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

Université Saad Dahlab Blida

Faculté des sciences

Département d'informatique

Mémoire Présenté par :

BOUAFIA Nabila

En vue d'obtenir le diplôme de Master

Domaine : Mathématique et informatique

Filière : Informatique
Spécialité : Informatique
Option : Ingénier logicielle

Thème

Optimisation par colonies d'abeilles pour l'extraction des itemsets fréquents à partir de données évidentielles

M^ME.Boumahdi fatima	Président
M^R.BENAISSI	Examinateur
Mme ZAHRA FATMA ZOHRA	Promotrice

Promotion : 2018 / 2019

Remerciements

Je remercie 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.

Mes 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.

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

Je remercie les nombres de jury pour nous avoir fait l'honneur de juger notre travail.

Je remercie également toute personne ayant contribué à notre éducation et notre formation.

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

Dédicace

Je dédie ce mémoire à:

Mes chers parents.

Mon marie

Mes enfants :

Soundous, Anes, Zaki

Mes sœurs :

Zahra , hanane

Mes frères :

Amar, Ahmed, Okba, Samir

Enfin à tous mes collègues et à ceux qui m’ont aidés et me souhaitent la réussite.

Résumé

L'extraction des motifs fréquents est une technique de Data Mining qui se repose sur un principe relativement simple. Elle consiste à découvrir des structure de données (appelées motifs) qui se répètent fréquemment dans un ensemble de données. Ce problème est considéré comme un problème NP-Difficile. De plus, les données collectées dans plusieurs domaines sont imparfaites (incomplètes, incertaines, et/ou imprécises), l'extraction des motifs fréquent à partir de ce genre données a attiré l'intention des chercheurs.

Notre travail consiste à traiter l'extraction des motifs fréquents à partir de données imparfaites comme étant un problème d'optimisation combinatoire. En effet, une métaheuristique inspirée du comportement des abeilles (Bee Swarm Optimization) a été adaptée à l'extraction des itemsets fréquents.

Les tests ont montré que notre méthode permet d'améliorer le temps d'exécution. Par contre, elle souffre de problème de qualité des résultats, étant une méthode approchée, elle ne peut pas fournir tous les itemsets évidentiels fréquents mais seulement une partie qui représente les itemsets les frénquents.

Mots clés : ***Extraction de motifs, Bee Swarm Optimization, BSO, données imparfaites, théorie d'évidence, Extraction des itemsets évidentiels fréquents.***

Abstract

Frequent pattern extraction is a Data Mining technique that relies on a relatively simple principle . It consists of discovering data structures (called patterns) that are repeated frequently in a set of data . This problem is considered an NP-Difficult problem. In addition , the data collected in several domains are imperfect (incomplete , uncertain , and / or imprecise) , the extraction of frequent patterns from this data has attracted the intention of researchers.

Our work consists in treating the extraction of frequent motifs from imperfect data as a combinatorial optimization problem.Indeed , a metaheuristic inspired by the behavior of bees (Bee Swarm Optimization) has been adapted to the extraction of frequent itemsets.

Tests have shown that our method improves the execution time. On the other hand, it suffers from a problem of quality of the results, being an approximate method, it can not provide all the frequent evidential itemsets but only a part which represents the itemsets and the frenquents.

As a result, the extraction of frequent patterns from imperfect data has attracted the interest of researchers in the data mining community, and several studies have been proposed in this direction.

In addition, the extraction of frequent itemsets is an NP-Difficult problem. The exact methods of extraction of frequent patterns suffer from the problem of scalability with a large number of items and a considerable amount of data.

Key words: Pattern extraction, Bee Swarm Optimization, BSO, imperfect data, evidence theory, Extraction of frequent evidential itemsets

ملخص

استخراج الانماط المتكررة هو تقنية لاستخراج البيانات تعتمد على مبدأ بسيط نسبيًا. يتكون من اكتشاف بنيات (تسمى الأنماط) التي تتكرر بشكل مستمر في مجموعة من البيانات. تعتبر هذه المشكلة مشكلة صعبة بالإضافة إلى. ذلك ، فإن البيانات التي تم جمعها في العديد من المجالات غير مثالية (غير كاملة ، غير مؤكدة / أو غير دقيقة) ، وقد اجتذب استخراج أنماط متكررة من هذه البيانات اهتمام الباحثين.

يتمثل عملنا في معالجة استخراج العناصر المتكررة من البيانات غير المكتملة باعتبارها مشكلة تحسين اندماجي. ,.مستوحاة من سلوك النحل (سرب النحل) في الواقع لاستخراج مجموعات متكررة

لقد أظهرت الاختبارات أن طريقتنا تعمل على تحسين وقت التنفيذ. من ناحية أخرى ، فإنه يعاني من مشكلة في جودة النتائج ، كونه طريقة تقريبية لاستخراج جزء من المجموعات المتكررة التي ليست بالضرورة مثالية

الكلمات المفتاحية: ، نمط الاستخراج ، تحسين سرب النحل ، بيانات غير مثالية ، نظرية الأدلة ، استخراج مجموعات الدليل المتكرر

TABLE DES MATIERES

TABLE DES MATIERES

Chapitre II

Extraction des itemsets fréquents à partir de données évidentielles

Chapitre III

Les Métaheuristiques

TABLE DES MATIERES

TABLE DES MATIERES

Liste des abréviations

EIF : Extraction des **I**temsets **F**réquents.

ECD : Extraction des **C**onnaissance à partir des **D**onnées.

DSET: Dempster Shafer Evidential Théory.

bba : La masse de croyance élémentaire.

Foc : Elément focaux.

BoE : Corps d'évidence.

Bel : La fonction de croyance.

Pl : La fonction de plausibilité.

FP-growth : Frequent Pattern growth.

FP-Tree : Fréquent Pattern tree.

expSup : Expected support.

BIT: Belief Itemset Tree.

FIM : Frequent Itemset Maintenance .

EDB : Evidentiel Data Base .

ER-Apriori : Evidential Reliable Apriori.

FBD : fuzzy base de données.

BSO UFIM-: Uncertaint frequent Item mining –Bee Swarm Optimisation

Listes des figures

Liste des Tableaux

Introduction générale

Introduction générale

1. Contexte et motivation

Vu la capacité naturel de l'être humain de prendre des décisions en présence d'une grande masse de données souvent imparfaites (incomplètes, incertaines ou imprécises...), l'automatisation du traitement de ce type de données a fait l'objet de plusieurs recherches dont l'objectif était de proposer des modèles et des méthodes afin de représenter et de manipuler ce type de données pour faire face à des situations réelles. En effet, plusieurs théories ont été proposées dans la littérature pour modéliser les données imparfaites. Parmi elles, on cite trois principales théories qui permettant d'intégrer la représentation des connaissances imparfaites: la théorie des probabilités qui traite l'incertitude des données, la théorie des possibilités qui traite l'imprécision des données et la théorie des croyances (appelée aussi théorie de Dempster Shafer ou théorie d'évidence) qui traite à la fois l'incertitude, l'imprécision et l'incomplétude des données.

En se basant sur ces modèles mathématiques, de nouveaux types de base données qui peuvent gérer les données incertaines ou imparfaites ont été introduits. Telles que, les bases de données probabiliste, possibiliste et récemment les bases de données évidentielles. Ces dernières peuvent représenter et intégrer trois aspects d'imperfections (imprécision, incertitude et incomplétude).

L'extraction des connaissances à partir ce genre de données, en d'autres termes l'adaptation des méthodes et algorithmes de Data Mining afin de découvrir des connaissances à par de données imparfaites a suscité l'intention des chercheurs. Notre travail s'intègre dans ce contexte.

2. Problématique

Etant l'étape cœur de plusieurs méthodes de Data Mining, l'extraction des motifs fréquents (EIF) à partir de données a reçu une attention particulière de la part de la communauté des chercheurs dans ce domaine. La majorité des travaux proposés dans la littérature dans ce contexte traitent les données précises.

Néanmoins, comme on avait dit, les données provenant du monde réel sont souvent incertaines, imprécises et en général imparfaites. En médecine par exemple, les médecins se trouvent souvent dans l'obligation d'émettre un diagnostic en présence de symptômes imprécis voire incertains. Les données générées par certains systèmes à base de capteurs sont aussi imparfaites. Les capteurs d'un même système peuvent produire des informations à

différents niveaux de confiance. Par conséquent, l'extraction des motifs fréquents à partir de données imparfaites a suscité l'intérêt des chercheurs de la communauté de Data Mining.

En outre, L'extraction des itemsets fréquents est un problème NP-Difficile. Les méthodes exactes d'extraction des motifs fréquents soufrent du problème de passage à l'échèle avec un nombre important des items et un volume de données considérable.

Par conséquent, ce dernier peut être traité comme étant un problème d'optimisation combinatoire. Or, de nombreux problèmes d'optimisation combinatoire sont difficiles et ne pourront donc pas être résolus de manière exacte dans un temps raisonnable puisque la capacité de calcul des machines évolue linéairement alors que le temps nécessaire à la résolution de ces problèmes évolue exponentiellement.

Les méta heuristiques sont des méthodes qui permettent de concevoir des algorithmes pour la résolution des problèmes d'optimisation complexe. Elles sont souvent inspirées par des systèmes naturels, qu'ils soient pris en physique (cas de recuit simulé), en biologie de l'évolution (cas des algorithmes génétiques) ou encore en éthologie (cas des algorithmes de colonies de fourmis ou de colonie d'abeilles). Néanmoins, un compromis entre la qualité des solutions obtenues et le temps de calcul doit être fait lorsqu'on veut utiliser Les métaheuristiques. Par conséquent ce sont des méthodes approchés et pas exactes.

3. Objectif

Dans ce travail, on va traiter l'extraction des motifs fréquents à partir de données imparfaites comme étant un problème d'optimisation combinatoire. En effet, une méta heuristique inspirée du comportement des abeilles (Bee Swarm Optmization) sera adaptée à l'extraction des itemsets fréquents. Par conséquent, l'objectif de ce travail est de développer une méthode approchée basé sur l'optimisation par colonies d'abeilles pour l'extraction des itemsets fréquents à partir de données imparfaites.

4. Organisation du mémoire

Le mémoire se réparti en Cinq chapitres.

Chapitre 1 : « Données incertaines »

Ce chapitre est consacré à la présentation des données incertaines et aux différents algorithmes (les plus connus) d'extraction de motifs fréquents.

Chapitre 2 : « Extraction des motifs fréquents à partir de données évidentielles»

Dans ce chapitre nous allons présenter l'extraction de motifs fréquente a partir de bases de données évidentielles ainsi une explication détaillée de la fonction de croyance.

Chapitre 3 : « méta heuristique »

Dans ce chapitre nous détaillerons les différentes solutions de résolution de problème d'optimisation.

Chapitre 4 : « Approche proposée et validation »

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

Chapitre 5 : « Implémentation et évaluation »

Dans ce chapitre nous présentons notre application ainsi que les tests et validation de la solution proposée.

Chapitre 1

Extraction des itemsets fréquents à partir de données incetaines

I. Introduction

L'incertitude des données est généralement causée par des facteurs tels que le caractère aléatoire et incomplet des données, les limites de l'équipement de mesure, la mise à jour tardive des données, etc…..

On utilise certains algorithmes pour trouver tous les modèles fréquents à partir de jeux de données incertaines probabilistes sont **U-Apriori**, **UF-growth**, **CUF-growth**, **PUF-growth [1]**. Lorsque les utilisateurs ne sont intéressés que par certains modèles fréquents plutôt que par tous, ces intérêts sont exprimés en termes de contraintes en réduisant l'espace de recherche. Enfin, l'ère du Big Data a engendré des défis, ainsi que des outils pour résoudre le problème de l'exploration des motifs frequent (FPM) de données incertaines.

II. L'extraction des itemsets

La recherche des régularités dans les bases de données est l'idée principale de l'exploration des données (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). **[02].**

Tableau 1 .1 : Base de transactions**[02].**

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. **[03].**

III. Modèle probabiliste pour les données incertaines

Dans un jeu de données probabiliste, PD (Probabilist Data) de données incertaines [4], l'utilisateur n'est pas certain de la présence ou absence d'un élément x dans une transaction ti. L'utilisateur peut deviner, mais ne peut pas garantie, la présence de x dans ti. Cette incertitude de présence peut être exprimée en utilisant probabilité existentielle P (x, ti), qui représente la probabilité que x soit présent dans ti dans PD. La valeur de P (x, ti) est comprise entre 0 et 1.

IV. L'exploration des motifs fréquents de données incertaines

L'exploration de modèles fréquente à partir de données incertaines PD a pour but de trouver chaque modèle X ayant satisfait à la condition expSup (X, D) minsup. Ainsi les motifs extraits sont appelés modèles fréquents basés sur le support attendu.

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

IV.1 Expected support

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

expSup (X)>= minsup ***[21]***

➢ **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.[3]

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

$$expSup(X) = \sum_{j=1}^{n} P(X, t_j) = \sum_{j=1}^{n} \left(\prod_{x \in X} P(x, t_j) \right)$$ [21]

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

IV.2 Probabiliste Support [05]

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 ***[5]***

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.

[5]

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

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,….., |T|}.

$$\sum_{0 \le i \le |T|} P_i(X) = 1.0. \quad [5]$$

Soit I l'ensemble de tous éléments possibles et T représente la base de données incertaines, où une transaction tj $\in$ T est un ensemble des éléments incertains, à savoir, tj $\subseteq$ I. Contrairement à la base de données précise traditionnelle, chaque item xi $\in$ tj est associée à une probabilité existentielle P (xi, tj) $\in$ (0,1], ce qui dénote la probabilité que xi est réellement présent dans tj. Pour un itemset X $\subseteq$ tj, basée sur l'hypothèse commune que les éléments de X sont indépendante la probabilité existentielle P (X $\subseteq$ tj) =$\prod$ x$\in$XP (x, tj). le soutien attendu (expected support) de X est alors la somme de la probabilité existentielle sur toutes les transactions. [6], [7],[8],

$$expSup\ (X) = \sum tj \in TP\ (X \subseteq tj). \text{ [6],}$$

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

$$P(X) = \sum_{i=minsup}^{|T|} P_i(X) \quad [6]$$

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

$$P_i(X) = \sum_{\substack{S \subseteq T \\ |S|=i}} \left(\prod_{t \in S} P(X \subseteq t) \prod_{t \in T-S} (1 - P(X \subseteq t)) \right) \quad [6]$$

Si P (X) $\geq$ 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) $\geq$ minProb.**

V.Les Algorithmes d'extraction d'items fréquents

Dans cette section nous allons présenter une analyse pour les algorithmes EIF.

Il existe deux approches des algorithmes d'extraction des itemsets fréquents à partir des données incertaines **[10]** :

Tableau 1.2: Algorithmes d'extraction des itemstes fréquents.

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

V. 1 Exploration de modèles fréquente incertaine à l'aide d'un candidat

a) Générer et tester (U-Apriori):

Chui et al. ont proposé l'algorithme U-Apriori [1], modification de l'algorithme Apriori [9]. L'algorithme U-Apriori utilise le candidat générer-et-tester, le paradigme dans une large et large méthode ascendante.

b) l'algorithme U-Apriori:

Étape 1: Calcule le support attendu (expected support) de tous les éléments de domaine. Les éléments avec **expSup>=min_sup** deviennent tous les modèles fréquents cohérents d'un élément.

Étape 2 : L'algorithme applique de manière répétée le processus de génération-test pour générer des (k + 1) ensembles d'éléments candidats à partir de k-ensemble éléments fréquents et pour vérifier s'ils sont fréquents (k + 1) ensemble éléments.

L'efficacité de l'algorithme peut être améliorée en incluant la stratégie de rognage LGS (lGS trimming)[1] .Cette stratégie supprime tous les éléments avec une probabilité existentielle inférieure au paramètre de réduction spécifié seuil (qui est local pour chaque élément) de l'ensemble de données probabiliste d'origine D de données incertaines, puis extrait les modèles fréquents de l'ensemble de données découpé résultant DTrim.

❖ **avantage de l'algorithme U Apriori:**

✓ Cet algorithme réduit considérablement la taille de l'ensemble de candidat

❖ ***Inconvinients:***

- la création de DTrim entraîne des frais généraux
- Il analyse la base de données plusieurs fois et ainsi la performance est affectée
- l'efficacité de l'algorithme est sensible au pourcentage d'éléments ayant une faible probabilité existentielle.

V. 2 Exploration de modèles fréquente incertaine à l'aide d'arborescences:

Comme alternative à l'extraction basée sur Apriori, l'extraction basée sur des arbres évite de générer de nombreux candidats. Ils utilisent une approche de profondeur **diviser pour reigner** du modèles fréquents à partir d'une arborescence qui capture le contenu de l'ensemble de données probabilistes.

