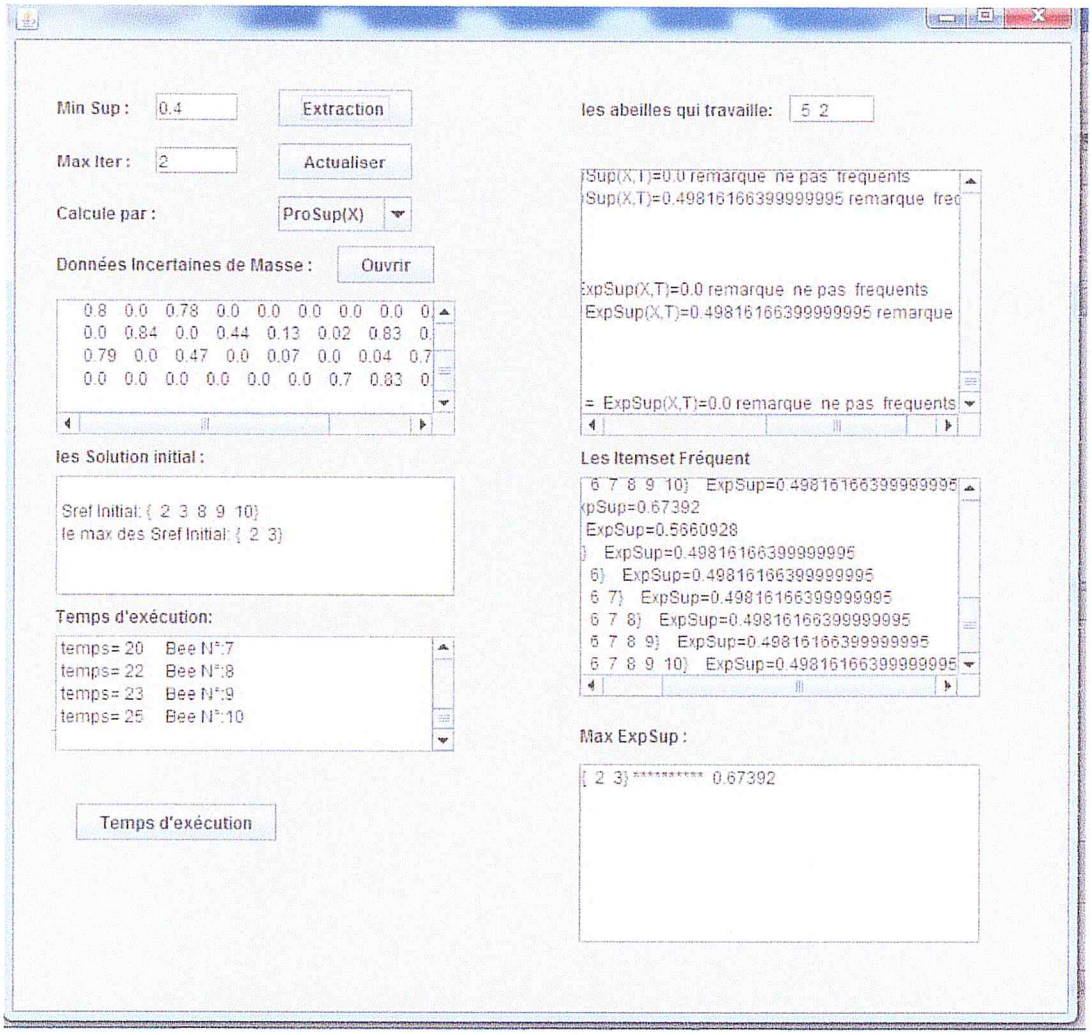


Figure 5.3: les résultats

5. Tests et validation

Nous avons validé notre algorithme avec des tests sur des données incertaines. Dans les tests suivants on va jouer sur les deux valeurs de MaxItération et minsup, et on compare leurs résultats avec le temps d'exécution en millisecondes.

5.1.Exécution parallèle en utilisant les threads

Le Muthreads va générer les solutions ou bien les zones de recherche, après il va créer un thread-B pour chaque zone de recherche.

