République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي والبحث العلمي

Université SAAD DAHLEB - Blida 1
Faculté des Sciences

Département informatique

PROJET DE FIN D'ÉTUDES

Pour l'obtention du titre de Master en Informatique

Option : Systèmes Informatiques et Réseaux

Thème

Extraction d'Itemsets Fréquents Basée sur l'Optimisation des Loups Gris dans un Environnement de Big Data

Réaliser par :
- **AOUES Mohamed Anis**

Encadre par :
- **Dr. ZAHRA Fatma Zohra**

Soutenu devant le jury composé de :
- **M. BALA Mahfoud**, Maître de Conférences, Département d'Informatique, U. Blida - **Président**
- **Mm HIRECHE Celia,** Maitre de Conférences, Département d'Informatique, U. Blida - **Examinateur**

Promotion : 2022/2023

Résumé

Dans l'ère du Big Data, l'extraction d'itemsets fréquents devient un défi crucial en raison de la taille énorme des ensembles de données et du nombre gigantesque des attributs nécessaire à la représentation des objets dans plusieurs domaines. Les approches traditionnelles, y compris les versions parallèles et distribuées des algorithmes classiques, restent souvent inefficaces, ce qui met en lumière la nécessité d'approches alternatives plus efficaces.

On propose dans ce travail une méthode basée sur l'Optimisation du Loup Gris (Grey Wolf Optimization, GWO) pour l'extraction d'itemsets fréquents à partir de grandes bases de données. En exploitant la hiérarchie et le comportement social des loups gris, nous développons une version distribuée et parallèle de GWO, capable de gérer efficacement des volumes de données massifs. Notre approche est adaptée aux paradigmes du Big Data, offrant une solution prometteuse à la problématique de l'extraction d'itemsets à grande échelle.

L'approche proposée a été capable d'obtenir des résultats satisfaisants en termes de qualité de motifs extraits, tout en maintenant un temps d'exécution acceptable, ce qui en fait une solution prometteuse pour l'extraction de motifs dans un contexte du big data.

Mots clés : Extraction d’Itemsets Fréquents, Big Data, Parallèle et distribuée, GWO, Spark.

Abstract

In the era of Big Data, the extraction of frequent itemsets becomes a crucial challenge due to the enormous size of the data sets and the vast number of attributes needed to represent objects in various fields. Traditional approaches, including parallel and distributed versions of classic algorithms, often remain ineffective, highlighting the need for more efficient alternative approaches.

In this work, we propose a method based on Grey Wolf Optimization (GWO) for the extraction of frequent itemsets from large databases. By exploiting the hierarchy and social behavior of grey wolves, we develop a distributed and parallel version of GWO, capable of effectively handling massive data volumes. Our approach is adapted to the paradigms of Big Data, offering a promising solution to the challenge of large-scale itemset extraction.

The proposed approach was able to achieve satisfactory results in terms of the quality of extracted patterns, while maintaining an acceptable execution time, making it a promising solution for pattern extraction in the context of big data.

Keywords: Frequent Itemset Mining, Big Data, Parallel and Distributed, GWO, Spark.

ملخص

في عصر البيانات الضخمة ، أصبح استخراج مجموعات العناصر المتكررة تحديًا حاسمًا نظرًا للحجم الهائل لمجموعات البيانات والعدد الهائل من السمات اللازمة لتمثيل الكائنات في مختلف المجالات. غالبًا ما تظل الأساليب التقليدية ، بما في ذلك الإصدارات المتوازية والموزعة من الخوارزميات الكلاسيكية ، غير فعالة ، مما يبرز الحاجة إلى مناهج بديلة أكثر كفاءة.

في هذا العمل ، نقترح طريقة تعتمد على Gray Wolf Optimization (GWO) لاستخراج مجموعات العناصر المتكررة من قواعد البيانات الكبيرة. من خلال استغلال التسلسل الهرمي والسلوك الاجتماعي للذئاب الرمادية ، نقوم بتطوير نسخة موزعة ومتوازية من GWO ، قادرة على التعامل بفعالية مع أحجام البيانات الضخمة. تم تكييف نهجنا مع نماذج البيانات الضخمة ، مما يوفر حلاً واعدًا لتحدي استخراج العناصر على نطاق واسع.

كان النهج المقترح قادرًا على تحقيق نتائج مرضية من حيث جودة الأنماط المستخرجة ، مع الحفاظ على وقت تنفيذ مقبول ، مما يجعله حلاً واعدًا لاستخراج الأنماط في سياق البيانات الضخمة.

الكلمات الرئيسية: التعدين المتكرر لمجموعة العناصر ، البيانات الضخمة ، الموازية والموزعة ، GWO، Spark.

Table des matières

Liste des tableaux

Liste des figues

Liste des abréviations

GWO: Gray Wolf Optimizer.

FIM: Frequent Itemset Mining.

UDB: Uncertain Database.

GA: Genetics Algorithm.

ES: Evolution Strategy.

ACO: Ant Colony Optimization.

BCO: BEE Colony Optimization.

PSO: Particle Swarm Optimization.

BSO: Bee Swarm Optimization.

BGWO: Binary Grey Wolf Optimization

FIM-SBGO: Frequent Itemset Mining Sequential Binary Grey Wolf Optimization.

HDFS: Hadoop Distributed File System.

RDD: Resilient Distributed Dataset.

SPGWO-FIM: Spark Grey Wolf Optimization for Frequent Itemset Mining.

INTRODUCTION GENERALE

Motivation et problématique

L'ère du "Big Data" a introduit des opportunités sans précédent ainsi que des défis majeurs dans de nombreux domaines, allant de la santé à la bio-informatique, en passant par le commerce électronique. De grandes quantités de données sont générées quotidiennement, et l'un des principaux défis est de les exploiter efficacement pour en extraire des informations significatives. L'extraction de motifs est l'une des techniques d'exploitation de données les plus courantes et elle vise à découvrir des motifs utiles, inattendus et intéressants dans un ensemble de données.