V. 2.1 The UF-growth:

c'est un algorithme basé sur des arbres permettant d'exploiter des données incertaines afin de trouver les itemsets fréquents [11].

Les deux étapes principales de cet algorithme sont

• la construction UF tree.

• l'extraction de motifs fréquents à partir d'arbres UF.

Pendant la construction de l'arbre UF,chaque nœud conserve les informations sur un élément,le support attendu de l'élément et le nombre d'occurrences pour un tel article .L'algorithme de croissance UF construit l'arbre

UF en scannant l'ensemble de données deux fois. Lors de la première analyse, il lit l'ensemble des données une fois et rassemble le expSup de chaque élément. Il ne conserve ensuite que les éléments de la liste dont on attend support minsup et les arrange dans l'ordre décroissant de leur expSup. Lors de la seconde analyse, les éléments de chaque transaction sont insérés dans l'arbre UF. Les chemins dans l'arbre UF ne sont partagés que s'ils ont le même article et la même probabilité existentielle conduisant à des arbres plus grands.

Tableau 1.3 Base de données incertaine.[11]

TID	Contents
t1	{a:0.5, b:0.8, c:0.5, e:0.6}
t2	{a:0.7, b:0.6, c:0.6, d:0.7}
t3	{a:0.3, c:0.8, e:0.5}
t4	{a:0.8, c:0.3, d:0.2, e:0.7}

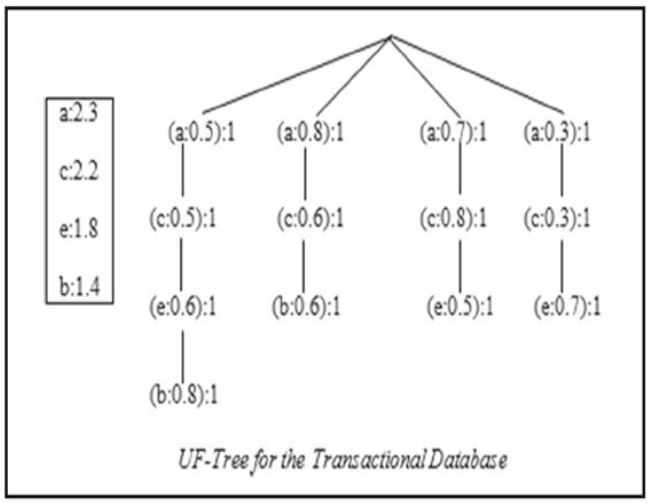

Figure 1.1' arbre UF (UF tree).[11]

- **Avantage de l'algorithme UF GROWTH**

Cet algorithme utilise des arbres UF pour extraire les motifs fréquents de bases de données incertaines dans deux analyses de base de données

- **Inconvénients**

- Il contient un chemin d'arbre distinct pour chaque élément distinct,

existentielle paire de probabilité.

V. 2.2 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.

V. 2.3 The CUF-growth

_Leung ET Tanbeer [12] ont proposé l'arborescence des modèles fréquents incertains plafonnés (capped)(CUF-tree) , une structure qui utilise l'arborescence pour représenter les éléments de la transaction et extrait également les modèles fréquents de l'arborescence. Ici, l'arbre CUF est construit en considérant une limite supérieure de la probabilité existentielle pour chaque transaction qui est appelé CAP plafond **Cap** de la probabilité existentielle de transaction.

- **Définition:** Le plafond d'une transaction ti, noté $p^{cap}(ti)$, est défini comme suit: le produit des deux valeurs de probabilité existentielle les plus élevées des éléments de ti. Soit h=|ti| représente la longueur ti,

  $M_1=\max_{q \in [1,h]} P(x_q,t_i)$ et $M2= \max_{r \in [1,h], r \neq q} P(x_q,t_i)$.[12]

$$P^{cap}(t_1)=\begin{cases} M_1 * M_2 si\ h>1 \\ P(x_1,t_1) si\ h=1 \\ ¿. \end{cases} [12]$$

✓ **La construction de l'algorithme CUF TREE,**

L'arbre CUF est construit dans deux balayage de la bases de données ,

1- Lors de la première analyse de la base de données, expected support de chaque élément de domaine est calculée, éliminant ainsi les éléments peu fréquents, puis tous les éléments fréquents sont triés par ordre décroissant de leur **expected support**,

2-Lors de la deuxième analyse de la base de données, l'arborescence CUF est construite et les CAP de transaction sont calculés en même temps.

3- Les éléments de la transaction sont insérés dans l'arborescence CUF en fonction de l'ordre de la liste triée et la valeur CAP de la transaction est ajoutée à chaque nœud en fonction de l'ordre de la liste triée.

Tableau 1.4 transaction de base de données en utilisant MinSup=1.0.[12]

TID	Contents	Contents (after 1st scan)	P^{Cap}
t1	{a:0.5, b:0.8, c:0.5, c:0.6}	{a:0.5, b:0.8, c:0.5, c:0.6}	0.48
t2	{a:0.7, b:0.6, e:0.6, d:0.7}	{a:0.7, b:0.6, e:0.6}	0.42
t3	{a:0.3, c:0.8, e:0.5}	{a:0.3, c:0.8, e:0.5}	0.40
t4	{a:0.8, e:0.3, d:0.2, e:0.7}	{a:0.8, e:0.3, e:0.7}	0.56

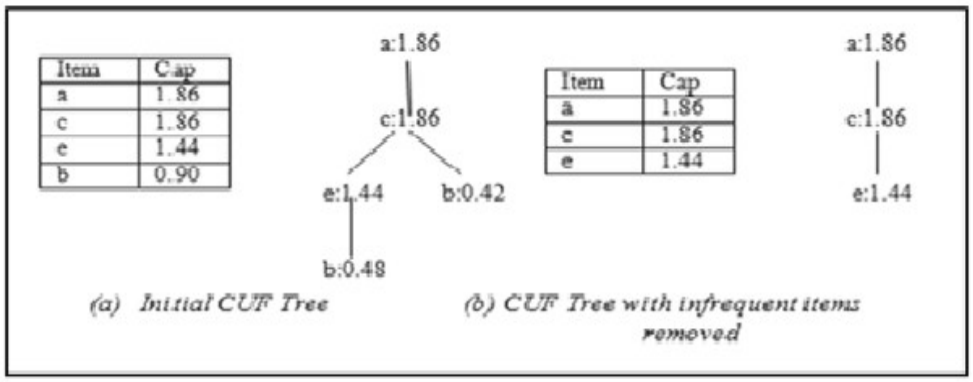

Figure1.2 l'arbre CUF (CUF tree)[12]

V.2.4 The PUF-growth:

Pour réduire la taille de l'UF-arbre et UFP-arbre une structure d'arbre des itemsets fréquents incertains préfixe capé (PUF-arbre) a été proposé [12], 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) $I^{cap}(x_r, t_j)$ 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):

$$I^{cap}(x_r,t_j) = \begin{cases} P(x_r,t_j) \times M \; if \; h > 1 \\ P(x_1,t_j) \quad if \; h = 1 \end{cases}, \text{ where M} = max_{1 \le q \le r-1} \, P(x_q,t_j) \qquad [12]$$

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

$$expSup^{cap}(X) = \sum_{j=1}^{n} \{(I^{cap}(x_k,t_j) | X \subset t_j)\} \qquad [12]$$

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

Exemple :

Tableau 1.5 : Une base de données de transactions avec minsup = 0,5[12]

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}

Considérons une base de données incertaine avec quatre transactions présentées dans la deuxième colonne du tableau 1.5 **[1*].** L'item plafond de c dans t1 peut calculer comme suit :

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

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

- ✓ **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 **[13]**
- 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.

- ➢ **avantage de l'algorithme PUF GROWTH:**
  - ✓ Cet algorithme élabore un modèle fréquent avec la construction d'une base de données projetée pour chaque modèle fréquent potentiel et explore récursivement ses possibles extensions fréquentes

- ***Inconvinients:***
  - Cela crée des faux positifs.
  - leur performance globale dépend du nombre de faux positifs générés.

VI. Comparaison entre les algorithmes d'extraction d'item fréquents a partir de données incertaines :

Le tableau suivant donne une comparaison entre l'algorithme U_Apriori et l'algorithme basé sur les arborescences selon plusieurs critères :

Tableau 1.6 Comparaison entre U-Apriori et les algorithmes bases sur les arborescences.

Paramètre	UAprirori	Basés sur les arborescences
approche	Niveau de profondeur de bas en haut	Profondeur : diviser et conquérir approche
méthode	Générer et tester	Extraction récursive d'arbres
scan	multiple	2(UF)3(CUF ET PUF)
Approbation et stratégie	LGS trimming	Caps
sensibilité	haute	Moyenne ou faible
évolutivité	Non linéaire	linéaire
Temps d'exécution	longue	court

Le tableau suivant donne une comparaison entre les l'algorithme base sue les arborescences selon plusieurs critères.

Tableau 1.7 comparaison entre les algorithmes basés sur les arborescences

	UF growth	CUF growth	PUF growth
Taille de l'arbre	large	faible	faible
caps	néant	transaction	Pré fixe élément
Faux positives	néant	moyen	faible
Temps déxecution	élevé	faible	faible

Mémoire demandée	**élevé**	**faible**	**faible**
Contenu des noeuds	▪ **données de élément** ▪ **valeur de probabilité** ▪ **occurrence**	▪ **Données de l'element** ▪ **Transaction Cap**	▪ **Données de l'élément** ▪ **Préfixe de l'élément Cap**

VII. Extraction de motifs fréquents incertains avec contrainte

Il existe de nombreuses situations réelles dans lesquelles l'utilisateur ne s'intéresse qu'à certains modèles fréquents, Leung et al. ont étendu l'algorithme UF GROWTH pour exploiter des ensembles de données probabilistes, de données incertaines pour des modèles fréquents satisfaisant les contraintes spécifiées par l'utilisateur, appelé algorithme U-FPS **[14]** qui permet de trouver efficacement des modèles fréquents contraints de données incertaines .l'algorithme et comme suit:

- 1-vérification par contrainte des transactions dans la base de données incertaines
- 2-la construction de structure de l'arbre (exp:UF tree)
- 3-l'exploration de motifs fréquents depuis UF tree,

VIII. Exploration des motifs fréquents incertains à partir de données volumineuses (Big Data)

Tout ensemble de données qui est difficile à capturer, nettoyer, analyser et visualiser à l'aide des technologies actuelles peut être appelé Big Data, En 2012, Gartner a révisé la définition du Big Data [8, 15, 16] en ces termes:

Actifs informationnels à haute vélocité et / ou à grande variété nécessitant de nouvelles formes de traitement permettre la prise de décision, l'améliorée, la découverte et l'optimisation des processus ».

Tout ensemble de données qui est difficile à capturer, nettoyer, analyser et visualiser à l'aide des des technologies actuelles peuvent être appelées Big Data. MapReduce [17] vu comme un framework supporte l'écriture et l'exécution d'applications pour le traitement d'énormes quantités de données, en parallèle, sur des clusters matériels de manière fiable. MapReduce vu comme un algorithme contient Map tâche et réduire la tâche.

Pour exploiter des modèles fréquents à partir de grands ensembles de données probabilistes de données incertaines [18], l'algorithme de croissance MR a été proposé[19]. il utilise MapReduce en appliquant deux ensembles, de fonctions «map» et «réduire», dans un motif environnement de croissance. Plus précisément, le nœud maître lit et divise un jeu de données probabiliste D de données incertaines en partitions, puis les affecte à un autre opérateur (workers nœuds.)

VIII.1 l'algorithme MRgrowth :

- 1-trouve tous les one-itemsets fréquents avec expected _support
- 2-construit ensuite les bases de données projetées appropriées en utilisant des arbres FP, des arbres CUF ou des arbres PUF pour trouver tous les k-itemsets fréquents (pour k >=2) avec leur expected_support,

X. Conclusion

Dans ce chapitre, on a résumé les algorithmes FPM sur des données incertaines, L'algorithme **U-Apriori** génère de nombreux candidats et teste s'ils sont fréquents ou non. Cette procédure de génération et de test est totalement évitée dans les algorithmes d'exploration de modèles fréquents basés sur des arbres, à savoir UF growth, CUF growth et PUF growth. Dans les chemins UF tree, les chemins sont partagés si leurs nœuds partagent le même élément**<item, probabilité existentielle>**conduisant à des arbres plus grands.

L'algorithme U-FPS permet à l'utilisateur d'introduire des contraintes dans le processus d'exploration en économisant beaucoup de temps. L'algorithme **MR-Growth** est utilisé pour extraire les modèles fréquents de Big Data à des fins d'analyse.

Les algorithmes mentionnés ci-dessus génèrent tous les motifs fréquents possibles mais si l'utilisateur n'est intéressé qu'a une partie des modèles fréquents, on utilise alors l'extraction de modèles fréquents limités.

Le chapitre suivant sera dédié à décrire les fondements théoriques de l'extraction d'items fréquents à partir de données évidentielles.

Chapitre 2

Extraction des itemsets fréquents à partir de données évidentielles

I. Introduction

Dans ce chapitre, nous étudions le problème de l'extraction des itemsets fréquents (EIF) à partir de données imparfaites, et plus particulièrement ce qu'on appelle désormais les données *évidentielles*. Une base de données *évidentielles* stocke en effet des données dont l'imperfection est modélisée via la théorie de l'évidence, un cadre formel pour le calcul et le raisonnement à partir d'informations partielles imparfaites (incertaines, imprécises).

II. Définition de la donnée imparfaite

A la rencontre d'un ensemble de données imparfaites, la première chose à faire est de donner un "sens" à ces données pour pouvoir les interpréter. Si les données dans le monde réel sont incomplètes ou incertaines, plusieurs scénarios et états sont possibles, mais on ne peut pas dire lequel de ces scenario représente l'état du monde réel. Donc une base de données qui contient des données incomplètes et / ou incertaines représente implicitement un ensemble d'états possibles et une description proche de la réalité.

La notion d'imperfection dans les données fait appel à quatre Concepts : l'imprécision, l'incertitude, l'incomplétude et l'erreur **[20]**. Illustrée sur la **Figure 2.1**

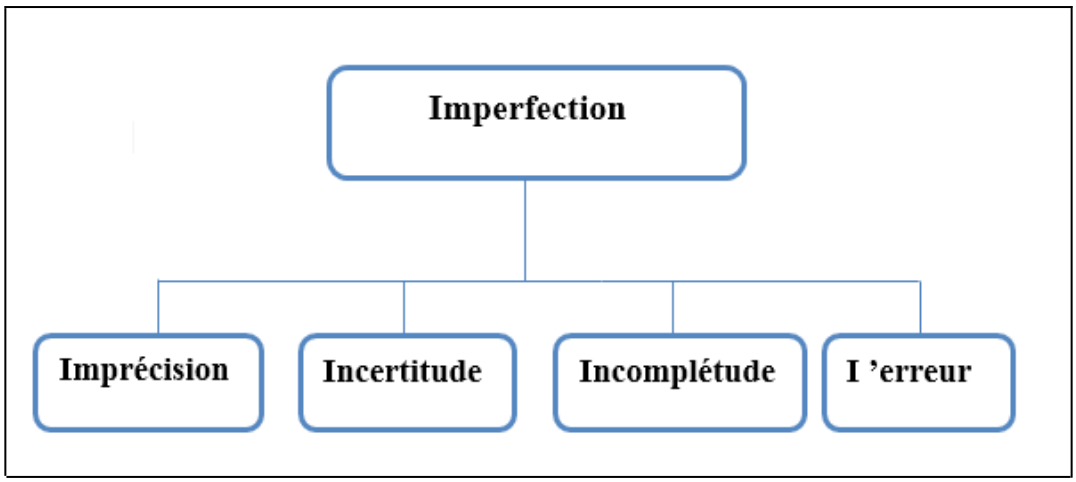

Figure 2 .1: Grands types d'imperfection. [**20]**

Les quatre concepts d'imperfection sont définis comme ci-dessous :

II.1 Imprécision

Elle correspond à une difficulté dans l'énoncé de la donnée, soit parce que des :

- Données numériques sont mal connues (causé par l'insuffisance des instruments d'observation, d'erreurs de mesure (poids à 1% près) ou encore de connaissances flexibles (la taille d'un adulte est environ entre 1.50 et 2 mètres)).

- Des termes du langage naturel sont utilisés pour qualifier une caractéristique du système de façon vague. (provient de l'interprétation spontanée de la donnée à titre d'exemple (température douce, grand appartement, proche de la plage) ou de l'utilisation de catégories aux limites mal définies (enfant, adulte, vieillard)).

Remarque : La donnée imprécise dénote un ensemble de valeurs possibles et la valeur réelle est l'un des éléments de cet ensemble de valeurs. Donc, elle n'est pas incorrecte et ne compromet pas l'intégrité des données.