Chaque thread-B représente une abeille qui calcule le ExpSup de chaque solution et le compare avec minSup. si la solution est supérieur a MinSup, elle s'ajoute dans table dance sinon, elle se supprime.

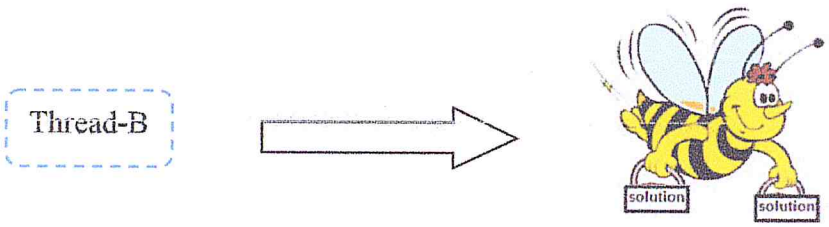


Figure 5.4: thread vers abeille

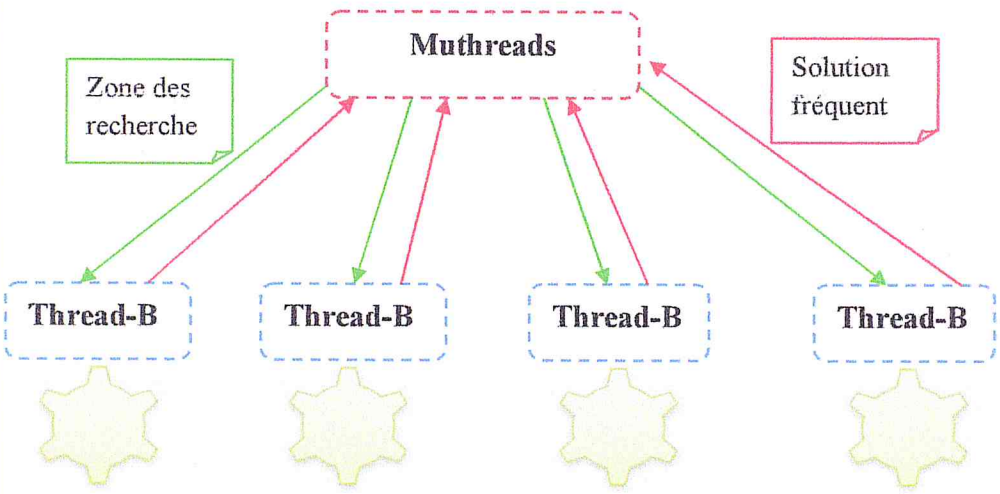


Figure 5.5: Architecteur générale d'application avec les threads

A. Premier test

Au début, dans le premier test, nous avons fixé la valeur de MinSup qui est égale à **0.4** et nous avons fait changer la valeur de nombre des itérations les résultats sont montré sur le tableau 5.1 le suivant :

MaxItération	Temps d'exécution (ms)
1	1056
3	2234
6	3760
9	6450
12	7892
15	8632

Tableau 5.1 : Résultat de test 1

On va présenter les résultats de test 1 dans le graphe suivant :

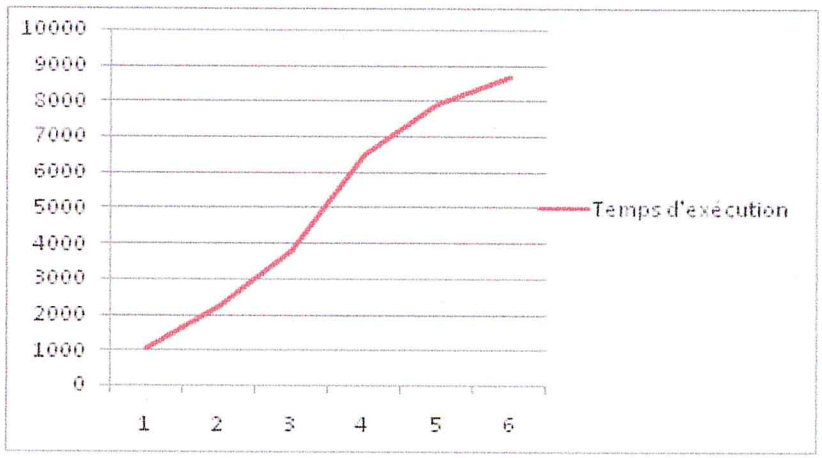


Figure 5.6: Temps d'exécution en fonction de MaxItération

Dans le graphe de la figure 5.6 (les résultats de test 1), nous remarquons que, à chaque fois que nous augmentons la valeur de Max Itération le temps d'exécution augmente aussi.