Cependant, l'extraction de motifs à partir de grands volumes de données est une tâche NP-difficile. Les algorithmes traditionnels d'extraction de motifs rencontrent des problèmes d'efficacité et de passage à l'échelle lorsqu'ils sont appliqués à de grands volumes de données. Bien que des méthodes parallèles et distribuées aient été proposées pour remédier à ce problème, elles restent insatisfaisantes. Les solutions approchées basées sur des métaheuristiques à population de solutions ont également été proposées, mais elles présentent également leurs propres défis.

Dans ce contexte, la problématique de cette recherche est : Comment peut-on améliorer l'efficacité et la scalabilité de l'extraction d'itemsets à partir de grands volumes de données. L'utilisation des méthodes approchées basées généralement sur les métaheuristiques ont montré leur efficacité sur des données de taille moyenne [1]. Par conséquent, le passage vers des ensembles de données plus volumineux (Voire d'ordre de Big Data) par la distribution et la parallélisation de ses méthodes peut être une piste de recherche prometteuse.

Objectif

L'optimisation du loup gris (Grey Wolf Optimisation, GWO) est une technique intelligente de l'essaim développée en 2014 [2], qui imite la hiérarchie de leadership des loups qui sont bien connus pour leur chasse de groupe. Cet algorithme imite le leadership social comportement de chasse des loups gris dans la nature. Cette métaheuristique a montré des résultats prometteurs dans divers domaines. Ce qui nous a poussés à l'adapter pour l'extraction d'itemsets fréquents.

En d'autres termes, l'objectif de travail est le développement d'une méthode approchée pour l'extraction d'itemsets fréquents à partir de données de masse en utilisant l'optimisation basée sur le Grey-Wolf (GWO). Plus précisément, l'objectif est de développer une version

distribuée et parallèle de cette méthode pour assurer le passage vers des données plus volumineux.

Organisation du mémoire

Hormis l'introduction, le reste de ce mémoire est organisé comme suit :

- Le chapitre 1 est une revue de la littérature des algorithmes d'extraction des itemsets fréquents en particulier à partir de données de masse.
- Le chapitre 2 présente les métaheuristiques, en particulier l'optimisation des loups gris (GWO).
- Le chapitre 3 explicite l'approche proposée pour l'extraction d'itemsets fréquents à partir de données de masse en se basant sur GWO.
- Le chapitre 4 décrit l'implémentation et l'évaluation de la méthode proposée.

On clôture ce mémoire par une conclusion et des perspectives pour des recherches futures.

Chapitre 1 : Extraction d’Itemset Fréquent et Big Data

1. Introduction

Data Mining (fouille de données) est le cœur du processus de découverte de connaissances. Elle joue un rôle important dans la découverte de modèles significatifs qui peuvent exister fréquemment dans les données transformées. Au milieu de diverses techniques du Data Mining, l'extraction de motifs fréquents (Frequent Pattern Mining ou FPM) est l'une des techniques les plus importantes en raison de sa capacité à localiser les relations répétitives entre différentes transactions dans un ensemble de données.

De nombreux algorithmes de découverte motifs fréquents ont été proposés dans la littérature. Ce chapitre explicite les différentes approches de l'extraction des motifs fréquents ainsi que leurs importants algorithmes. Le terme motif dans notre travail désigne itemsets (ensemble d'attributs), par conséquent on utilise l'abréviation FIM pour désigner (Frequent Itemset Mining ou extraction d'itemset frequent).

2. Extraction d'itemsets fréquents

Extraction d'itemset fréquent (Frequent itemset mining, FIM), d'abord proposée par Agrawal, Imielinski, et Swami (1993), est devenue une technique d'exploration de données populaire et joue un rôle fondamental dans de nombreuse tâches de data mining [3].

2.1. Définition

Une définition généralisée de FIM est de trouver des modèles fréquents, des associations, des corrélations, des motifs et des structures causales parmi des ensembles d'éléments ou d'objets dans des bases de données de transactions, des bases de données relationnelles et d'autres référentiels d'informations.[4]

Considérant le jeu de données transactionnels, tels que D illustré dans le tableau 1, est composés de m transactions, chaque transaction T_t , $1 \leq t \leq m$, est identifiée par une valeur indexée, tid et comprend un nombre fini d'éléments. A titre d'exemple, la première transaction T_1 dans D (tableau 1) a la valeur tid de 1, et les éléments de transaction sont l'ensemble $T = \{c, f, g\}$. Une transaction ne contient pas d'éléments répétés et doit inclure uniquement les éléments de l'ensemble des n éléments $I = \{a, b, c, d, e, f, g, h, i\}$ composant D. Puisque le motif, ou itemset, $P = \{cf\}$ est inclus dans T_1, cette transaction appartient à la couverture de P. Le support de P, ou $sup(P)$, est la cardinalité de la couverture d'ensemble (P). Par exemple, pour l'itemset $= \{cf\}$, $sup(P) = 2$ puisque couverture(P) = { T_1 , T_5}. La couverture du motif P est également notée $\psi(P)$. L'itemset

$P' = \{ cfg \}$ est un sur-ensemble de P, alors que pattern $P'' = \{ c \}$ est un sous-ensemble de P. L'itemset P est fréquent si $sup(P) \geq min\_sup$ avec min_sup étant une valeur donner par l'utilisateur.

Tableau 1: Ensembles de données transactionnelles. [5]

tid	Items
1	c, g, f
2	e, a, c, b
3	e, c, b, i
4	b, f, h
5	b, f, e, c, d

2.2. Données précises et données incertaines pour l'extraction des itemsets

Les données à partir desquelles l'extraction d'itemsets fréquents est faite peuvent être précises comme elles peuvent être incertaines.

2.2.1. Données précises

Les données précises (déterministes) sont des données pour lesquelles nous avons une connaissance complète et précise. Il n'y a pas d'incertitude quant à leur valeur. Dans le contexte l'extraction d'itemsets fréquents sur des bases de données certaines les définitions d'itemset et transaction son comme vu dans l'exemple de définition de FIM.