II. 2 Incertitude

Parfois notre connaissance du réel ne peut pas être énoncée avec confiance ou garantie absolue. Car la donnée énoncée avec l'incertitude n'est pas incorrecte et ne compromet pas son intégrité. Bien que : l'âge d'un personne de 20 ou 24 ans est imprécis. La donnée : l'âge est probablement 20 est une donnée incertaine, dans quelques cas, le degré de la certitude est donnée : l'âge est 32 ans avec une probabilité de 0.6 et 33 avec une probabilité de 0.4.

Remarque : L'imprécision et l'incertitude sont deux notions très liées, on peut dans quelque cas modéliser l'imprécision par l'incertitude et vice versa. Plus la donnée est précise plus elle est certaine à titre d'exemple «je suis sûr que la note est entre 10 et 12 mais je ne suis pas certaine qu'elle soit 11 » ou bien « je suis certaine que je serais à l'université l'après-midi, mais je ne suis pas sûr que je serais la à 13h30min » si la valeur précise mais pas certaine est entourée par d'autres valeurs possibles ceci incrémente la certitude mais l'imprécision sera importante également.

II. 3 Incomplétude

La donnée est dite incomplète si elle contient au moins une valeur manquante, dans ce cas on a uniquement une donnée partielle du réel perçu. Les incomplétudes sont :

- Des absences de données ou des données partielles sur certaines caractéristiques de l'objet. Elles peuvent être dues à l'impossibilité d'obtenir certains renseignements (fichiers de malades dans lesquels certaines rubriques ne sont parfois pas remplies) ou à un problème au moment de la capture de la donnée.
- Elles peuvent aussi être associées à l'existence de données générales sur l'état d'un système, habituellement vraies, soumises à des exceptions que l'on ne peut pas énumérer ou prévoir, selon les cas, (" généralement, Pierre est à son bureau tous les jours ", sauf s'il est malade ou si un événement grave survient dans sa famille).

- Elles sont généralement liées à l'existence de données implicites, par exemple dans une recherche d'information auprès d'experts. Ces imperfection ne sont pas exclusives l'une de l'autre et l'incomplétude est toujours ramenée à l'imprécision.

II. 4 L'erreur

C'est le type le plus simple de donnée imparfaite. La donnée Stockée est incorrecte quand elle est différente de l'information vraie. Cependant d'après un balayage de la littérature il n'existe pas de représentation ou une modélisation précise pour les données erronées sauf que ce sont des données aberrantes.[20]

III. La théorie de Dempster-Shafer

III.1 Les concepts de base

La ***théorie de Dempster-Shafer***, appelée aussi ***théorie de l'évidence***, a été introduite par (Dempster, 1967) et formalisée mathématiquement par Shafer[37]. La théorie de Dempster-Shafer est souvent décrite comme une généralisation de la théorie bayésienne. Nous présentons dans cette section les concepts formels de cette théorie.

Soit $\Theta = \{\theta 1, \theta 2, ..., \theta n\}$ un ensemble fini non-vide incluant n hypothèses exhaustives et mutuellement exclusives. Θ est appelée ***cadre de discernement*** ou ***cadre d'intérêt*** du problème posé. Le ***référentiel de définition*** utilisé pour évaluer la véracité d'une proposition est constitué de tous les sous-ensembles possibles de Θ, soit l'ensemble 2^{Θ}. La ***fonction de masse élémentaire m*** d'une hypothèse $X \subseteq \Theta$, notée $m(X)$, représente la partie du degré de croyance placée sur X et qui n'a pas été distribuée aux sous-ensembles de X. La masse de croyance élémentaire, notée ***bba*** est définie par :

$$m : 2^{\Theta} \rightarrow [0, 1][37]$$

m : satisfait deux conditions qui trouvent leur correspondance dans la théorie des probabilités à savoir :

La masse de l'ensemble vide est nulle :

$$m(\emptyset) = 0[37]$$

La somme des masses de toutes les hypothèses correspond à l'unité :

$$\sum_{X \subseteq \Theta} m(X) = 1 \ [37]$$

Les sous-ensembles de Θ sur lesquels sont placées des masses strictement positives, sont appelés ***éléments focaux***, leur ensemble est noté *Foc*. Le triplet $\{\Theta, Foc, m\}$ est appelé ***corps d'évidence*** **noté *BoE***. La ***fonction de croyance (bel)*** est définie à partir de la fonction de

masse *m*, dans la mesure où la croyance d'un événement $A \subseteq \Theta$ reflète la croyance totale assignée à *A*, c.-à-d., la somme des masses de tous les sous-ensembles de *A*.[2]

$$bel(A) = \sum_{B \subseteq A} m(B) \text{[21]}$$

La *fonction de plausibilité (pl)* est la fonction duale de la fonction de croyance, elle mesure l'intensité avec laquelle une proposition *A* peut être considérée comme vraie. La plausibilité de *A* est donc la somme des masses de tous les éléments focaux qui sont compatibles avec *A*. [2]

$$\text{pl(A)} = \sum_{B \cap A \neq \varnothing} m(B) \text{[21]}$$

III.2 La monotonie de la fonction de croyance

La fonction de croyance est en effet monotone. Si une proposition *X* est incluse dans une proposition *Y,* alors la croyance de *Y* est supérieure à celle de *X*. Nous verrons que cette propriété est extrêmement importante pour hausser la performance de notre technique d'EIF.

$$\text{X} \subseteq \text{Y} \Rightarrow \text{bel(X)} \leq \text{bel(Y)[21]}$$

III. 3 La règle conjonctive de combinaison

Soient m1 et m2 deux distributions de masse définies pour le même cadre de discernement Θ et fournies par deux corps d'évidence indépendants. La règle conjonctive de combinaison (Smets, 1998) est applicable lorsque les sources d'informations, relatives aux distributions combinées, sont fiables. La règle conjonctive de combinaison est applicable à plus de deux distributions de masse :

$$m1 \cap m2(Z) = \sum_{X,Y \subseteq \Theta : X \cap Y = Z} m1(X) \times m2(Y) \text{[21]}$$

IV. Bases de données évidentielles

IV.1 Définition

Une base de données évidentielles permet le stockage de données incertaines et imprécises qui sont modélisées à l'aide de la théorie de Dempster-Shafer. Une base de données évidentielles notée *EDB* contient *n* attributs et *d* lignes. Chaque attribut *i* ($1 \leq i \leq n$) a un domaine Θ_i de valeurs discrètes. La valeur de la colonne *i* pour la ligne *j* est dite *valeur évidentielle* et notée V_{ij} . Elle correspond à une ***bba(***La masse de croyance élémentaire.***)*** qui est définie comme suit :

✓ **<u>Définition</u> (Valeur évidentielle) :** Soit V_{ij} une valeur évidentielle qui correspond à la colonne i et à la ligne j. V_{ij} correspond à un corps d'évidence composé du cadre de discernement Θ_i (le domaine de la colonne i), de l'ensemble F_{ij} des éléments focaux et de la fonction de masse m_{ij} qui est définie comme suit :

$m_{ij} : 2^{\Theta i} \rightarrow [0, 1]$ avec :

$m_{ij}(\emptyset) = 0$ et $\sum_{x \subseteq \Theta i} mij(x) = 1$.[21]

IV.2 Imperfection de données

Grâce à la théorie de Dempster-Shafer, plusieurs types d'imperfections peuvent être modélisés à travers les données sauvegardées dans une même base de données évidentielles. Ainsi, les données stockées peuvent être :

Tableau. 2.1 – *Exemple d'une base de données évidentielle*[21]

Id	A	B	C
1	***A*1(0.6)** ***A*2(0.4)**	***B*1**	***C*2(0.2)** ***{C*1, *C*2*}*(0.8)**
2	***A*1(0.2)** ***{A*2,*A*3*}*(0.8)**	***B*1**	***C*1(0.5)** ***C*2(0.5)**

<u>– Parfaites :</u> Dans une base de données évidentielles, nous avons mentionné plus haut que la valeur d'un attribut est un corps d'évidence. Quand ce corps d'évidence inclut exactement un

élément focal qui est singleton, et dont la masse est par conséquent égale à l'unité, nous parlons de donnée parfaite. Le tableau 1 est une base de données évidentielles exemple où la valeur de la colonne *B* est parfaite dans les deux lignes d'identifiants 1 et 2.

– Probabilistes : Quand le corps d'évidence inclut des éléments focaux singletons, là nous parlons d'une valeur probabiliste. Dans notre base de données exemple, la valeur de la colonne *C* dans la deuxième ligne est probabiliste.

– Possibilistes. Lorsque le corps d'évidence inclut des éléments focaux imbriqués, nous parlons de valeur possibiliste. En effet, la fonction π en théorie de possibilité correspond à la fonction *pl* en théorie, de Dempster-Shafer. Dans ce cas, la valeur de l'attribut *C* dans la première ligne est possibiliste avec $\pi(C1) = 0.8$ et $\pi(C2) = 1$[21]

– Manquantes : Si une la valeur d'un attribut *i* pour une ligne *j* est manquante, alors la *bba Vij* inclut un seul élément focal qui coïncide avec Θi, soit le domaine de l'attribut *i*, avec une masse égale à l'unité.

– Évidentielles : A l'instar de la valeur de l'attribut *A* dans la deuxième ligne qui n'est ni parfaite, ni probabiliste, ni possibiliste, ni manquante.[21]

V. Extraction des itemsets fréquents à partir de données imparfaite

Plusieurs théories ont été proposées dans la littérature pour modéliser les données imparfaites. Parmi elles, on cite trois principales théories qui permettant d'intégrer la représentation des connaissances imparfaites: la théorie des probabilités qui traite l'incertitude des données, la théorie des possibilités qui traite l'imprécision des données et la théorie des croyances (appelée aussi théorie de Dempster Shafer ou théorie d'évidence) [37]qui traite à la fois l'incertitude, l'imprécision et l'incomplétude des données .nous nous intéressons dans notre travail a la théorie des croyances.

V.1 Mesures de calcule des fréquences des itemsets imparfaites

V.1.1Evidential support

Une nouvelle approche a été introduite pour support itemset informatique[38], et appliqués sur un Fréquent Itemset Maintenance (FIM) problème. Toutes les méthodes [19][1] ont été basés sur le produit cartésien entre $BBAs$. Le support d'un itemset $X = \prod i \in [1....n]$ tel que xi est un item évidentiel appartenant au cadre de discernement θi. Étant donné que les items ne partagent pas le même cadre de discernement, toute règle de fusion ne peut pas être appliquée. Dans ce qui suit, nous étudions le support de croyance [10] calculé par l'équation suivante :

$$mj(X)=\prod_{x_i \in X} mij(x_i)[21]$$

Où $mj(X)$est le produit cartésien de tous *BBA* dans la transactionTj. Ainsi, le *BBA* de l'itemset X exprimé dans l'ensemble *EDB* devient :

$$m_{EDB}(X)=\frac{1}{d}\sum_{j=1}^{d} mj(X)[21]$$

Ensuite, le support de X dans la base de données *EDB*devient :

$$Support_{EDB}(X)=Bel_{EDB}(X)[21]$$

Le produit cartésien d'un support à base, présenté ci-dessus, remplit plusieurs propriétés telle que la propriété anti-monotonie. Une mesure de support satisfaisant la propriété anti-monotonie consiste dans le fait qu'un itemset qui contient un itemset fréquent est également fréquent. L'inverse est vrai, tous les itemsets constituant un des plus fréquents sont également fréquents. Avec cette propriété satisfaite, la construction d'un algorithme Apriori devient simple [38].

V.2 Approche basé sur le support évidentiel

L'extraction de données évidentielles n'a pas eu tant d'attention. En effet, il y a peu d'ouvrages qui ont abordé le thème de l'extraction des itemsets fréquents. Bach et al [21] ont proposé d'extraire des itemsets fréquents basés sur la représentation verticale de la base de données évidentielles en utilisant une structure de données de listes d'identificateurs d'enregistrement (RidLists). Cette structure de données contribue à accélérer le comptage de support évidentiel d'itemsets. La fonction de croyance basée sur le support évidentiel a été utilisée dans ce travail. Les mêmes auteurs ont proposé dans [39] une méthode basée sur la structure d'arbre pour extraire des itemsets fréquents à partir de bases de données évidentielles qui incluent exactement un attribut évidentiel. Ils ont utilisé (BIT) Belief itemset tree qui est une structure de données pour stocker une base de données évidentielles de manière compressée. Des itemsets fréquents sont générés par la suite à partir de cette structure de données arborescente.

L'algorithme de EDMA [40] est un algorithme basé sur Apriori- qui extrait les itemsets fréquents à partir des bases de données évidentielles en utilisant évidential precise support pour le calcul de la fréquence des itemsets.

Samet et al [40] ont étendu le concept de base de données évidentielles à des bases de données fiables qui tiennent en compte la fiabilité des données. Par conséquent. Pour les itemsets fréquents de ce type de données les auteurs ont proposé une nouvelle définition pour le support évidential des données fiables. La version classique de l'algorithme Apriori a été mise à jour pour extraire les itemsets fréquents à partir des données fiables. La nouvelle version est appelée Evidential Reliable Apriori (ER-Apriori).

Conclusion

Dans ce chapitre, nous avons étudié le problème de l'extraction des itemsets fréquents (EIF) à partir de données évidentielles. Une base de données évidentielle stocke en effet des données dont l'imperfection est modélisée via la théorie de l'évidence.

La théorie des fonctions de croyance constitue un cadre très général (englobant les formalismes ensembliste et probabiliste) pour la représentation et la manipulation d'informations imparfaites. Ce cadre est adapté à la résolution de problèmes complexes, particulièrement ceux impliquant :

- Des informations imparfaites (données partiellement supervisées, capteurs peu fiables, etc.) ;
- Une combinaison d'informations objectives (données) et subjectives (opinions d'experts, perceptions) : intégration d'informations a priori en classification, appréciation du contexte par des experts, etc. ;
- Des sources d'informations multiples (fusion multi-capteurs, commination multi-experts, méthodes d'ensemble en classification supervisée ou non).

Il existes plusieurs algorithmes d'extraction d'itemsets frequents a partir de données evidentiels telque ER-Apriori et EDMA.

Le chapitre suivant sera dédié à décrire les différentes solutions de résolution de problèmes d'optimisation, particulièrement les métaheuristiques.

Chapitre 3

Métaheuristique

I. Introduction

Un problème d'optimisation peut se ramener à un problème d'existence de solution de bonne qualité. Il consiste à rechercher une solution de qualité suffisante au regard d'un (des) critère(s) donné(s) et des objectifs à satisfaites. De nombreuses méthodes de résolution de différents problèmes ont été proposées dans la littérature. Puisque la performance de méthodes de résolution de problèmes est mesurée en termes de deux notions: la rapidité de la recherche et la qualité des solutions fournies, les méthodes de résolution de problèmes ont été classées en deux catégories: les méthodes exactes et les méthodes approchées.

II. Notions de base en optimisation

Deux types de problèmes d'optimisation sont distingués: des problèmes de minimisation et des problèmes de maximisation. Un problème d'optimisation (de minimisation ou de maximisation) est défini par un ensemble de données et un ensemble de contraintes. Un ensemble de solutions S est associé au problème d'optimisation. Parmi les solutions S, un sous-ensemble $X \subseteq S$ représente des solutions réalisables respectant les contraintes C du problème, à chaque solution s est associée une valeur f(s) qui représente sa qualité. La résolution du problème d'optimisation consiste à trouver une solution $s¿ \in X$ qui minimise ou maximise la valeur f(s).[22]

III. Les méthodes de résolution de problème d'optimisation

Nombreuses méthodes de résolution de problèmes de différentes complexités ont été proposées. Ainsi, une grande variété et des différences remarquables au niveau du principe, de la stratégie et des performances ont été discernées. Cette variété et ces différences ont permis de regrouper les différentes méthodes de résolution de différents problèmes en deux classes principales: la classe de méthodes exactes et la classe de méthodes approchées. L'hybridation des méthodes de ces deux classes a donné naissance à une pseudo classe qui englobe des méthodes dites hybrides (voir figure 3.1).

- **Les méthodes exactes** : Les méthodes exactes sont connues par le fait qu'elles garantissent l'optimalité de la solution mais elles sont très gourmandes en termes de temps de calcul et de l'espace mémoire nécessaire.
- **Les méthodes Approchées** : Ces dernières constituent une alternative aux méthodes exactes. En fait, elles permettent de fournir des solutions de très bonne qualité en un temps de calcul raisonnable.
- **Les méthodes hybrides** : chacune des méthodes déjà proposée a des avantages dont

on cherche à maximiser et des lacunes dont on cherche à combler. Partant de ce principe, beaucoup de chercheurs ont envisagé la combinaison des méthodes de résolution des problèmes afin de tirer profit des points forts de chacune et de proposer des alternatives plus efficaces et plus performantes.