B. Deuxième test

Dans le deuxième test nous allons affiche la consommation de mémoire CPU, et nous avons joué sur la valeur de nombre des itérations les résultats sont montré sur le tableau 5.2 le suivant :

MaxItération	Mémoire(Mb)
1	12
3	26
6	50
9	66
12	82
15	124

Tableau 5.2 : Résultat de test 2

On va présenter les résultats de test 2 dans le graphe suivant :

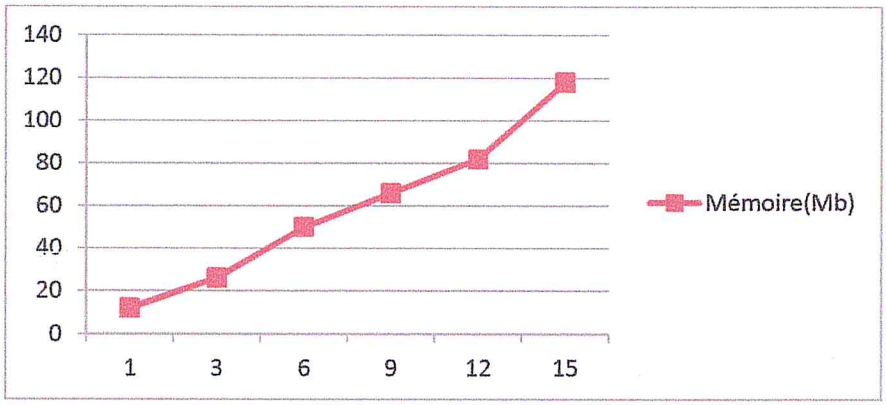


Figure 5.7 : Consommation de mémoire en fonction de MaxIter

Dans le graphe de la figure 5.7 (les résultats de test 2), nous remarquant que, à chaque fois que nous augmentons la valeur de Max Itération le consommation de mémoire augmente aussi.

6. Conclusion

Nous avons présenté dans ce chapitre l'environnement de notre travail et les outils de développement utilisés et la base de données qu'on a utilisé pour valider notre travail, nous avons aussi présenté les différentes interfaces de notre application, puis nous avons présenté notre résultat obtenu.

Conclusion Générale

Conclusion générale

1. Conclusions

Le traitement de données est un domaine de recherche en plein essor et devient de plus en plus important. Aussi, la communauté scientifique cherche développe de nouveaux outils et méthode de traitement. Au cours de notre travail nous avons vue L'extraction d'itemsets fréquents qui est une méthode de traitement de données. Et afin de comprendre et cerner la problématique de l'extraction d'itemsets fréquents à partir de données incertaines, nous avons étudié les différents algorithmes proposés dans la littérature dans ce contexte.

La méthode d'optimisation par colonie d'abeille est l'une des récentes méthodes d'optimisation. Elle est représentée par un algorithme qui peut être appliqué à de nombreux problèmes d'optimisation dans le management, l'ingénierie, et le contrôle.

Elle est basée sur le concept de coopération qui rend les abeilles plus efficaces et ainsi arrivées à leurs buts rapidement. Cette méthode a la capacité, grâce à l'échange d'informations et le processus de recrutement d'intensifier la recherche dans les régions prometteuses de l'espace de solutions.

Des résultats préliminaires ont montré que le développement de nouveaux modèles basés sur les principes des abeilles doit certainement contribuer dans des problèmes assez complexes.

Ce travail nous a permis d'avoir des nouvelles connaissances sur les nouvelles technologies d'extraction d'itemsets fréquents, des notions sur les problèmes combinatoires et une nouvelle approche de leur résolution.