2.2.2. Données incertaines

Les données incertaines sont des données qui contiennent du bruit (erronées, inutiles ou non pertinentes) qui les fait s'écarter des valeurs correctes, prévues ou originales. Les données de ce type peuvent provenir de diffèrent source des réseaux de capteurs, texte où le texte bruyant se trouve en abondance sur les réseaux sociaux par exemple, dans un système de localisation GPS, la position exacte d'un utilisateur peut être considérée comme une "donnée incertaine" en raison de divers facteurs, tels que l'erreur de mesure du GPS, la réflexion des signaux GPS sur les bâtiments environnants. Voici quelques définitions de base sur l'extraction d'itemsets fréquents sur des bases de données incertaines [6] :

- **Itemset :** un item incertain est un item x ∈ I dont la présence dans une transaction t ∈ T est définie par une probabilité existentielle P (x ∈ t) ∈ (0, 1).
- **Une transaction incertaine :** t est une transaction qui contient des éléments incertains.
- **Base de données incertaine :** Une base de données de transactions T contenant des transactions incertaines est appelée une base de données de transactions incertaines.

Le tableau ci-dessous représente une base de données avec des transactions incertaines :

Tableau 2: Base de données incertaine. [7]

TID	Transactions
T1	A (0.8) B (0.2) C (0.9) D (0.7) F (0.8)
T2	A (0.8) B (0.7) C (0.9) E (0.5)
T3	A (0.5) C (0.8) E (0.8) F(0.3)
T4	B (0.5) D (0.5) F (0.7)

3. Algorithmes d'extraction d'itemsets fréquents

Les algorithmes de FIM peuvent être comme des approches exactes ou des approches approchées basées généralement sur des métaheuristiques.

3.1. Algorithmes classiques d'extraction d'itemsets fréquents à partir des données précises

3.1.1 L'algorithme Apriori

L'algorithme Apriori utilise une technique de recherche itérative par niveau pour découvrir (k + 1)-itemsets à partir de k-itemsets. Un échantillon de données transactionnelles qui se compose d'articles de produits achetés lors de différentes transactions est présenté dans le tableau 2. Tout d'abord, la base de données est analysée pour identifier tous les ensembles d'éléments fréquents en comptant chacun d'eux et en capturant ceux qui satisfont au seuil de support minimum. [7]

Tableau 3: Échantillon de transactions. [8]

TID	Liste des Item
T100	I1. I2, I3
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

L'identification de chaque ensemble d'items fréquents nécessite de balayer toute la base de données jusqu'à ce qu'il ne soit plus possible d'identifier des k-itemsets plus fréquents. Selon la Figure 2, le seuil de support minimum utilisé est de 2. Par conséquent, seuls les enregistrements qui remplissent un nombre de support minimum de 2 seront inclus dans le prochain cycle de traitement de l'algorithme. [8]

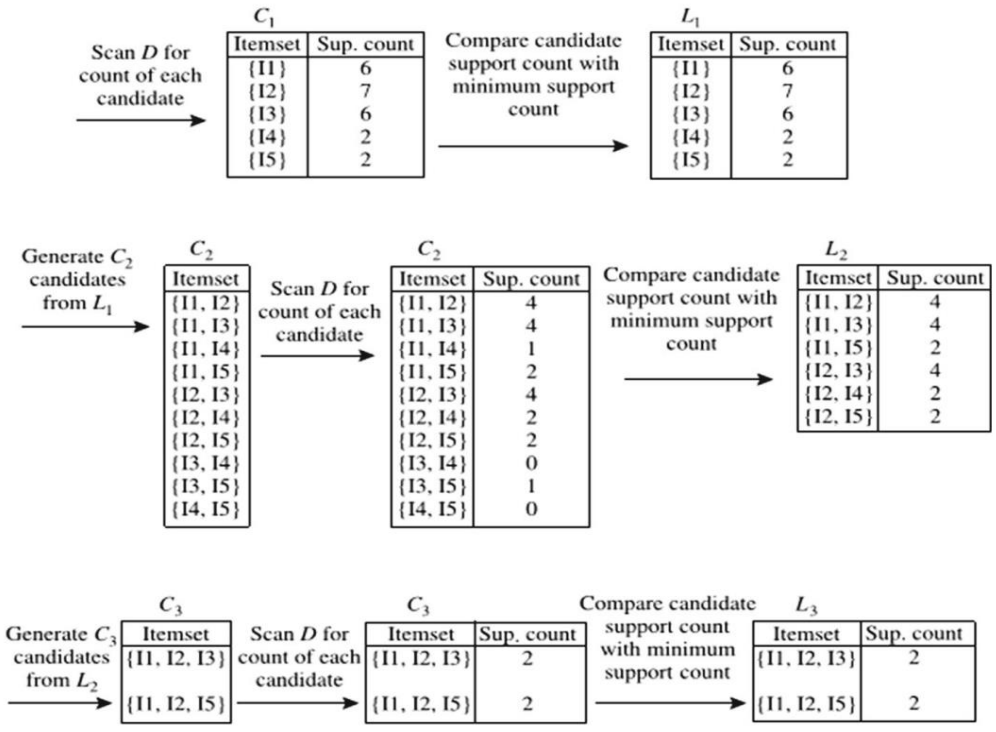

Figure 1: Génération d'itemsets candidats et d'itemsets fréquents. [8]

3.1.2 L'Algorithme FP-Growth

Frequent Pattern Growth (FP-Growth) est un algorithme qui exploite des ensembles d'items fréquents sans processus coûteux de génération de candidats. Il implémente la technique de division pour mieux régner pour compresser les éléments fréquents dans un arbre de modèles fréquents (Frequent pattern Tree « FP-Tree ») qui conserve les informations d'association des éléments fréquents. FP-Tree est ensuite divisé en un ensemble d'arbres FP-Tree conditionnels pour chaque élément fréquent afin qu'ils puissent être extraits séparément [8]. Un exemple d'un FP-Tree qui représente les éléments fréquents est illustré dans la Figure 3.