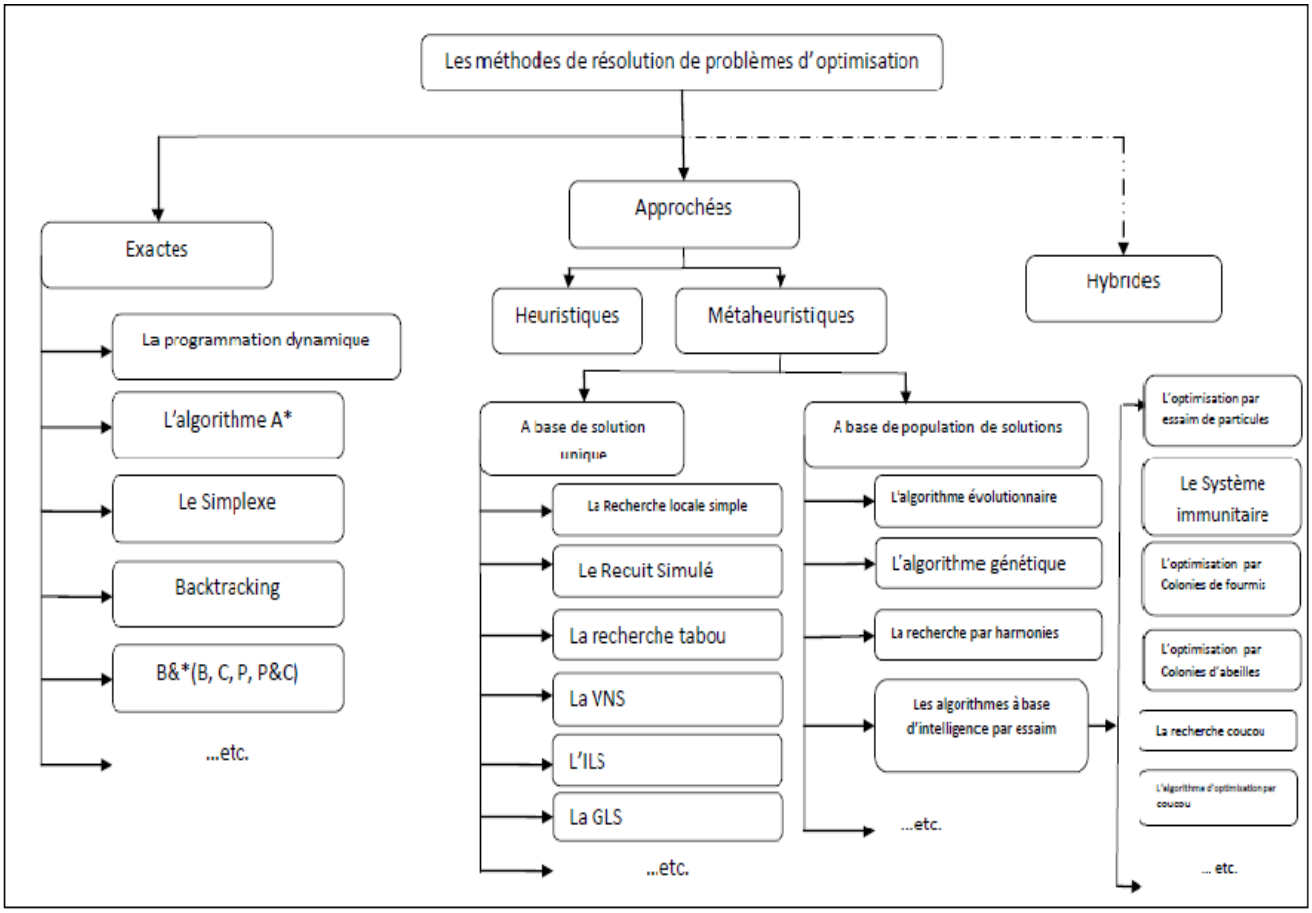

Figure 3.1 Classification de méthode de résolution de problème d'optimisation [22]

IV. La classe des méthodes exactes

L'intérêt des méthodes exactes réside dans le fait qu'elles assurent l'obtention de la solution optimale du problème traité. les méthodes exactes sont très connues par le fait qu'elles nécessitent un coût de recherche souvent prohibitif en termes de ressources requises. En effet, le temps de recherche et/ou l'espace mémoire nécessaire pour l'obtention de la solution optimale par une méthode exacte sont souvent trop grands, notamment avec des problèmes de grandes tailles. De ce fait, la complexité de ce type d'algorithme croit exponentiellement avec la taille de l'instance à traiter, elle devient très importante face à des problèmes comprenant plusieurs variables, fonctions «objectif»s et/ou critères. Il existe plusieurs méthodes que nous allons citer quelque unes dans ce qui suit.

IV. 1 L'algorithme de retour arrière (Backtracking)

Son principe est de parcourir les valeurs des variables par instanciation en remettant à cause la dernière affectation si une contrainte du problème est violée. Afin d'aboutir à une configuration consistante, une nouvelle valeur est affectée à la variable xi en respectant son domaine de définition. Le processus est répété à chaque itération jusqu'à l'émergence d'une solution complète. Si toutes les possibilités ont été essayées sans donner de bonnes solutions on dit que le problème est irréalisable.

- **Avantage :**
  - permet une exploitation complète de l'espace de recherche.
  - garantit l'aboutissement à la solution optimale.
- **Inconvénients :**
  - il est très lent face à un problème de n variables ayant chacune un domaine de définition de k valeurs.[22]

IV.2. La méthode Branch and Bound (B&B)

La méthode par séparation et évaluation (nommée Branch and Bound en anglais) est notée B&B, propose un mécanisme de recherche très intelligent, grâce à lequel elle permet une bonne exploitation de l'espace de recherche et l'aboutissement à la solution optimale plus rapidement que d'autres méthodes exactes.

Le principe de base de la méthode B&B se base sur la technique «Diviser pour régner ». Elle consiste à dissocier le problème en sous problèmes de manière à représenter le problème sous forme d'une arborescence, où chaque noeud correspond à une solution partielle. Les solutions partielles se forment de manière incrémentale en s'enfonçant dans l'arbre. Chacune des solutions partielles potentielles possède une borne supérieure et une autre inférieure. Ces dernières sont utilisées pour couper quelques branches de l'arbre et ainsi éviter d'explorer tout l'arbre. En fait, si l'évaluation partielle d'un noeud xi a montré que sa qualité est supérieure à la borne supérieure, le sous arbre en question sera élagué; sinon, le nœud sera divisé en sous nœuds. Ce processus se répète tant qu'il reste des branches non parcourues et la recherche continue jusqu'à trouver la solution optimale si elle existe.

Avantage :

Elle ne parcourt pas les sous branches dont on peut savoir à priori qu'elles ne permettent pas d'améliorer la solution rencontrée, cela permet de trouver de bonnes solutions en un temps de recherche raisonnable**.**

V.Les méthodes approchées

De nombreuses méthodes approchées ont été proposées. Elles sont plus pratiques pour la

résolution de problèmes difficiles ou de problèmes dont on cherche des solutions en un bref délai. Ces méthodes sont souvent classées en deux catégories: des méthodes heuristiques et des méthodes méta heuristiques (voir figure 2.1).

V.1. les méthodes heuristiques

Les méthodes dites heuristiques sont des méthodes spécifiques à un problème particulier. Elles nécessitent des connaissances du domaine du problème traité. En fait, se sont des règles empiriques qui se basent sur l'expérience et les résultats acquis afin d'améliorer les recherches future.

C'est une méthode qui aide à découvrir la solution d'un problème en faisant des conjectures plausibles mais faillible de ce qui est la meilleure chose à faire. »[26] .

V.2. les méthodes méta heuristiques

Les métaheuristiques construisent une solution moins exigeante. En fait, elles sont applicables sur une grande variété de problèmes d'optimisation de différentes complexités. Elles permettent de fournir des solutions de très bonne qualité (pas nécessairement optimales) en temps de calcul raisonnable.

La majorité des méta heuristiques sont inspirées des systèmes naturels et les solutions proposées dans la littérature sont partagées en deux classes: des métaheuristiques à base de solution unique et des métaheuristiques à base de population de solutions. Nous présentons quelques métaheuristiques des deux classes dans ce qui suit.

V.2.1 les métasheuristiques à base de solution unique

Elles débutent la recherche avec une seule solution initiale. et se basent sur la notion du voisinage pour améliorer la qualité de la solution courante. En fait, la solution initiale subit une série de modifications en fonction de son voisinage. Le but de ces modifications locales est d'explorer le voisinage de la solution actuelle afin d'améliorer progressivement sa qualité au cours des différentes itérations. Le voisinage de la solution s englobe l'ensemble des modifications qui peuvent être effectuées sur la solution elle-même. La qualité de la solution finale dépend particulièrement des modifications effectuées par les opérateurs de voisinages.De nombreuses méthodes à base de solution unique ont été proposées dans la littérature.

Parmi lesquelles: la descente, le recuit simulé, la recherche tabou, la recherche à voisinage variable etc..[22]

V.2.1.1 La recherche locale simple (la descente) :

La recherche locale simple ou la descente est un algorithme d'amélioration très ancien. Son principe consiste à explorer le voisinage de la solution courante afin d'améliorer sa qualité progressivement, comme le montre la figure 3.2 qui représente un schéma d'évolution d'une recherche locale simple. A chaque itération du processus d'amélioration, l'algorithme modifie un ensemble de composantes de la solution courante pour permettre le déplacement vers une solution voisine de meilleure qualité. Le processus est répété itérativement jusqu'à la satisfaction du critère d'arrêt. Il est à noter qu'il existe trois types de la descente: la descente déterministe, la descente stochastique et la descente vers le premier meilleur voisin.

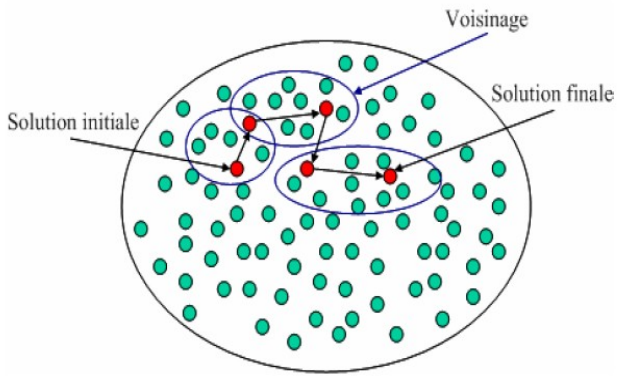

Figure 3.2 Un schéma d'évolution d'une recherche locale simple[22]

L'algorithme 3.1 résume les étapes de l'algorithme général de la descente.

F(s) : évalue la qualité de la solution donnée s.

F(s') :évalue la qualité de la nouvelle solution donnée s'.

Algorithme 3.1. La recherche locale simple (la descente)
Début
Construire une solution initiale s ;
Calculer la fitness f (s) de s ;
Tant que la condition d'arrêt n'est pas vérifiée **faire**
Modifier s pour obtenir une nouvelle solution voisine s' ;
Calculer f (s') ;
Si f (s') est meilleure que f (s) **alors**
Remplacer s par s' ;
Fin Si
Fin Tant que
Retourner s **;**
Fin

Figure 3.3 Algorithme de la recherche locale simple (la descente)[22]

V.2.1.2. Le recuit simulé

Son principe se base sur la procédure du recuit des métaux utilisée par les métallurgistes. Ces derniers chauffe à blanc le métal, puis ils laissent l'alliage se refroidir très lentement afin d'aboutir à un alliage sans défauts. C'est l'idée prise en considération par les métallurgistes qui savent que si le métal refroidit trop vite, il contiendra beaucoup de défauts microscopiques et s'il refroidit lentement ils obtiendront une structure bien ordonnée[23]. La métaheuristique du recuit simulé s'inspire de l'algorithme de Métropolies [24] dont le principe (pour un problème de maximisation) peut être résumé comme suit:

1. Entamer la recherche avec une solution initiale s;
2. Affecter une valeur initiale à la température T;
3. Calculer la fitness f(s) de la solution initiale s.
4. Générer une solution s' voisine de s;
5. Calculer la fitness f (s') de s' ;
6. Calculer l'écart de qualité (fitness) entre la solution s et la solution s' comme suit:
7. $$\Delta(f) = f(s') - f(s) \quad (2.1)$$
8. **Si** $\Delta(f) \geq 0$ **alors**
9. $s \leftarrow s'$
10. **Sinon** générer un nombre aléatoire $r \in [0,1]$;
11. **Si** r < exp($^{\Delta(f)}/T$) **alors** $s \leftarrow s'$.

Un schéma général de l'algorithme du recuit simulé (RS) est présenté dans l'algorithme 3.2.

Algorithme 3.2. Le recuit simulé
Début
Construire une solution initiale s ;
Calculer la fitness f(s) de s ;
Initialiser une valeur de la température T ;
s_{best}=s ;
Tant que la condition d'arrêt n'est pas satisfaite **faire**
Générer une solution s' voisine de s;
Calculer f(s') ;
Calculer $\Delta(f) = f(s') - f(s)$;
Si $\Delta(f) \geq 0$**alors** // cas de maximisation
$s_{best} = s'$;
$s = s'$;
Sinon **Si** r < exp($^{\Delta(f)}/T$) **alors**
$s \leftarrow s'$.
Fin Si
Décroitre la température T ;
Fin Tant que
Retourner s_{best} ;
Fin

Figure 3.4 Algorithme de recuit simulé[23]

L'acceptation d'une solution de mauvaise qualité est établi en fonction de deux facteurs: l'écart de qualité entre la solution courante et sa voisine d'un coté, et la température de l'autre coté. Plus la température est élevée, plus la probabilité d'accepter des solutions de mauvaises qualités est forte.

VI.2.1.3 La recherche tabou

C'est une méthode de recherche locale avancée, elle fait appel à un ensemble de règles et de mécanismes généraux pour guider la recherche de manière intelligente [25] L'optimisation de la solution avec la recherche tabou se base sur deux astuces: l'utilisation de la notion du voisinage et l'utilisation d'une mémoire permettant le guidage intelligent du processus de la recherche. En parcourant le voisinage de la solution courante s, la recherche tabou ne s'arrête pas au premier optimum local rencontré.

Elle examine un échantillonnage de solution du voisinage de s et retient toujours la meilleure solution voisine s', même si celle-ci est de piètre qualité que la solution courante s, afin d'échapper de la vallée de l'optimum local et donner au processus de la recherche d'autres possibilités d'exploration de l'espace de recherche afin de rencontrer l'optimum

global. [27]

En fait, les solutions de mauvaise qualité peuvent avoir de bons voisinages et donc guider la recherche vers de meilleures solutions. Cependant, cette stratégie peut créer un phénomène de cyclage (i.e. on peut revisiter des solutions déjà parcourues plusieurs fois). Afin de pallier à ce problème, la recherche tabou propose l'utilisation d'une mémoire permettant le stockage des dernières solutions rencontrées. [27]

Un schéma général de l'algorithme La recherche tabou est présenté dans l'algorithme 3.3

Algorithme 3.3. La recherche tabou
Début
Construire une solution initiale s ;
Calculer la fitness f(s) de s ;
Initialiser une liste tabou vide ;
s_{best}=s ;
Tant que le critère d'arrêt n'est pas vérifié **faire**
Trouver la meilleure solution s' dans le voisinage de s qui
ne soit pas tabou ou qui vérifie le critère d'aspiration ;
Calculer f(s') ;
Si fitness de (s') est meilleure que fitness de (s_{best}) **alors**
$s_{best} = s'$;
Fin Si
Mettre à jour la liste tabou ;
s= s' ;
Fin Tant que
Retourner s_{best}
Fin

Figure 3.5 Algorithme La recherche tabou[27]

V. 2.2 Le méta heuristique à base de population de solutions :

Les métaheuristiques à base de population de solutions débutent la recherche avec une panoplie de solutions. Elles s'appliquent sur un ensemble de solutions afin d'en extraire la meilleure (l'optimum global) qui représentera la solution du problème traité. L'idée d'utiliserun ensemble de solutions au lieu d'une seule solution renforce la diversité de la recherche et augmente la possibilité d'émergence de solutions de bonne qualité. Une grande variété de méthodes basées sur une population de solutions a été proposée dans la littérature,

commençant par les algorithmes évolutionnaires, passant par les algorithmes génétiques et arrivant aux algorithmes à base d'intelligence par essaims.