2. Perspectives

Les perspectives de ce travail préliminaire sont nombreuses. Tout d'abord, nous devons valider la démarche proposée avec des bases de données réelles.

Il est probablement intéressant de pouvoir créer des méthodes qui lance plusieurs machines sur plusieurs bases de données au même temps pour extraire des itemsets fréquentes à partir des données incertaines dans le contexte de big data.

Bibliographie

[01] FREQUENT ITEMSETS par Tanagra le 3.10.11, Octobre 2011.

[02] R. Lovin, "Mining Frequent Patterns", Avril 2012

[03] R. Agrawal and R. Srikant. Fast algorithms for mining association rulesin large databases. In Proceedings of the 20th international conferenceon Very Large Data Bases (VLDB'94), pages 478-499. Morgan Kaufmann, September 1994.

[04] R. Srikant. Fast algorithms for mining association rules and sequentialpatterns. PhD thesis, University of Wisconsin, 1996.

[05] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD international conference on Management of Data (SIG-MOD'93), pages 207-216. ACM Press, May 1993..

[06] R. J. Bayardo. Efficiently mining long patterns from databases. In Proceedings of the 1998 ACM SIGMOD international conference on Management of Data (SIGMOD'98), pages 85-93. ACM Press, June 1998.

[07] Chun Kit Chui and Ben Kao. A decremental approach for mining frequent itemsets from uncertain data. In The 12th Paci_c-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), pages 64/75, 2008.

[08] M.Muzammal an R.Ramran.On Probabiliste Models for Uncetain Sequential Pattern Mining.pp.4-6.

[09] M.A. Soliman, I.F. Ilyas, and K. Chen-Chuan Chang. "Top-k Query Processing in Uncertain Databases". In Proc. 23rd Int. Conf. on Data Engineering, Istanbul, Turkey, pages 896/905, 2007.

[10] P. Agrawal, O. Benjelloun, A. Das Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. "Trio: A system for data, uncertainty, and lineage". In Proc. Int. Conf. on Very Large Databases (VLDB'06), 2006.

[11] N. Dalvi and D. Suciu. "E_cient query evaluation on probabilistic databases". The VLDB Journal,16(4):523{544, 2007.

Bibliographie

[12] C. Re, N. Dalvi, and D. Suciu. "E_cient top-k query evaluation on probalistic databases". In Proc. 23rd Int. Conf. on Data Engineering, Istanbul, Turkey, 2007.

[13] Leung, C.K.-S., Hao, B.: Mining of frequent itemsets from streams of uncertain data. In: IEEE ICDE 2009, pp. 1663–1670 (2009).

[14] Tan, P.-N., Chawla, S., Ho, C.K., Bailey, J. (eds.) PAKDD 2012, Part II. LNCS (LNAI), vol. 7302, pp. 322–334. Springer, Heidelberg (2012).

[15] Jin-Kao Hao et Christine Solnon Méta-heuristiques et intelligence artificielle.

[16] D. Karaboga, *An idea based on honey bee swarm for numerical optimization*, Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[17] Glover F.(1986), Future paths for integer programming and links to artificial intelligence, *Computers & Operations Research*, 13 (5), pp-533-549.

[18] Djenouri Y., Amine Chemchem (Aug. 2013), A Hybrid Bees Swarm Optimization and Tabu Search Algorithm for Association, *Nature and Biologically Inspired Computing (NaBIC), 2013 World Congress*, pp-120 -125.

[19] Han,J., Kamber, J. and Pei, M. (2006), Data Mining: Concepts and Techniques, *Elsevier 2nd edition*.

[20] NN Das, Anjali Saini (November 2013), A Study on Association Rule Mining Using ACO Algorithm for Generating Optimized ResultSet, *International Journal of Computer Science and Mobile Computing, IJCSMC*, 2(11), pp 123 – 128.

[21] S.MOUASSA .Optimisation de l'écoulment de puissance par une methode métaheuristique (technique des abeilles) en presence d'une source renouvelable (Éolienne) et des dispositifs FACTS.

[22] http://jade.tilab.com/ Dernière consultation : Avril 2016.