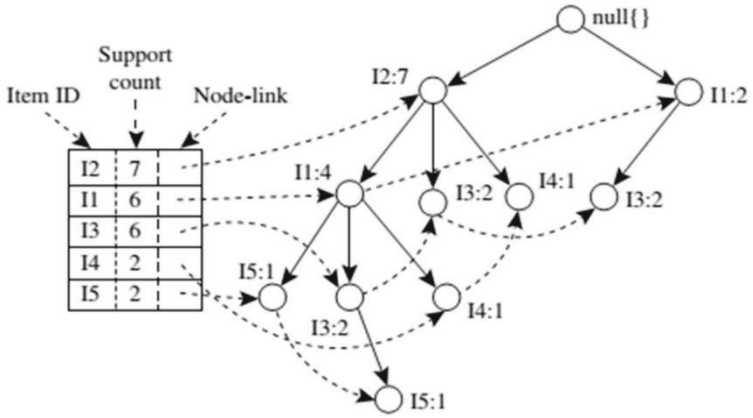

Figure 2: Frequent pattern tree (FP-Tree). [8]

L'algorithme FP-Growth résout le problème de l'identification de longs modèles fréquents en recherchant à plusieurs reprises dans des arbres FP conditionnels plus petits. Un exemple de FP-Tree conditionnel associé au nœud I3 est illustré dans la Figure 4, et les détails de tous les FP-Trees conditionnels trouvés dans la Figure 3 sont indiqués dans le Tableau 2. " Qui se compose de chaque chemin de préfixe dans l'arborescence FP qui coexiste avec chaque élément fréquent de longueur 1. Il est utilisé pour construire FP-Tree conditionnel et générer tous les modèles fréquents liés à chaque élément fréquent de longueur

De cette manière, le coût de la recherche des motifs fréquents est sensiblement réduit. Cependant, la construction du FP-Tree prend du temps si l'ensemble de données est très volumineux [8].

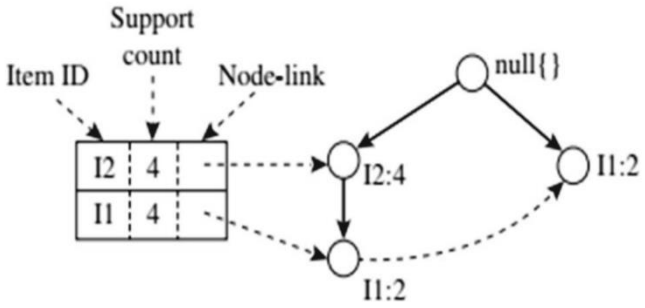

Figure 3: FP-Tree conditionnel associé au nœud I3 [8].

3.1.4 Comparaison des algorithmes

Le tableau suivant présente quelques avantages et inconvénients des algorithmes explicités précédemment.

Tableau 4 : Comparaison des algorithmes de base de Frequent Itemset Mining. [8]

FIM Algorithme	**Avantage**	**Inconvénient**
Apriori	-Utilise une technique de recherche itérative par niveau pour découvrir (k + 1)-itemsets à partir de k-itemsets.	-Doit produire un grand nombre d'ensembles candidats si les k-itemsets sont plus nombreux. -Doit parcourir la base de données à plusieurs reprises pour déterminer le nombre de supports des itemsets.
FP-Growth	-Préserve les informations d'association de tous les ensembles d'éléments. -Réduit la quantité de données à rechercher.	-Construire le FP-Tree prend du temps si l'ensemble de données est très grand.

3.2. Extraction d'itemsets fréquents à partir de données incertaines

Dans les données déterministes, il est clair qu'un itemset est fréquent si et seulement si le support (fréquence) d'un tel itemset n'est pas inférieur à un support minimum spécifié,

min_sup. Cependant, à la différence du cas déterministe, la définition d'itemset fréquents sur des données incertaines à deux explications sémantiques différentes : itemset fréquents basé sur le support attendu (expected support-based frequent itemset) et itemset fréquents probabiliste (probabilistic frequent itemset). Les deux considèrent le support d'un itemset comme une variable aléatoire discrète [7].

Soit $I = \{i_1, i_2, \dots, i_n\}$ un ensemble d'éléments distincts. Nous nommons un sous-ensemble non vide, X de I comme itemset. Par souci de concision, nous utilisons $X = x_1\, x_2 \dots x_n$ pour désigner l'ensemble d'éléments $X = \{x_1, x_2, \dots, x_n\}$. X est un $l - itemset$ s'il a l éléments [7].

Tableau 5: Base de données incertaine UDB. [7]

TID	**Transactions**
T1	A (0.8) B (0.2) C (0.9) D (0.7) F (0.8)
T2	A (0.8) B (0.7) C (0.9) E (0.5)
T3	A (0.5) C (0.8) E (0.8) F(0.3)
T4	B (0.5) D (0.5) F (0.7)

Étant donné UDB une base de données de transactions incertaines, chaque transaction est désignée par un tuple $< tid, Y >$ où tid est l'identifiant de la transaction et :

$$Y = \{y_1(p_1), y_2(p_2), \dots, y_m(p_m)\}.$$

Y contient m unités. Chaque unité a un élément y_i et une probabilité, pi, indiquant la possibilité que l'élément y_i apparaisse dans le tuple tid. Le nombre de transactions contenant X dans UDB est une variable aléatoire, notée $sup(X)$. Compte tenu de UDB[7], l'ensemble d'éléments fréquents basé sur le support attendu et les ensembles d'éléments fréquents probabilistes sont définis comme suit :

Tableau 6: Distribution de probabilité de sup(A). [7]

$Sup(A)$	0	1	2	3
Probabilité	0.1	0.18	0.4	0.32

3.2.1. Expected support-based frequent itemset

Étant donné une UDB de transactions incertaines qui comprend N transactions et X itemset, le support attendu de X est :

$$esup(X) = \sum_{i=1}^{N} p_i(X)$$

Et un support minimum attendu min_sup, un itemset X est un itemset fréquent si et seulement si $esup(X) \geq N \times min\_sup$.