V. 3.2.1. Les algorithmes évolutionnaires

Les algorithmes évolutionnaires s'inspire de l'évolution naturelle des êtres vivants. Ils adoptent une sorte d'évolution artificielle pour améliorer la qualité des individus de la population. En fait, ils font évoluer itérativement une population d'individus. Ces derniers représentent des solutions du problème traité. Les qualités des individus sont mesurées à chaque itération du processus d'évolution. En fonction de leurs qualités, les meilleurs individus seront sélectionnés pour subir des combinaisons qui permettent la production d'une nouvelle population (dite: population d'enfants). Le processus d'évolution d'un algorithme évolutionnaire est basé sur trois opérations principales: la sélection, la reproduction et l'évaluation:

- **La Sélection**: Cette opération s'intervient dans deux phases. Elle est appliquée au premier lieu pour choisir les meilleurs individus parents qui vont se reproduire pour construire de nouveaux individus enfants. Ensuite, elle est appliquée à la fin de chaque itération pour opter pour les individus qui vont survivre et construire la nouvelle population.
- **La reproduction:** Cette opération est en général composée de deux autres opérations: le croisement et la mutation. Elle permet la génération de nouveaux individus en combinant (phase de croisement) les caractéristiques des individus sélectionnés puis en appliquant quelques modifications de certains individus (phase de mutation) pour améliorer leurs qualités.
- **L'évaluation:** Cette opération consiste à mesurer la qualité de chaque individu (calculer la fitness des individus).[22]

Les algorithmes évolutionnaires forment une classe principale de trois sous classes d'algorithmes (voir figure 3.3): les stratégies d'évolution, la programmation évolutionnaire et les algorithmes génétiques.

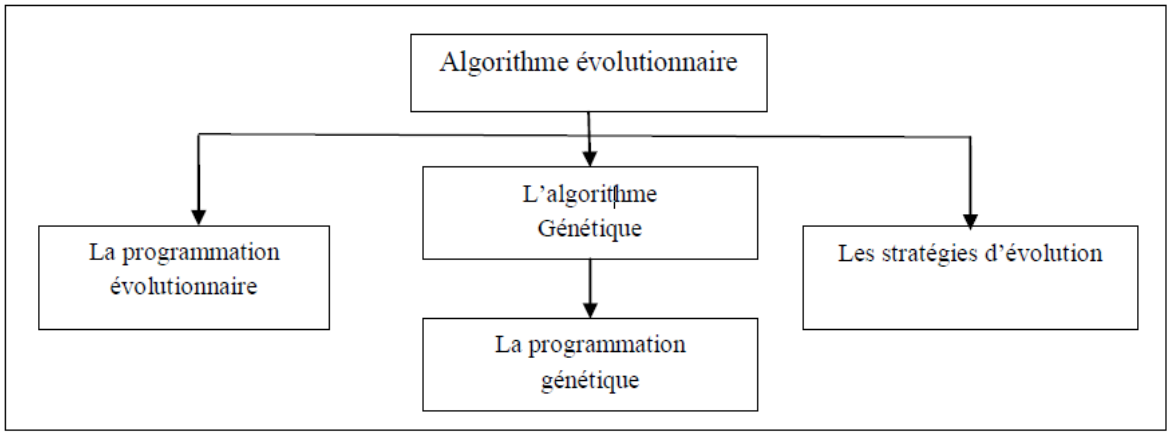

Figure 3.6 Les types des algorithmes évolutionnaires.[22]

- ✓ **Les stratégies d'évolution [**30**]:** Ont été conçues pour la résolution des problèmes d'optimisation continus.
- ✓ **La programmation évolutionnaire [33]** Les algorithmes de la classe de la programmation évolutionnaire sont conçus pour faire évoluer des structures d'automates à état fini.
- ✓ **L'algorithme génétique [**34**]:** Il sera bien détaillé dans la section suivante.

V.3.2.2. L'algorithme génétique

L'algorithme génétique représente une célèbre méta heuristique évolutionnaire[28] .il s'inspire des mécanismes biologiques tels que les lois de Mendel et la théorie de l'évolution .Il utilise le même vocabulaire que celui de la biologie et la génétique classique, on parle donc de: gène, chromosome, individu, population et génération.

✓ **Un gène:** est un ensemble de symboles représentant la valeur d'une variable. Dans la plupart des cas, un gène est représenté par un seul symbole (un bit, un entier, un réel ou un caractère).

✓ **Un chromosome**: est un ensemble de gènes, présentés dans un ordre donné de manière qui prend en considération les contraintes du problème à traiter. Par exemple, dans le problème du voyageur de commerce, la taille du chromosome est égale au nombre de villes à parcourir. Son contenu représente l'ordre de parcours de différentes villes. En outre, on doit veiller à ce qu'une ville (représentée par un nombre ou un caractère par exemple) ne doit pas figurer dans le chromosome plus qu'une seule fois.

✓ **Un individu**: est composé d'un ou de plusieurs chromosomes. Il représente une solution possible au problème traité.

✓ **Une population**: est représentée par un ensemble d'individus (i.e. l'ensemble des

solutions du problème).

✓ **Une génération: est une succession d'itérations composées d'un ensemble d'opérations permettant le passage d'une population à une autre.**

L'algorithme génétique fait évoluer une population composée d'un ensemble d'individus pendant un ensemble de génération jusqu'à ce qu'un critère d'arrêt soit vérifié. Le passage d'une population à une autre est réalisé grâce à des opérations d'évaluation, de sélection, de reproduction (croisement et mutation) et de remplacement.

Un schéma général de l'algorithme génétique est présenté dans l'algorithme 3.4

Algorithme 3.4 L'algorithme génétique
Début
Initialiser les paramètres nécessaires ;
Initialiser une population de N individus ;
Evaluer les N individus ;
Tant que la condition d'arrêt n'est pas satisfaite **faire**
Utiliser l'opérateur de sélection pour sélectionner K individus ;
Appliquer l'opérateur de croisement sur les K individus avec la probabilité P_c ;
Appliquer l'opérateur de mutation sur les individus avec la probabilité P_m ;
Utiliser l'opérateur d'évaluation pour évaluer les enfants obtenus ;
Utiliser l'opérateur de sélection pour remplacer quelques individus parents par des individus enfants ;
Fin Tant que
Retourner la ou les meilleures solutions ;
Fin

Figure 3.7 L'algorithme génétique[22]

Avantage :

- L'algorithme génétique permet une bonne combinaison entre l'exploitation de solutions et l'exploration de l'espace de recherche. Cela est établi en fonction des opérateurs de croisement et de mutation respectivement.

Inconvénients :

- un temps de calcul assez important pour pouvoir converger vers la solution optimale.
- le nombre de paramètres importants (taille de la population, paramètres de sélection, paramètres de croisement, paramètres de mutation, critère d'arrêt…).

V.3.2.3. L'intelligence par essaim

Les algorithmes basés sur l'intelligence par essaim forment une branche

d'algorithmes inspirés des phénomènes naturels. Ces algorithmes s'inspirent généralement des comportements collectifs de certaines espèces dans la résolution de leurs problèmes.

Le mot « essaim » est généralement utilisé pour désigner un ensemble fini de particules (l'oiseau, le poisson, la fourmille…etc.) ou d'agents interactifs. Les oiseaux évoluant en groupes, les bancs de poissons, les colonies de fourmis, les colonies d'abeilles..etc , Ainsi, en imitant le comportement social des particules formant des essaims capable de s'auto-organiser, plusieurs algorithmes ont été proposés ces dernières décennies comme: L'optimisation par essaim de particules, le système immunitaire artificiel, les colonies de fourmis artificielles, les colonies d'abeilles artificielles..etc

A. L'optimisation par essaim de particules (Particule swarm optimisation :PSO)

L'optimisation par essaim de particules est un méta heuristique populaire basé sur l'intelligence par essaim. Elle s'inspire du comportement social des oiseaux évoluant en groupe et des bancs de poissons. L'algorithme d'optimisation par essaim de particule lance la recherche, avec une population de solutions, où chacune est appelée « particule ». Cette dernière est caractérisée par une vitesse de déplacement et une position dans l'espace de recherche. Au cours du processus de la recherche, chaque particule se déplace pour modifier sa position dans l'espace de recherche en fonction de sa vitesse actuelle, sa position actuelle, sa meilleure position trouvée au cours des itérations passées et la meilleure position trouvée par l'essaim.[22]

B. L'algorithme de colonies de fourmis

L'idée de base de cet algorithme imite le comportement collectif des fourmis lors de leur déplacement entre la fourmilière et la source de nourriture. L'objectif du comportement collectif des fourmis est de collecter la nourriture sans perdre le chemin menant à leur nid. Les fourmis sont des insectes qui oeuvrent pour le bien du groupe. Leurs capacités physiques et cognitives limitées n'ont jamais construit un obstacle pour elles. Dans l'objectif de rechercher la nourriture en parcourant le plus court chemin, les fourmis se communiquent indirectement entre elles en provoquant des changements dans leur environnement. Au début de la recherche, les fourmis se propagent aléatoirement en prenant des chemins de différentes tailles (court, long,..) dont elles déposent sur le sol une matière odorante appelée « phéromone » d'intensités égales., elles déposent des phéromones un peu différents contenant un message concernant la qualité du site visité. Les fourmis ont tendance de suivre le chemin de plus forte intensité de phéromones comme le montre la figure 3.4.

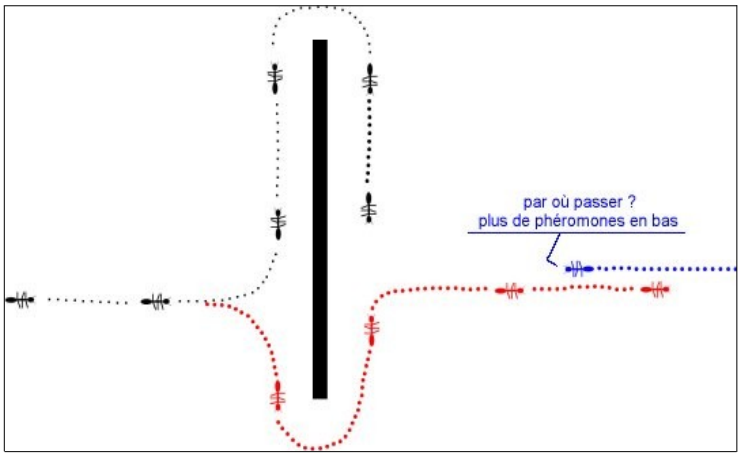

Figure3.8 la structure phéromone des fourmilles.

La fourmi informatique représente une solution au problème traité. Les phéromones informatiques sont des valeurs associées à des solutions trouvées. Ces valeurs dépondent des qualités des solutions. En fait, chaque fourmi (i.e. solution) dépose une certaine quantité de phéromones qui dépond de sa qualité.

L'algorithme3.5 de colonies de fourmis a été proposé pour la première fois pour résoudre le problème du voyageur de commerce [31] il se base sur trois phases essentielles:

- La construction du trajet de chaque fourmi.
- La distribution de phéromones sur le trajet de chaque fourmi.
- Evaporation des pistes de phéromones.

Algorithme 3.5. L'algorithme de colonies de fourmis pour le TSP
Début
Initialiser une population de m fourmis ;
Evaluer les m fourmis ;
Tant que la condition d'arrêt n'est pas satisfaite **faire**
Pour i=1 à m **faire**
Construire le trajet de la fourmi i;
Déposer des phéromones sur le trajet de la fourmi i;
Fin pour
Evaluer les m fourmis;
Evaporer les pistes de phéromones;
Fin Tant que
Retourner la ou les meilleures solutions ;
Fin

Figure3.9 L'algorithme de colonies de fourmis pour le TSP[31]

C. L'optimisation par colonies d'abeilles

L'algorithme ABC s'inspire du modèle naturel du comportement des abeilles mellifères lors de la recherche de leur nourriture. Le processus de recherche de nourriture chez les abeilles est fondé sur un mécanisme de déplacement très efficace. Il leur permet d'attirer l'attention d'autres abeilles de la colonie aux sources alimentaires trouvées dans le but de collecter des ressources diverses. En fait, les abeilles utilisent un ensemble de danses frétillantes comme moyen de communication entres elles. Ces danses permettent aux abeilles de partager des informations sur la direction, la distance et la quantité du nectar avec ses congénères. La collaboration et la connaissance collective des abeilles de la même colonie sont basées sur l'échange d'information sur la quantité du nectar dans la source de nourriture trouvée par les différents membres. Des études sur le comportement de danses frétillantes des abeilles ont montré [32]

- La direction des abeilles indique la direction de la source de nourriture par rapport au soleil.
- L'intensité de la danse indique la distance de la source de nourriture.
- La durée de la danse indique la quantité du nectar dans la source de nourriture trouvée.

Dans un algorithme d'optimisation par colonies d'abeilles[36], une source de nectar correspond à une solution possible au problème à traiter. La colonie d'abeilles artificielle est composée de trois types d'abeilles: les ouvrières, les spectatrices et les scoutes.

- **L'ouvrière** : exploite la source de nourriture trouvée. Elle se base sur sa mémoire et

essaye d'apporter des modifications à sa position (solution) actuelle pour découvrir une nouvelle position (i.e. source de nourriture).

- **L'abeille spectatrice** : attend le retour des ouvrières au champ de danse pour observer leurs danses et recueillir des informations sur les sources de nectar qu'elles ont trouvé.
- **L'abeille scoute** : exploite l'espace de recherche en lançant une recherche aléatoire d'une nouvelle source de nourriture.

Une abeille ouvrière est assignée à chaque source de nourriture. La taille de la population de la colonie est égale au nombre des abeilles ouvrières et la quantité du nectar dans une source de nourriture correspond à la qualité (fitness) de la solution proposée. L'algorithme sera présenté dans le chapitre suivant.

Toutes ces étapes sont résumées dans le pseudo Algorithme3.6 de la Figure 3.10.

Entrée : S, W, O

S : nombre de butineuse éclaireuse.

W : nombre de butineuse active.

O : nombre de butineuse inactive

Sortie : la meilleure solution

algorithme 3.6 de l'optimisation par colonie d'abeille

begin

Initialiser la population avec S+W solutions aléatoires

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

end

Figure 3.10: algorithme de l'optimisation par colonie d'abeille[32]

VI. Conclusion

Dans ce chapitre nous avons essayé de présenter un état de l'art sur les méthodes de résolution de différents problèmes d'optimisation présentées dans la littérature commençant par les méthodes exactes aux méthodes approchées.

Nous avons constaté que les méthodes exactes permettent d'aboutir à la solution optimale, mais elles sont trop gourmandes en termes de temps de calcul et d'espace mémoire

requis. Cependant, les méthodes approchées demandent des coûts de recherche raisonnables. Mais, elles ne garantissent pas l'optimalité de la solution.

Nous avons pu constater que les méthodes approchées peuvent être partagées en deux classes: des méthodes heuristiques et des méthodes méta heuristiques. Une méthode heuristique est applicable sur un problème donné. Tandis qu'une méthode méta heuristique est plus générique et elle peut être appliquée sur une panoplie de problèmes d'optimisation. En outre, nous avons constaté que les méthodes méta heuristiques peuvent être partagées en deux sous classes: des méthodes à base d'une solution unique et des méthodes à base de population de solutions.

Nous avons essayé de présenter le principe de plusieurs méthodes méta heuristiques y compris les méthodes basées sur l'intelligence par essaim qui ont construit une tendance très active ces dernières décennies. Nous nous focalisons sur l'algorithme d'essaim d'abeille que fera l'objet de notre solution proposée, et qui sera présenté dans le chapitre suivant.

Chapitre 4

Solution proposée

I. Introduction

Après avoir présenté dans les chapitres précèdent des généralités sur l'extraction d'itemset fréquents à partir des bases de données évidentielles ainsi que les différentes méthodes d'optimisation pour la résolution des problèmes complexe. Nous allons dans ce chapitre décrire notre solution qui est basée sur les métas heuristiques en s'inspirant de l'algorithme général d'optimisation par colonie d'abeilles (BSO).

II. Extraction des itemsets fréquents evidentiels basée sur l'optimisation par colonie d'abeilles (Bee Swarm Optimization, BSO)

Nous avons adapté l'algorithme (cooperative bees swarm optimization) proposé par et Drias et al [29] à l'extraction des itemsets fréquents à partir de données évidentielles. Notre algorithme est appelé UFIM-BSO est présenté comme suit :

UFIM-BSO Algorithm

***Entrée**: Base de données evidentielles ,Minsup ,nombre itération, nombre abeille ;*
***Sortie:** Ensemble des itemsets fréquents*

Begin

```
1.  Ff=-1
2.   While(Ff==-1)
3.     begin
4.     Sref = la solution initiale générée aléatoirement depuis la base de données évidentielles ;
5.     Ff=Fitness(Sref) ;
6.     end
7.  End While
8.  While(nombre iteration>0)
9.    begin
10.  Tabou=sref ;
11.  Déterminer l'espace de recherche voisinage de Sref ;
12.  Affecter (abeille,solution) ;
13.  for pour chaque abeille n do
14.    begin
15.     Déterminer l'espace de recherche voisinnage de  la solution affectée à elle ;
16.     Table dance=la meilleur solution trouvée ;
17.    end
18.  End for
19.  if ( tabou< la meilleure solution de table dance)
20.     tabou= la meilleure solution de table dance ;
21.  else
22.    While(( Ff=Fitness(Sref))==-1);
23.  end if
24. end
```

25. endWhile
26. End

Dans ce qui suit nous allons d'écrire l'architecture de notre système et les différentes étapes menées pour développer notre solution.

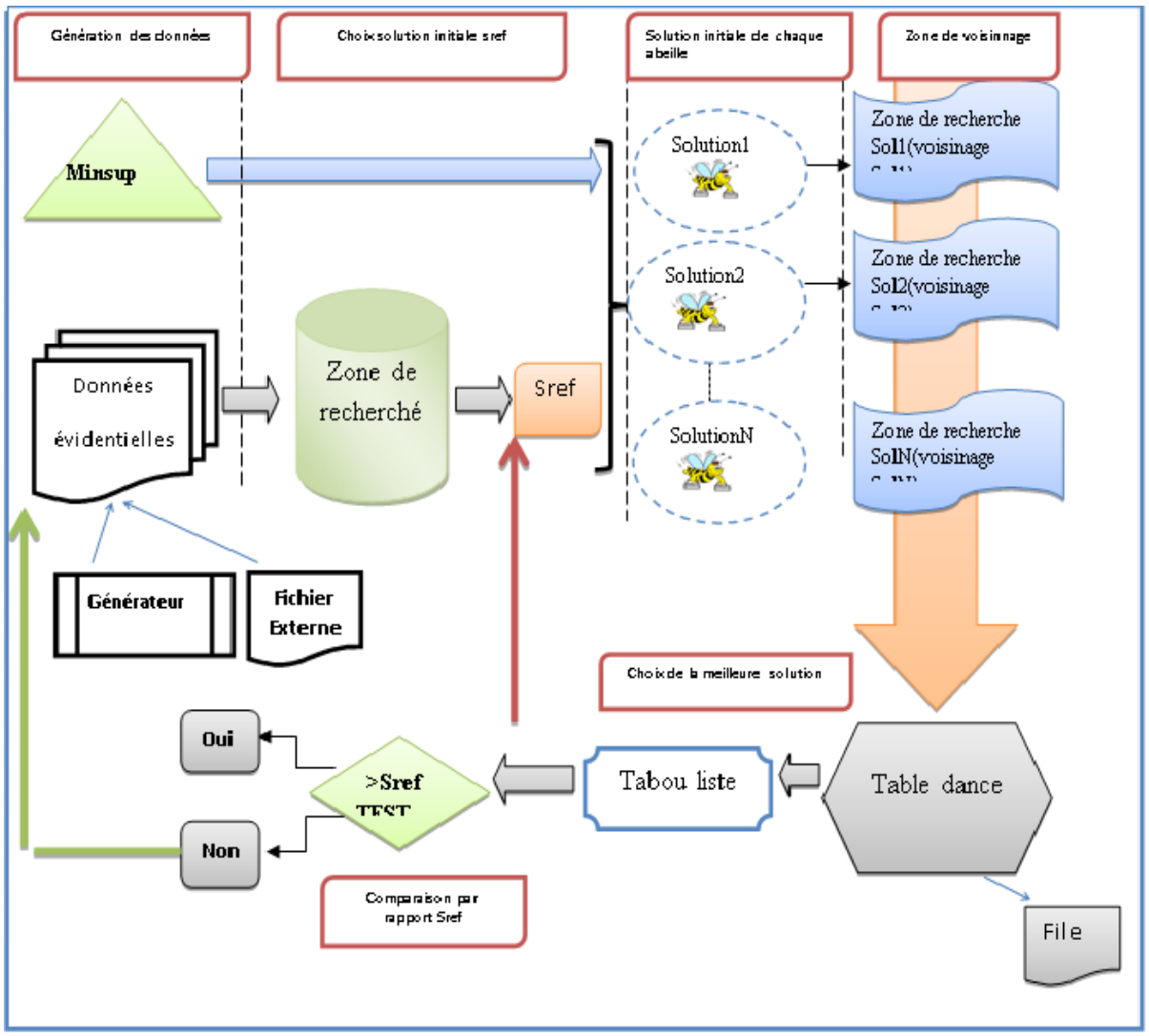

Figure4.1 Schéma général UFIM_ BSO

Le schéma suivant présentes les différentes étapes de notre solution BSO-UFIM au problème d'extraction d'itemsets fréquents .

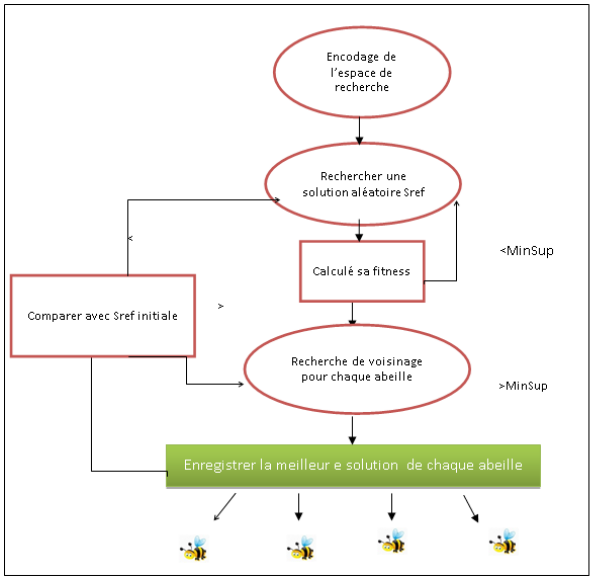

Figure4.2 Les étapes principale de la solution BSO UFIM

Notre système est devisé en deux composants principaux : le générateur et l'extracteur d'itemsets fréquents.

II.1 Principe de fonctionnement de générateur de données evidentielles

Le générateur permet de produire des données évidentielles selon la théorie de Dampster Shaffer, les données seront de forme transaction /objet (itemsets). De plus, il permet de générer plusieurs types de base de données, à savoir :

- Base de données certaines (données parfaites).
- Base de données probabilistes.
- Base de données possibilistes (données imprécises).
- Base de données qui contient des données manquantes.
- Base de données Evidentielles.

Tableau. 4.1 – Exemple d'une base de données évidentielles.

	A	**B**
T01	A1{0.5}, A2{0.5}, Θ{0.0}	B1{1}
T02	A1{0.2}, (A1,A3){0.1}, Θ{0.7}	B2{0.1} , Θ{0.9}
T03	A1{0.1}, (A2,A3) {0.4} , Θ{0.5}	B3{0.5} , Θ{0.5}
T04	A1{0.8} , Θ{0.2}	B4 {1}

Grace à la théorie de Dumpster Shafer, plusieurs types d'imperfection peuvent être modélisés. En effet, le tableau 4.1 illustre un exemple pour deux objets A et B et quatre transactions, qui présentent les différents types d'imperfections. Selon Bach tobji et al [21] les données imparfaites peuvent être modélisées par des données évidentielles comme suit :

- **Parfaite** : la valeur de la colonne B dans les lignes 1 et 4.
- **Probabilistes** : la valeur de la colonne A dans la première ligne est probabilistes et mêmes chose pour la valeur de l'attribut B dans la 3 ^{iem} ligne.
- **Possibiliste** : lorsque le corps d'évidence inclut des éléments focaux imbriqués, la fonction π en théorie de possibilité correspond à la fonction ***pl*** en théorie d'évidence. cas valeur de l'attribut A dans la 2 ^{ieme} ligne.
- **Manquante** : si une valeur d'un attribut i pour la ligne j est manquante.
- **Evidentielle** : A l'instar de l'attribut A dans la 3 ieme ligne qui est ni parfaite, ni probabiliste, ni possibiliste, ni manquante.

II.2 l'extracteur d'itemsets evidentiels fréquents

Notre solution consiste à adapter l'algorithme **4.1 (Colonies d'abeille)** décrit si dessus pour extraire des itemsets fréquents à partir de données évidentielles.

Cette solution proposée passe par plusieurs étapes :

➔ ***1^{ier} étapes :*** Considérons qu'on a une base des données évidentielles et la valeur Minsup donnée par l'utilisateur.

-Tout d'abord on commence par chercher une solution aléatoire Sref fréquente qui vérifie la fonction objective *Fitness par rapport au* ***Minsup.***

- *Case of Expected-support*

Begin

if (*ExpSupport >MinSup)*

Ff=ExpSupport (*MinSup*)

else *Ff*= -1

End

- *Case of Evidential-support*

Begin

if (*Evidential support) > MinSup)*

Ff = Evidentiel-Support(MinSup)

else *Ff*= -1

End

a) Calcul du support

Pour calculer le frequentness de Sref il existe plusieurs types de support. Dans notre solution on a opté pour :

a-1 Evidentielle support

une nouvelle approche pour le support itemset informatique et appliqués sur un Fréquent Itemset Maintenance (*FIM*) problème[21]. Toutes les méthodes [21] ont été basés sur le produit cartésien entre *BBAs*. Le support d'un itemset$X = \prod i \in [1 \ldots . n]$ tel que xi est un item évidentiel appartenant au cadre de discernementθi. Étant donné que les items ne partagent pas le même cadre de discernement, toute règle de fusion ne peut pas être appliquée. Dans ce qui suit, nous étudions le support de croyance calculé par l'équation suivante :

$$mj(X) = \prod_{x_i \in X} mij(x_i)[10]$$

Où $Mj(X)$est le produit cartésien de tous *BBA* dans la transactionTj. Ainsi, le *BBA* de l'itemset X exprimé dans l'ensemble *EDB* devient :

$$m_{EDB}(X) = \frac{1}{d} \sum_{j=1}^{d} mj(X)[10]$$

Ensuite, le support de X dans la base de données *EDB*devient :

$$Support_{EDB}(X) = Bel_{EDB}(X)[10]$$

➔ **2iem Etapes :**

a- L'encodage de la solution Sref :

On peut citer deux célèbres représentations, à savoir le codage binaire et le codage entier. En codage binaire, chaque solution (itemset) est représenté par un vecteur S de ***n***

éléments où ***n*** est le nombre d'articles. De plus, S [i] = 1 si l'élément i est dans la solution (intemset) et 0 sinon.

Exemple : Soit Sref la solution suivante :{A1A3,B2,C4}, La représentation binaire de Sref sous forme d'un vecteur et la suivante :

Tableau4.2 Exemple d'une solution initiale Sref

A1	A2	A3	B1	B2	B3	C1	C2	C3	C4
1	0	1	0	1	0	0	0	0	1
Objet A			Objet B			Objet C			

La taille du vecteur est la somme de tous les items des objets qui existe dans la base de données évidentielle, Sref=3+3+4=10.

b-Recherche de voisinage (neighborhood) : La recherche de voisinage est calculée à partir de la solution de référence Sref et du paramètre Flip. Notre solution est inspirée de la méthode de [Y. Djenouri, H. Drias][40] en ajoutant la valeur de Flip=1 a Sref pour chaque bit. chaque solution s trouvée est un vecteur de m (m est le nombre d'éléments) et représente une solution initiale a attribuée à une abeille i(le nombre d'abeille est donnée par l'utilisateur).

Exemple : selon la valeur de Sref donnée précédemment : les solutions initiales sont les suivantes :

- Le nombre de solution est égale au nombre d'abeilles.
- La valeur du Flip=1 ; exemple nombre d'abeilles=4

Tableau4.3 Exemple de détermination de l'espace de recherche selon Sref.

	A 1	A 2	A 3	B 1	B 2	B 3	C 1	C 2	C 3	C 4
Sref initiale	1	0	1	0	1	0	0	0	0	1
Sref 0	0	0	1	0	1	0	0	0	0	1
Sref 1	1	1	1	0	1	0	0	0	0	1
Sref 2	1	0	0	0	1	0	0	0	0	1
Sref 3	1	0	1	1	1	0	0	0	0	1

➔ **3 iem étape** : Cette étape

Consiste à affecter aléatoirement à chaque abeille une solution parmi les solution générer dans l'étape precedente.chaque abeille va refaire le même travail cité dans l'étape 2 pour trouver son espace de recherche (voisinage).

Exemple : abeille0 va avoir la solution Sref0 :

- Le nombre de solution est égale à la taille de Sref.
- **Flip=1, nombre de solution=10**

Tableau 4.4. Exemple de recherche de solution voisinage Sref pour abeille(i)

	A1	A2	A3	B1	B2	B3	C1	C2	C3	C4
Sref initiale Bee0	1	1	1	0	1	0	0	0	0	1
Sref 0	0	1	1	0	1	0	0	0	0	1
Sref 1	1	0	1	0	1	0	0	0	0	1
Sref 2	1	1	0	0	1	0	0	0	0	1
Sref 3	1	1	1	1	1	0	0	0	0	1
Sref4	1	1	1	0	0	0	0	0	0	1
Sref5	1	1	1	0	1	1	0	0	0	1
Sref6	1	1	1	0	1	0	1	0	0	1
Sref7	1	1	1	0	1	0	0	1	0	1
Sref8	1	1	1	0	1	0	0	0	1	1
Sref9	1	1	1	0	1	0	0	0	0	0

➔ **4^{iem} étape** : Apres avoir trouvé son espace de recherche, l'abeille i va calculer la Fitness de chaque solution obtenue, la meilleur solution sera sauvegarder dans Tableau de dance ainsi que son support.il contiendra toute les meilleures solutions obtenu de Sref.

➔ **5 ^{iem} étape** : Enregistrer la meilleur solution du tableau dance de chaque itération dans Tabou List .Si cette dernière est meilleur que Sref alors Sref prend la valeur de la solution Tabou liste.

➔ **6 ^{iem} étape** : refaire les mêmes étapes pour chaque itération.

a-2 L'Expected Support

Expected support d'un itemest X dans la base de données incertaine est la somme de produit de probabilité P (X, t j) de X dans la transaction t j sur tous les n transactions dans la base de données. [16]

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

$$expSup(X)=\sum_{j=1}^{n} P(X,tj)=\sum_{j=1}^{n}\left[\prod_{x\in X} P(x,tj)\right] \quad [16]$$

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

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

$$\text{ExpSup}(X) >= \text{minsup}. \quad [16]$$

Notre solution de recherche des itemsets fréquents à partir de base de données probabiliste repose sur les mêmes étapes de l'algorithme 4.1 *BSO-UFIM* sauf que nous avons proposées une méthode de calcul d'espace de recherche de voisinage etape2(b). Combinée entre celle basée sur Sref et celle qui parcours tout l'espace de voisinage. on attribut a chaque abeille une solution Sref initiale et à partir de cette dernière en va parcourir une zone de recherche défini comme suit :

-Exemple : nombre d'abeilles=4

Zone de recherche abeille0.

Tableau 4.5 Exemple de notre solution proposée pour la recherche de voisinage d'abeille(i).

	A1	A2	A3	B1	B2	B3	C1	C2	C3	C4
Sref initiale Bee0	1	0	0	0	1	0	0	0	0	1
Sref 0	0	1	0	0	1	0	0	0	0	1
Sref 1	0	0	1	0	1	0	0	0	0	1
Sref 2	1	0	0	1	0	0	0	0	0	1
Sref 3	1	0	0	0	1	0	0	0	0	1
Sref4	1	0	0	0	0	1	0	0	0	1
Sref5	1	0	0	0	1	0	1	0	0	0
Sref6	1	0	0	0	1	1	0	1	0	0
Sref7	1	0	0	0	1	0	0	0	1	0
Sref8	1	0	0	0	1	0	0	0	0	1

III. Conclusion

Dans ce chapitre nous avons présenté l'algorithme de colonie d'abeille et son principe de fonctionnement, aussi nous avons expliqués en détaille notre solution d'extraction d'itemsets fréquents basée sur des données évidentielles ainsi que EIF basée sur des données probabilistes.

Dans le chapitre suivant on définit les outils et les logiciels de programmation utilisée pour implémenter et tester notre algorithme et démontrer son efficacité.

Chapitre 5

Test et validation

I. 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.

II. Présentation de l'environnement de travail

1. Systèmes d'exploitation

Nous utilisons le système Windows 7 son type est. 64-bit, sa RAM et de 8GO, son Disque dure et de 500 GO Le processeur est Intel(R) Core(TM) i3 CPU M 330 @ 2.20GHz 2.20 GHz

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]

II.2 Présentation des données utilisées pour les tests

Les fichiers de donnée que nous avons utilisée sont des fichiers qui contient des itemsets , produit par notre générateur ou par des fichiers importé.

La figure 5.1 illustre un exemple de fichier de données évidentielles produites par notre générateur. Chaque ligne représente une transaction. Toutes les transactions ont même nombre d'objets.

Exemple : la transaction T10 contient 4 attributs et chaque attribut est

composé d'items.