Étant donné une base de données incertaine illustre dans le tableau précèdent et le support minimum attendu, $min\_esup = 0{,}5$, il n'y a que deux itemset fréquents : A(2.1) et C(2.6) où le nombre entre guillemet est le support attendu de l'itemset correspondant [7].

3.2.2. Probabilistic frequent itemset

Étant donné une UDB qui comprend N transactions, un support minimum min_sup et un itemset X, la probabilité fréquence de X, notée $Pr(X)$,[7] est représentée comme suit :

$$Pr(X) = Pr\{sup(X) \geq N \times min_{sup}\}. [7]$$

pft étant le seuil de fréquent probabiliste, en dit que l'itemset X est fréquent si ça sa probabilité fréquente est supérieure au seuil fréquent probabiliste, à savoir :

$$Pr(X) = Pr\{sup(X) \geq N \times min_{sup}\} > pft. [7]$$

Étant donné $min\,sup = 0{,}5$ et $pft = 0{,}7$ et la distribution de probabilité du support de A présentée dans le tableau 2[7]. Ainsi, la probabilité fréquente de A est :

$$Pr(X) = Pr\{sup(A) \geq 4 \times 0{,}5\} = Pr\{sup(A) \geq 2\}$$
$$= Pr\{sup(A) = 2\} + Pr\{sup(A) = 3\} = 0{,}4 + 0{,}32 > 0{,}7 = pft. [7]$$

Ainsi, {A} est un itemset fréquent probabiliste.[7]

3.2.3. Algorithmes de base

La plupart des algorithmes utiliser pour des types de données incertaines sont des algorithmes qui ont étais modifier pour être utiliser pour ce type de données.

3.2.3.1 UApriori

UApriori est le premier algorithme d'extraction d'itemset fréquents basé sur l'expected support. Cet algorithme étend l'algorithme Apriori à l'environnement incertain. L'algorithme trouve d'abord tous les itemsets fréquents basés sur le support attendu. Ensuite, il joint à plusieurs reprises tous les i-itemsets fréquents pour produire i + 1- itemset candidats et teste les candidats i + 1-itemset pour obtenir les i + 1-itemsets fréquents. Enfin, il se termine lorsqu'aucun ensemble d'éléments i + 1 fréquents basé sur le support attendu n'est généré [7], [9].

3.2.3.2 UFP-Growth

Semblable à l'algorithme traditionnel FP-Growth, l'algorithme construit également dans un premier temps un arbre d'index, appelé UFP-tree pour stocker toutes les informations de la base de données incertaine. Ensuit à partir de UFP-tree, l'algorithme construit de manière récursive des sous-arbres et trouve des itemsets fréquente basées sur le support attendu. L'arbre UFP-tree pour UDB du tableau 1 est illustrée à la figure suivante lorsque $min\_esup = 0{,}25$. [7][9]

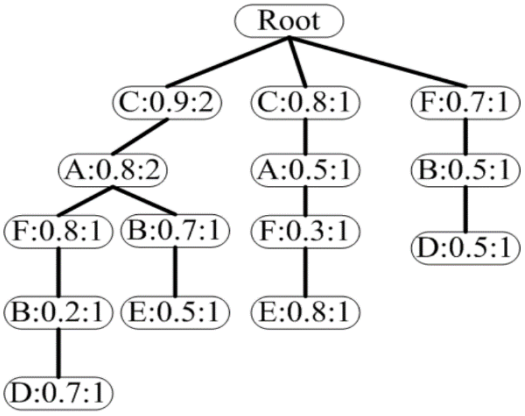

Figure 4: UFP-Tree du Tableau 6. [7]

Le tableau ci-dessous présente quelques avantages et inconvenant de UFP-Growth et UApriori :

Tableau 7: Comparaison des algorithmes de base d'extraction des itemsets fréquent à partir de données incertaines. [9]

FIM Algorithme	Avantage	Inconvénient
UApriori	-Utilise une technique de recherche itérative par niveau pour découvrir (k + 1)-itemsets à partir de k-itemsets.	-souffre d'une dégradation des performances en termes de temp d'exécution.
UFP-Growth	-analyse des bases de données incertaines que deux fois et construit des arborescences.	-souffre d'une dégradation des performances en termes de temp d'exécution et d'utilisation de la mémoire.

4. Méthode approchées pour l'extraction d'itemset fréquents

Les approches exactes pour FIM sont inefficaces sur des bases de données volumineuses, à la fois en termes d'exécution et de consommation de mémoire. Il est donc souhaitable de concevoir des algorithmes plus efficaces pour extraire des itemsets fréquents. À cet égard, différentes stratégies ont été envisagées, telles que la réduction du nombre de balayages de la base de données, l'échantillonnage de la base de données, l'utilisation du parallélisme ou l'ajout de contraintes sur la structure des itemsets. Les approches FIM basées sur les métaheuristiques utilisent des techniques de calcul inspirées de la nature, pour explorer l'espace de recherche.

Les approches basées sur les métaheuristiques peuvent être catégorisée en approches basées sur l'évolution (evolutionary-based) et les approches basées sur l'intelligence en essaim (swarm intelligence based)[18].

5. Extraction d'itemsets fréquents dans le contexte de Big Data

Dans le contexte du Big Data, l'extraction d'itemsets fréquents présente des défis significatifs. Les caractéristiques bien connues du Big Data, comme le volume, la variété, la vélocité et la véracité, rendent difficile l'application directe des algorithmes traditionnels de

FIM tels que Apriori ou FP-Growth. Pour faire face à ces défis, des améliorations ont été apportées à ces algorithmes standards afin de les adapter au nouvel environnement et aux nouvelles exigences.

5.1. Le modèle MapReduce

L'énorme volume du Big Data rend le traitement avec une solution centralisée de manière traditionnelle une mission impossible, c'est pourquoi de nombreuses approches distribuées ont été proposées dans la littérature : Message Passing (MPI), threads, workflow et MapReduce. MapReduce est proposé par Google comme modèle de programmation pour traiter par des algorithmes parallèles et distribués, des jeux de données volumineux et scalables.

Le principe de fonctionnement du MapReduce repose sur la répartition des données sur un cluster pouvant contenir des milliers de machines de manière fiable et tolérante aux pannes. Le travail de MapReduce consiste en trois étapes principales qui sont la procédure Mapper, la procédure Shuffle et la procédure Reducer. Cependant, seules les procédures de mappeur et de Reducer doivent être implémentées puisque Shuffle est exécuté automatiquement par l'ordinateur [10].

- **Map (étape de mappage) :** Cette étape consiste à traiter les données et à les diviser en paires clé-valeur distinctes .
- **Shuffle (étape de mélange) :** Cette étape rassemble toutes les données partageant la même clé.
- **Reduce (étape de réduction) :** Cette étape prend les groupes de données organisés par la fonction Shuffle et les réduit à une forme plus petite ou plus maniable selon certains critères.

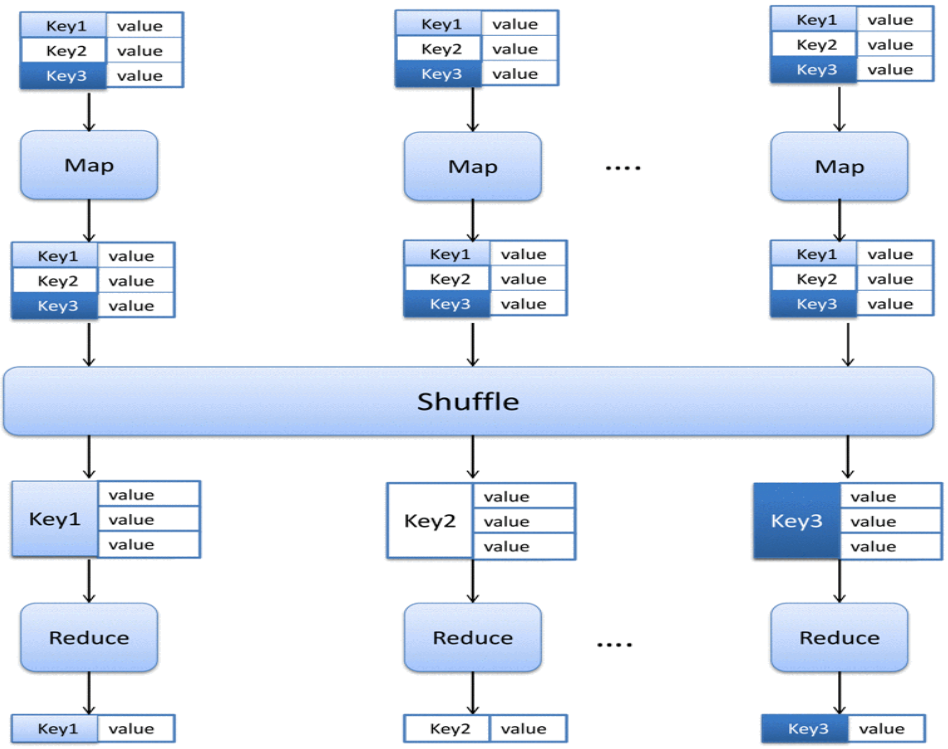

Figure 5: Organigramme du modèle MapReduce. [11]

Dans le processus initial (comme montre la figure ci-dessus) [10], [11], les données qui doivent être traitées par MapReduce sont segmentées en divers blocs. Donc, la première étape implique l'utilisation de la fonction Map pour traiter chaque bloc. De cette façon, toutes les données entrantes sont traitées en parallèle, et le résultat de la fonction Map est un ensemble de paires (clé, valeur).

Ensuite, la fonction Shuffle est mise en action automatiquement afin de rassembler les données associées à la même clé. La dernière phase du processus est la fonction Reduce. Au sortir de la phase Shuffle, nous disposons de plusieurs clés uniques, chacune associée à plusieurs valeurs.

L'objectif de la phase de réduction est de déterminer la valeur optimale pour chaque clé en fonction de critères spécifiques.

5.2. Stratégies de parallélisation pour l'extraction d'itemsets fréquents

La parallélisation et un processus par lequel les tâches de calcul sont divisés en sous-tâches plus petites qui peuvent être exécutées simultanément. Utilisée pour accélérer le

traitement. Deux approches algorithmiques principales sont proposées pour traiter l'exécution parallèle des algorithmes de FIM au moyen du paradigme MapReduce [12] :

1. Approche de fractionnement des données (Data Split) : Il divise le problème en sous-problèmes "similaires", exécutant la même fonction sur différents blocs de données. Enfin, les résultats locaux émis par chaque sous-problème/tâche sont fusionnés pour calculer le résultat global.
2. Approche de division de l'espace de recherche (Search space split) : Elle divise le problème en attribuant à chaque sous-problème la visite d'un sous-ensemble de l'espace de recherche. Plus précisément, cette approche génère, à partir du jeu de données distribué en entrée, un ensemble de jeux de données projetés, chacun suffisamment petit pour être stocké dans la mémoire principale d'une seule tâche. Chaque jeu de données projeté contient toutes les informations nécessaires pour extraire un sous-ensemble d'itemsets sans avoir besoin de la contribution des résultats des autres tâches. Le résultat final est l'union des sous-ensembles d'itemsets extraits de chaque jeu de données projeté.