Les items de l'objet 1 sont les suivants :{A2}{A1A3}(0.21) données évidentielles

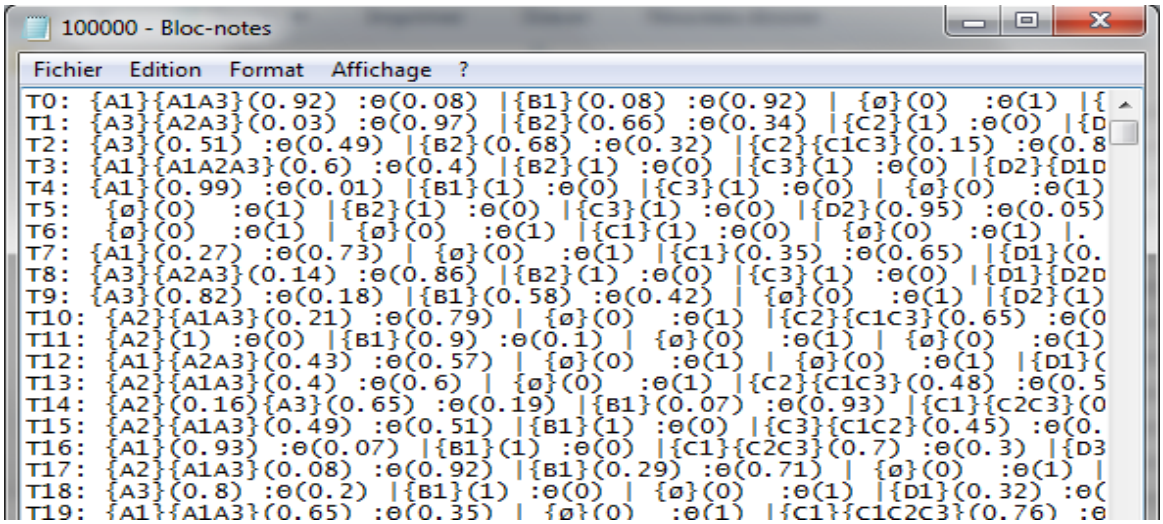

Figure 5.1 Fichier de base de données évidentielles

III. Présentation de l'interface utilisateur

Dans qui suit nous allons présenter tous les composants de interfaces utilisateur de notre application.

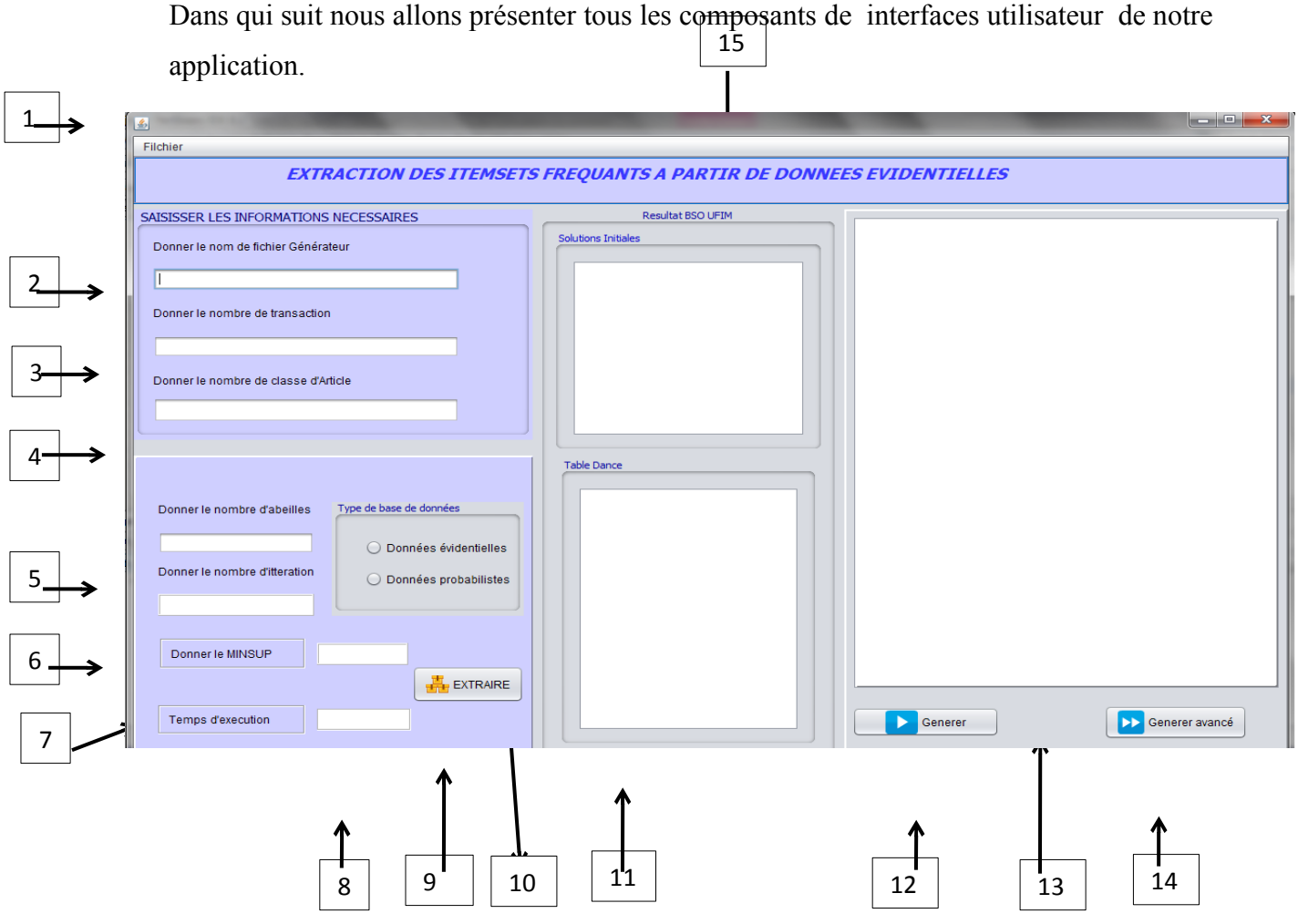

Figure5.2 Interface de l'application

1. Menu fichier pour importe le fichier des itemsets évidentielles.
2. Saisir le nom de fichier a générer.
3. Saisir le nombre de transaction
4. Saisir le nombre d'item des objets
5. Définir le nombre d'abeilles
6. Saisir le nombre d'itteration
7. Définir le Minsup
8. Détermine le temps d'execution en seconde
9. Lance la fonction d'extraction
10. Choisir extraction avec une base de données probabiliste ou évidentielle.
11. Afficher les solution trouvées de chaque abeilles
12. Produire un fichier des itemset evidentielles.
13. Afficher le fichier des itemsets evidentielles.
14. Produire des données soit probabiliste soit evidentielle avec des pourcentages des types d'itemsets.
15. afficher les meilleures solutions de l'itération. (table dance).

La figure 5.3 présente les résultats obtenus après extraction l'espace d'affichage 15 représente les solutions obtenues par chaque abeille qui vérifie la fonction objective.

Dans l'espace d'affichage 11 on présente les meilleures solutions obtenues par chaque abeille dans chaque itération.

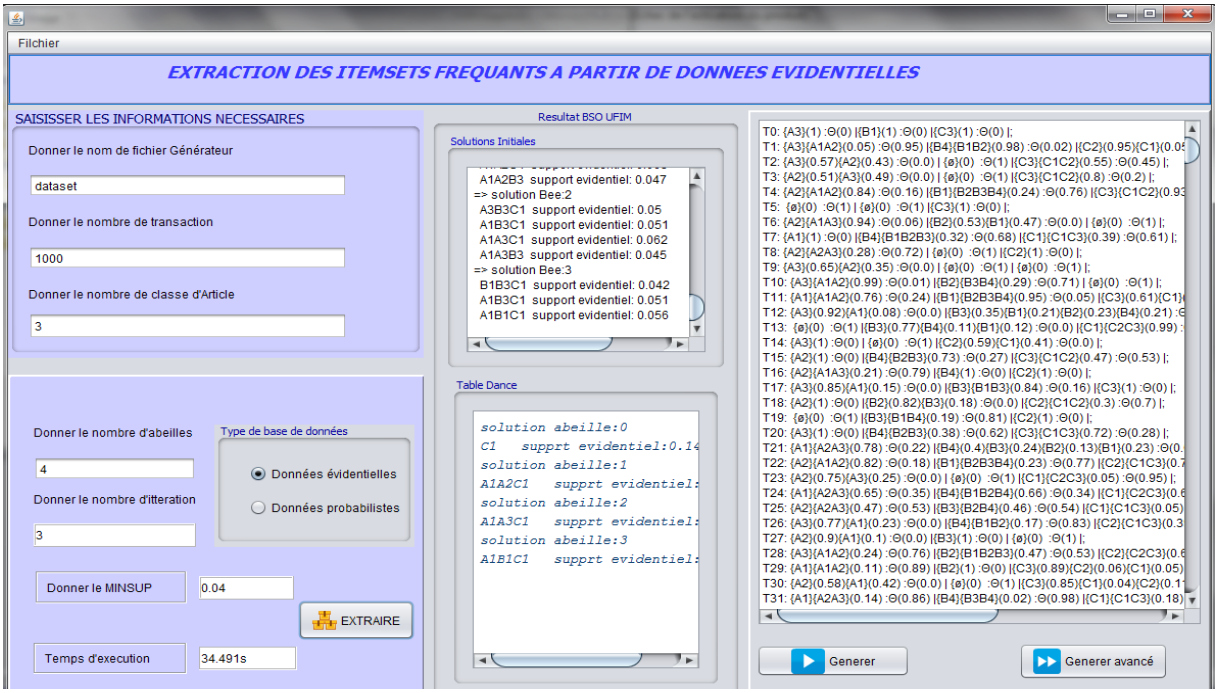

Figure5.3 Résultat après extraction de données évidentielles

Le résultat final des meilleures solutions et sauvegardée dans un fichier illustré dans la figure 5.4.

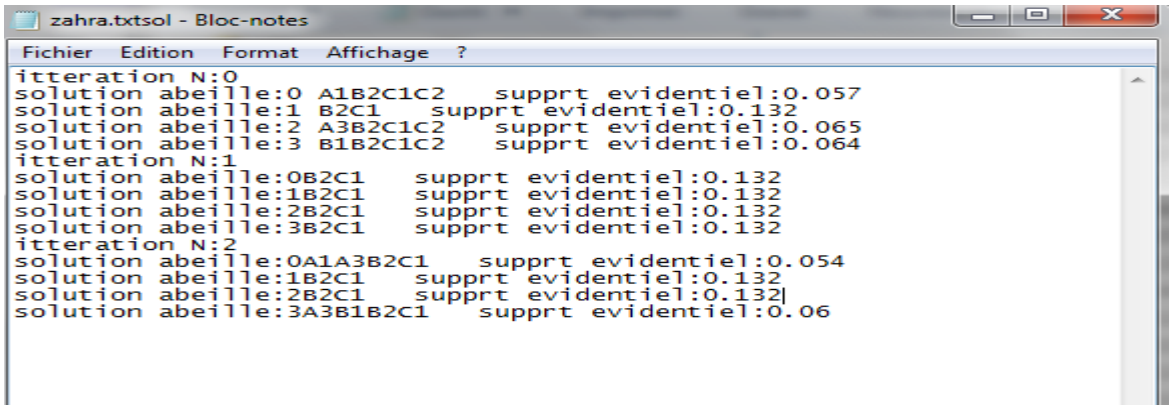

Figure 5.4 fichier de sauvegarde des meilleures solutions

IV. Test et validation

Nous avons validé notre algorithme avec tests sur des fichier produit par le générateur ou données importés.

IV.1.Test de performances :

A. Base de données évidentielles.

- **Première expérimentation**

Dans ce premier test nous allons changer à chaque fois le MinSup et observer le temps de recherche sur la même base de données evidentielle .avec 1000 transaction,3 attribut,2 abeilles et 2 itérations.

Tableau 5.1 résultat du test changement MinSup

Valeur minsup	Temps d'execution
0.02	30.05sec
0.04	32.12sec
0.06	38.45sec
0.08	63.60sec
0.10	139.16sec

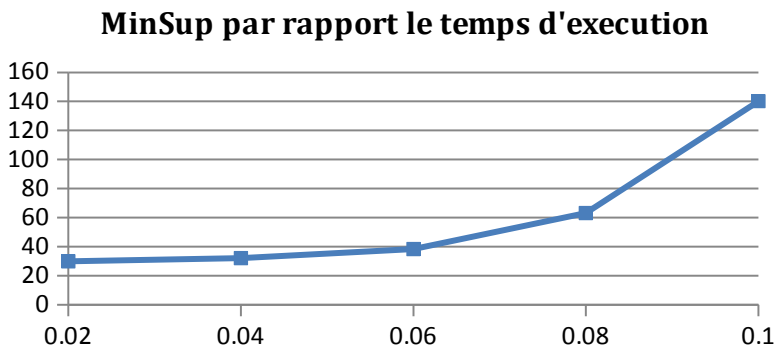

Figure 5.5 Temps d'execution en fonction de MinSup

Depuis la figure 5.5, on a remarqué qu'à chaque fois qu'on augmente le MinSup le temps de recherche augmente surtout à partir de la valeur 0.06 de MinSup.

- **Deuxième expérimentation**

Dans cette expérimentation nous allons jouer sur la taille de DtatSet en fixant MinSup = 0.4.nombre d'iteration=2, nombre d'objet =3

Tableau5.2 taille DtaSetpar rapport le temps d'exécution

Taille DataSet	Temps d'exécution
1000	9.34sec
2000	18.43sec
3000	29.43sec
4000	36.05sec
5000	54.62sec

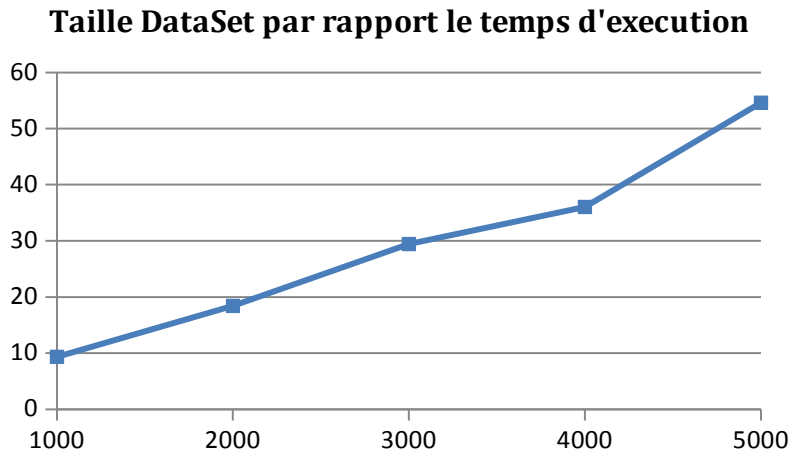

Figure5.6: Temps d'exécution en fonction de la taille de DataSet

Dans la deuxième expérience **figure5.6** on a remarqué que le temps d'exécution vari linéairement avec l'augmentation de la taille de DataSet.

- **Troisième expérimentation**

Dans cette expérimentation nous avons allons varier le nombre d'objet et observe le temps de réponse sachant que chaque objet contient 2items

Tableau 5.3 Test3(Temps d'exécution en fonction de Nombre d'objet)

Nombre d'objet	Temps de reponse
2	**4.55**
3	**6.64**
4	**9.56**
5	**13.08**
6	**15.07**
7	**18.02**

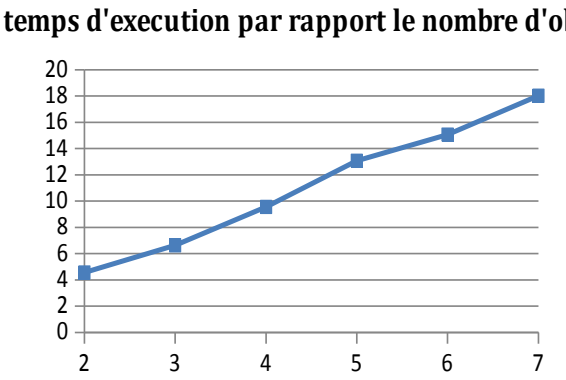

Figure 5.7: Temps d'exécution en fonction de Nombre d'objet

Dans le graphe de la figure 5.7 (les résultats de test 1), nous remarquons que, à chaque fois que nous augmentons la valeur de nombre d'objet le temps d'exécution augmente aussi .