Lors de la mise en œuvre d'approches parallèles et distribuées pour l'extraction de motifs fréquents dans le Big Data, plusieurs facteurs clés doivent être pris en compte : le type de division ou la division de l'espace de recherche, l'utilisation de la mémoire principale, les coûts de communication et l'équilibrage de charge. Le tableau ci-dessous résume les principales caractéristiques des deux approches de parallélisation :

Tableau 8 : Comparaison des approches de parallélisation. [12]

Critère	**Fractionnement des données**	**Division de l'espace de recherche**
Type de fractionnement	Chaque sous-problème analyse un sous-ensemble différent de la donnée d'entrée et calcule les supports locaux de tous les itemsets candidats de longueur k sur ses blocs de données. Le résultat final est donné par la fusion des résultats locaux	Chaque sous-problème analyse un sous-ensemble différent d'itemsets/une partie différente de l'espace de recherche. Le résultat final est la réunion des résultats locaux.
Utilisation de la mémoire	L'ensemble candidat de longueur k est stocké dans la mémoire d'une seule tâche.	L'ensemble de données projeté complet est stocké dans la mémoire principale d'une seule tâche.
Coût des communications	Nombre d'itemsets candidats × nombre de mappeurs × nombre d'itérations.	Somme des tailles des jeux de données projetés locaux.
L'équilibrage de charge	L'équilibrage de charge est obtenu en associant les mêmes nombre d'itemsets pour chaque Reducer	Les tâches pourraient être significativement déséquilibrées en fonction des caractéristiques des jeux de données projetés affectés à chaque nœud
Nombre maximum de Mappers	Nombre de bloc de données	Nombre de bloc de données
Nombre maximum de Reducers	Nombre d'itemsets candidats	Nombre d'items

5.3. Facteurs à considérer pour la conception d'une approche parallèle et distribuée

Les facteurs à considérer dans la conception d'une approche parallèle et distribuée d'extraction d'itemsets fréquents sont [13] :

A. Type de découpage de l'espace de recherche

La stratégie adoptée pour découper le problème en sous-problèmes a un impact significatif sur l'approche. L'approche de partition de données divise le problème en sous-tâches similaires, en effectuant la même opération sur des divisions de données distinctes. L'approche de partition de données s'attend à ce que la mémoire principale de chaque mappeur contienne des k-itemsets candidats. Par conséquent, il ne peut pas évoluer avec de grands ensembles de candidats sur des ensembles de données denses. De l'autre côté, la méthode de partition de l'espace de recherche construit une base de données projetée à partir de la base de données d'entrée. Chaque jeu de données projeté comprend les données nécessaires pour extraire un sous-ensemble d'itemsets sans avoir besoin des résultats des autres tâches. Le résultat final est la combinaison des résultats de chaque ensemble de données projetées.

B. Représentation des données

La représentation des données joue un rôle crucial. Cela affecte le processus d'extraction, l'accélération, l'évolutivité et le nombre d'analyses de base de données. Différentes manières de stocker les données peuvent être adoptées, en fonction des exigences et des besoins de développement. La disposition horizontale est un moyen conventionnel de stocker et d'accéder aux informations par rangées. Le deuxième est la disposition verticale, c'est une manière spécifique de stocker les données verticalement sous forme de paires clé/valeur accessibles par des colonnes.

C. Coûts de communication

Les frais généraux de communication sont cruciaux dans un algorithme distribué et parallèle car son réseau peut facilement se transformer en goulot d'étranglement s'il transfère de grandes et multiples quantités d'informations. La surcharge de communication est principalement liée aux résultats des mappeurs envoyés aux réducteurs. La réduction du surcoût de transmission devient donc un facteur important de développement algorithmique.

D. Équilibrage de charge

L’équilibrage de charge est crucial, augmente l'efficacité et les performances du système distribué et permet la migration de charge pour une utilisation efficace des ressources. Le partage des ressources est l'objectif du système distribué sur le cluster. L'équilibrage de

charge divise la tâche entre les nœuds du cluster afin de terminer la tâche rapidement. Les différentes manières de partitionner la tâche affectent considérablement l'équilibre de la charge. Une bonne répartition des tâches à chaque nœud de traitement réduirait le déséquilibre.

5.4. Frameworks distribués

Dans les systèmes typiques de fouille de données, les procédures d'exploitation nécessitent des unités de calcul intensif pour l'analyse et la comparaison des données. Pour l'exploitation de Big Data, étant donné que l'échelle des données dépasse de loin la capacité qu'un ordinateur personnel unique peut gérer, un cadre typique de traitement de Big Data s'appuiera sur des ordinateurs en cluster sur les quelles sera installer un Framework dédié pour faciliter la distribution des tâches.

Hadoop et Spark [14-15] sont les deux implémentations open-source les plus populaires du modèle de programmation MapReduce répondant à une large gamme d'applications de Big Data.

Hadoop [14] fournit une solution pour la persistance des données via une installation de stockage distribuée tolérante aux pannes appelée Hadoop Distributed File System (HDFS). Alors que HDFS forme une partie de la solution pour stocker des volumes massifs de données, une computation plus rapide en parallèle à travers différents nœuds de calcul est réalisée grâce au moteur MapReduce.

Quand, à Spark [15] il est devenu une alternative populaire grâce à sa rapidité car il est conçu pour le traitement en mémoire comparé a Hadoop qui lui utilise la persistance sur disque. Ceci est réalisé grâce à son abstraction de base appelée RDD (Resilient Distributed Datasets) qui est une collection distribuée et immuable d'éléments sur lesquels des opérations peuvent être effectuées en parallèle.

5.5. Approches Big Data pour l'extraction d'itemsets fréquents

La majorité des algorithmes les plus populaires ont été modifiés afin de s'adapter aux exigences du Big Data dont le tableau suivant présente quelques-uns :

Tableau 9: Comparaison de quelques algorithmes d'extraction d'itemsets fréquents à partir de données de masse. [13]

Algorithme	Étend	Stratégie de division	Avantage	Inconvénient
MRApriori	Apriori	Fractionnement des données	- Rapide et efficace, - peut générer rapidement de courts itemset.	- Génère une grande quantité d'itemset fréquents partiels car le temps d'exécution est plus long à chaque nœud.
PFP	FP-Growth	Division de l'espace de recherche	- Un groupe indépendant d'espace de recherche divisé entraîne un faible coût de communication.	- Asymétrie élevée de la charge de travail.
YAFIM	Apriori	Fractionnement des données	- Évolutif et plus rapide que les méthodes basées sur Mapreduce.	- Temps d'exécution élevé si la valeur de support minimum est base.

Discussion

Malgré ces améliorations, ces méthodes peuvent toujours rencontrer des problèmes d'efficacité et de performance lorsqu'elles sont appliquées à des volumes de données extrêmement importants. De plus, le nombre élevé de motifs fréquents potentiels dans le Big Data peut entraîner une explosion combinatoire du nombre d'itemsets à examiner, ce qui rend l'application direct de ces méthodes inefficace.