- **Quatrième expérimentation :**

Dans ce dernier test nous allons observer le temps de réponse en fonction de la taille des solutions trouvées. On suppose qu'on a 4 objets ,2 abeille et MinSUP =0.04

Tableau 5.4 Test 4 (temps de réponse en fonction du nombre des items)

Nombre d'item	Temps de reponse
1,2,3=6	6.75 sec
2,3,3=8	9.38 sec
2,4,3=9	11 sec
3,4,3=10	12.09 sec
3,5,3=11	13.82 sec
3,6,4=13	17.78 sec

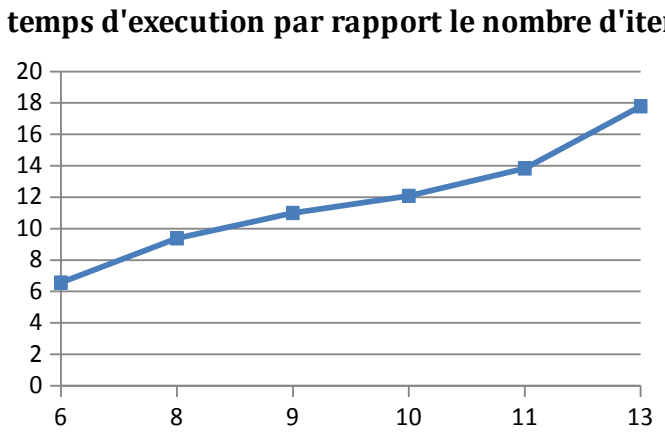

Figure 5.8 Temps d'execution en fonction de nombre d'item

Dans le graphe de la figure 5.8 (les résultats de test 4), nous remarquant qu' à chaque fois que nous augmentons la valeur de nombre d'item le temps de réponse augmente aussi d'une façon presque linéaire.

B -Base de données probabiliste :

- **Première expérimentation**

Dans ce premier test nous allons changer à chaque fois le MinSup et observé le temps de recherche sur la même base de données evidentielle .avec 1000 transaction,3 attribut,2 abeilles et 2 itérations.

Tableau 5.5 résultat du test changement MinSup

Valeur minsup	Temps d'execution
0.02	48.150sec
0.04	45.601sec
0.06	44.661sec
0.08	44.895sec
0.10	45.112sec

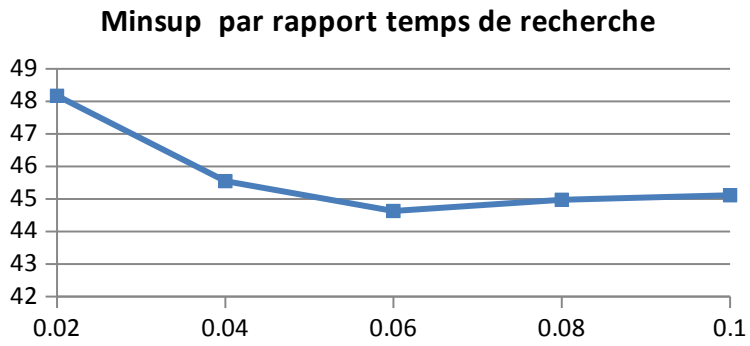

Figure 5.9 Temps d'execution en fonction de MinSup

Depuis la figure 5.9, on a remarqué qu'à chaque fois qu'on augmente le MinSup le temps de recherche diminu.

- **Deuxième expérimentation**

Dans cette expérimentation nous allons jouer sur la taille de DtatSet en fixant MinSup = 0.4.nombre d'iteration=2, nombre d'objet =3

Tableau5.6 taille DtaSetpar rapport le temps d'exécution

Taille DataSet	Temps d'exécution
1000	43.376
2000	89.621
3000	128.865
4000	183.073
5000	217.708

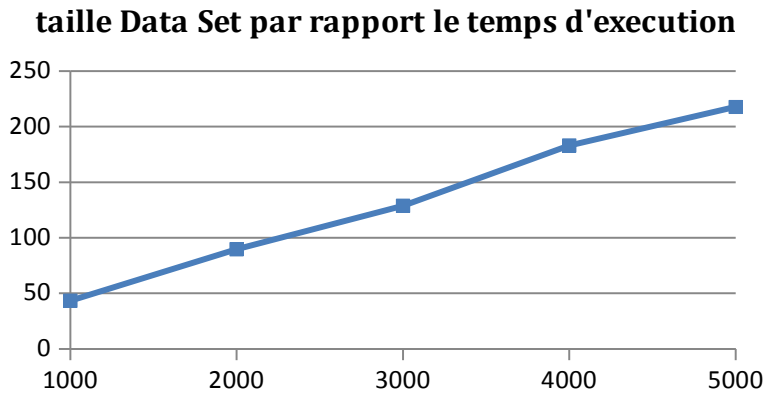

Figure5.10: Temps d'exécution en fonction de la taille de DataSet

Dans la deuxième expérience **figure5.10** on a remarqué que le temps d'exécution vari linéairement avec l'augmentation de la taille de DataSet.

- **Troisième expérimentation**

Dans cette expérimentation nous avons allons varier le nombre d'objet et observe le temps de réponse sachant que chaque objet contient 2items

Tableau 5.7 Test3(Temps d'exécution en fonction de Nombre d'objet)

Nombre d'objet	**Temps de reponse**
2	**30.515sec**
3	**41.262sec**

4	**63.106sec**
5	**99.956sec**
6	**187.7sec**

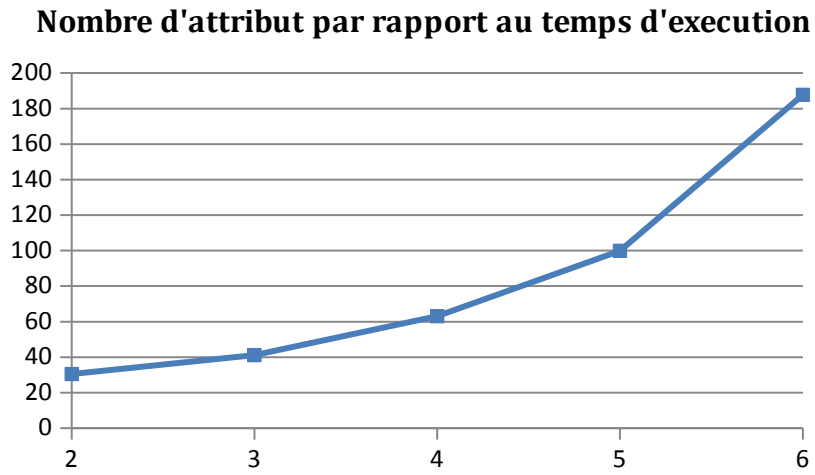

Figure 5.11: Temps d'exécution en fonction de Nombre d'objet

Dans le graphe de la figure 5.11 nous remarquons que, à chaque fois que nous augmentons la valeur de nombre d'objet le temps d'exécution augmente .

- **Quatrième expérimentation :**

Dans ce dernier test nous allons observer le temps de réponse en fonction de la taille des solutions trouvées. On suppose qu'on a 4 objets ,2 abeille et MinSUP =0.04

Tableau 5.8 Test 4 (temps de réponse en fonction du nombre des items)

Nombre d'item	Temps de réponse
1,2,3=6	45.301sec
2,3,3=8	44.741sec
2,4,3=9	46.900sec
3,4,3=10	47.160sec
3,5,3=11	50.365sec
3,6,4=13	54.544sec

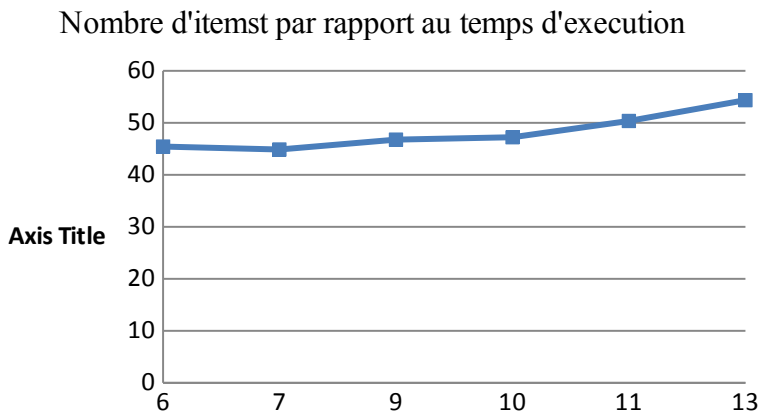

Figure 5.12 Temps d'execution en fonction de nombre d'item

Dans le graphe de la figure 5.12 ,nous remarquant que' à chaque fois que nous augmentons la valeur de nombre d'item le temps de réponse reste considérablement constant.

V. 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.

Nous avons présenté des tests de performance sur notre solution pour découvrir la relation entre temps d'exécution et les différents critère nécessaire a l'extraction d'itemsets fréquents.

Conclusion générale

I. Conclusions

Grâce aux techniques d'extraction de motifs fréquents, les bases de données volumineuses sont devenues des sources riches et fiables pour la génération et la validation de connaissances. de ce fait plusieurs méthodes de résolution de différents problèmes d'optimisation été présentées dans la littérature commençant par les méthodes exactes aux méthodes approchées.

Nous avons constaté que les méthodes exactes permettent d'aboutir à la solution optimale, mais elles sont trop gourmandes en termes de temps de calcul et d'espace mémoire requis. Cependant, les méthodes approchées demandent des coûts de recherche raisonnables. Mais, elles ne garantissent pas l'optimalité de la solution. C'est pour ces raisons que nous avons panachés dans notre projet vers les méthodes méta heuristique pour la résolution des problèmes d'extraction d'itemsets fréquents évidentes, plus précisément les algorithmes d'optimisation par colonie d'abeilles. .

Et afin de comprendre et cerner la problématique de EIF à partir de données évidentielles nous avons étudié dans le 1ier chapitre l'incertitude de données, et présenté les différents algorithmes dEIF a partir des données incertaines.et puisque on s'intéresse plus au données évidentielles le chapitres 2 a été consacré a la présentation de la théorie de Dempster Staffer [21]. Dans le chapitre 3 nous avons présentées un état de l'art des méthodes de résolution des problèmes complexes. Le chapitre 4 et 5 présente notre solution et les outils nécessaires a sa réalisation.

II. Limites et perspectives

- Evaluer la pertinence des itemset résultats obtenus en utilisant des données réelles
- Evaluer la qualité des résultats obtenus par rapport aux méthodes exactes afin d'augmenter le nombre itemsets évidentiels produit par notre méthode.
- Vu que la taille des bases de données augmente de façon considérable, ce qui nous mette dans le cadre de Big Data.Une version parallèle distribuée de notre méthode est nécessaire pour le passage à l'échelle.
- L'utilisation des mesures de fréquentness plus fiable ou plus précis tel que le support evidentiel précis et le support probabiliste, peut améliorer la qualité des itemsets evidentiels générés par notre méthode.

BIBLIOGRAPHIE

---

[1] Chui, C.-K., Kao, B., Hung, E.: Mining frequent itemsets from uncertain data. In:Proceedings of the PAKDD 2007, pp. 47–58. Springer (2007)

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

[3]. Fréquent itemset par Tanagra le 3.10.11, Octobre 2011.

[4]. Green, T., Tannen, V.: Models for incomplete and probabilistic information. Bull. Tech. Committee Data Eng. 29(1), 17–24 (2006).

[5]. Leung, C.K.-S., Brajczuk, D.A.: uCFS2: an enhanced system that mines uncertain data for constrained frequent sets. In: Proceedings of the IDEAS 2010, pp. 32–37. ACM (2010)

[6]. Leung, C.K.-S., Brajczuk, D.A.: Efficient algorithms for the mining of constrained frequent patterns from uncertain data. ACM SIGKDD Explor. 11(2), 123–130 (2009)

[7]. Leung, C.K.-S., Hao, B., Brajczuk, D.A.: Mining uncertain data for frequent itemsets that satisfy aggregate constraints. In: Proceedings of the ACM SAC 2010, pp. 1034–1038 (2010)

[8]. Madden, S.: From databases to big data. IEEE Internet Comput. 16(3), 4–6 (2012)

[9] . Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. J. Comput. Sci. Technol. 15, 487–499 (1994)

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

[11]Leung, C.K.-S., Mateo, M.A.F., Brajczuk, D.A.: A tree-based approach for frequent pattern mining from uncertain data. In: Washio, T., et al. (eds.) PAKDD 2008. LNAI, vol. 5012, pp. 653–661 (2008)

[12] Leung, C.K.-S., Tanbeer, S.K.: Fast tree-based mining of frequent itemsets from uncertain data. In: Proceedings of the DASFAA 2012, Part I, pp. 272–287. Springer (2012)

[13]. Chi, Y., Muntz, R.R., Nijssen, S., Kok, J.N.: Frequent subtree mining- an overview. In: Proceedings of the PAKDD 2013 66(1–2), 161–198 (2004)

[14] .Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in

[15]. http://www.gartner.com/resId=2057415

[16] . Wienhofen, L.W.M., Mathisen, B.M., Roman, D.: Empirical Big Data Research: A Systematic Literature Mapping. http://arxiv.org/pdf/1509.03045.p

[17] http://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm

BIBLIOGRAPHIE

[18]. Lukoianova, T., Rubin, V.: Veracity Roadmap: is big data objective, truthful and credible? Adv. Classif. Res. Online 24(1) (2014). doi:10.7152/acro.v24i1.14671

[19] Leung, C.K.-S., Hayduk, Y.: Mining frequent patterns from uncertain data with MapReduce for Big Data analytics. In: Proceedings of the DASFAA 2013, Part I, pp. 440–455. Springer (2013).

[20] : Vaughn, R. B, J. Farrell, R. Henning, M. Knepper, et K. :Fox Sensor fusion and automatic vulnerability analysis. In Proceedings of the 4Th International Symposium on Information and Communication Technologies, Cape Town, South Africa, pp. 230–235 (2005).

[21] Mohamed Anis Bach Tobji , Boutheina Ben Yaghlane Extraction des itemsets fréquents à partir de données évidentielles : application à une base de données éducationnelles

[22] Amira Gherboudj these de doctora « Méthodes de résolution de problèmes difficiles académiques »2012/2013

[23]Optimization by Simulated Annealing[Article]/ aut. Kirkpatrick, Gelatt et Vecchi// Science, New Series.-13 Mai 1988.-4598.-pp. 671-680.

[24]L'Algorithme Metropolis-HastingsProjet de recherche CRSNGVanessa Bergeron Laperrière(supervisée par Mylène Bédard)Été 2010

[25] F Glover and M Laguna. Kluwer Academic Press, London, Tabu Search 1997

[26] E.A Feigenbaum et J.Feldman(Edirors),Computers and thought.McGraw-Hill Inc pp.192 new york 1963.

[27] M.Dorigo Optimization,learning and natural algorithms.Italy,Phd thesis,DEI Politicnico di Milano(1992).

[28]J.H Holland. Genetic Algorithms and the optimal allocation of trials,SIAM journal of computing.Vol2 N2 pp 88-105,1973

[29] Habiba Drias , Souhila Sadeg , and Safa Yahi Cooperative Bees Swarm for Solving the Maximum Weighted Satisfiability Problem

[30] HANS-GEORG BEYER and HANS-PAUL SCHWEFEL Evolution strategies A comprehensive introduction

[31] Colorni A., Dorigo M., Maniezzo V., « Distributed Optimization by ant colonies »,First Euro-pean Conference on Artificial Life, p. 134-142, 1991

[32] [Chan et Tiwari, 2007]: Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem.

BIBLIOGRAPHIE

[33] *Fogel et al*. (*1966*), Artificial Intelligence through Simulated Evolution, John Wiley

[34] Goldberg, D.E. (1989b)Genetic Algorithms in Search, Optimization and Machine Learning,Addison-Wesley, Reading, Massachussetts

[35] K. K. Rohitha G. K. Hewawasam, Kamal Premaratne, Senior Member, IEEE, and Mei-Ling Shyu, Senior Member, IEEE Rule Mining and Classification in a Situation Assessment Application: A Belief-Theoretic Approach for Handling Data Imperfections

[36] Karaboga Dervis and Akay Bahriye.A comparative study of ArtificialBee Colony algorithm. PhD thesis, Erciyes University, The Departmentof Computer Engineering, Melikgazi, 38039 Kayseri, Turkey, 2002

[37] D.Shafer, G. A: mathematical theory of evidence. Princeton University Press. (1976).

[38] Hewawasam, K.K.R., Premaratne, K., Shyu, M.-L., Subasingha, S.P.: Rule mining and classification in the presence of feature level and class label ambiguities. In: SPIE 5803, Intelligent Computing: Theory and Applications III, p. 98 (2005).

[39] M. Tobji, B. Yaghlane, and K. Mellouli :Frequent itemset mining from databases including one evidential attribute, Scalable Uncertain. Manag., 2008.

[40] A. Samet, E. Lefèvre, and S. Yahia: Mining frequent itemsets in evidential database,Knowl. Syst. Eng., 2014.