Dans ce contexte, des approches approximatives ont été proposées, notamment l'utilisation de méta-heuristiques, pour résoudre efficacement les problèmes d'optimisation dans l'exploration de grands ensembles de données. Cependant, malgré ces progrès, il existe toujours des limitations et des défis à surmonter.

6. Conclusion

Ce chapitre a présenté une analyse des technologies récemment utilisées et développées dans le domaine du FIM. Nous pouvons conclure que tous les algorithmes mentionnés précédemment présentent encore des inconvénients, mettant en évidence la nécessité de nouvelles approches d'optimisation. La section suivante traitera la nouvelle approche propose pour contrer les limitations de algorithmes vu précédemment.

Chapitre 2 : Métaheuristiques et l’Optimisation des Loups Gris

1. Introduction

Dans notre quête de solutions optimales, l'optimisation joue un rôle crucial dans divers aspects de notre vie. Que ce soit dans le domaine de l'ingénierie, des affaires ou de la prise de décisions personnelles, nous cherchons constamment à trouver les meilleures solutions, voire les solutions quasi-optimales. L'optimisation consiste à trouver la meilleure solution possible à un problème donné dans des conditions spécifiques [16].

Traditionnellement, les problèmes d'optimisation étaient résolus à l'aide de méthodes exactes garantissant la recherche du meilleur résultat global. Cependant, ces méthodes sont souvent confrontées à des défis en termes de complexité computationnelle et d'évolutivité lorsqu'il s'agit de problèmes réels et complexes. Cela a conduit au développement de nouvelles approches connues sous le nom de métaheuristiques. En effet, ce chapitre est consacré à la définition d'une métaheuristique et la présentation de quelques métaheuristiques les plus connus et les plus utilisées dans différents domaines [16].

2. Définition d'une métaheuristique

Les métaheuristiques sont une classe de techniques d'optimisation offrant des stratégies de résolution de problèmes flexibles. Elles se distinguent par leur capacité à explorer de vastes espaces de solutions et à trouver des solutions quasi-optimales dans des délais raisonnables. Contrairement aux méthodes exactes qui reposent sur des structures de problèmes spécifiques, les métaheuristiques fournissent des algorithmes polyvalents pouvant être appliqués à une large gamme de problèmes d'optimisation. [16]

Le mot métaheuristique est une combinaison de deux mots grecs anciens : le verbe heuriskein et le suffixe meta qui signifient respectivement « trouver » et « au-delà, à un niveau supérieur ». Une métaheuristique est formellement définie comme un processus de génération itératif qui guide une heuristique subordonnée (méthode spécifique utilisée à l'intérieur d'une métaheuristique) en combinant intelligemment différents concepts afin d'explorer et d'exploiter l'espace de recherche ; des stratégies d'apprentissage sont employées pour structurer l'information afin de trouver efficacement des solutions quasi optimales. Deux concepts clé sont utilisés dans les métaheuristiques, exploitation et exploration [16] :

- **L'exploration (diversification)** est nécessaire pour identifier des parties de l'espace de recherche avec des solutions de haute qualité.

- **L’exploitation (intensification)** est importante pour intensifier la recherche dans certains domaines prometteurs de l'expérience de recherche accumulée.

3. Les types de métaheuristiques

Il existe différents critères pour classer les métaheuristiques, dont les plus courants sont les métaheuristiques basées sur la population (Population-Based), et les métaheuristiques basées sur une solution unique (Single-Solution Based). [17]

3.1. Métaheuristiques basée sur une solution unique

Également appelées méthodes de trajectoire. Contrairement aux métaheuristiques basées sur la population, elles partent d'une seule solution initiale et s'en éloignent, décrivant une trajectoire dans l'espace de recherche. Certains d'entre eux peuvent être considérés comme des extensions "intelligentes" des algorithmes de recherche locale [17].

3.2. Métaheuristiques basée sur la population

Les métaheuristiques basées sur la population traitent un ensemble (c'est-à-dire une population) de solutions plutôt qu'une solution unique. Les méthodes basées sur la population les plus étudiées peuvent être classées en approches basées sur l'évolution (evolutionary-based) et les approches basées sur l'intelligence en essaim (swarm intelligence based) [18].

3.2.1. Approches basées sur l'évolution

S’inspire de l'évolution naturelle. Dans le calcul évolutif, un conteneur est défini qui imite à quoi ressemblent les solutions et quels sont les ingrédients. Les systèmes génèrent alors aléatoirement des solutions viables mais pas forcément bonnes car elles ont évidemment été assemblées aléatoirement. Ces solutions ne seront pas très bonnes, nous classons donc ces solutions en utilisant une certaine mesure. Supprimez ensuite les moins bons et conservez ceux qui sont un peu meilleurs. De plus, dans la prochaine génération, d'autres solutions candidates sont créé. Ensuite, mélangez et empruntez à ces solutions qui étaient moins mauvaises dans les dernières générations. Ce processus se poursuit pendant un certain temps jusqu'à ce qu'une certaine condition soit atteinte et souvent le système est capable de proposer des solutions presque optimales [17].

Une forme générique d'une approche basée sur l'évolution de base est présentée par l'algorithme ci-dessous :

1. Initialisation de la population avec des avec des individus aléatoire
2. Evaluation de chaque individu
3. Répéter
4. | Sélection des parents
5. | Recombiner les paires de parents
6. | Muter la progéniture résultante
7. | Evaluation des nouveaux individus
8. | Sélection des individus de la prochaine génération
9. Jusqu'à ce qu'une condition soit satisfaite

Figure 6: Pseudo algorithme d'une approche basée sur l'évolution.

[17]

Chaque itération de l'algorithme correspond à une génération, où une population de solutions candidates à un problème d'optimisation donné, appelées individus, est capable de se reproduire et est soumise à des variations génétiques suivies de la pression environnementale qui provoque la sélection naturelle (survie du plus apte). De nouvelles solutions sont créées en appliquant la recombinaison, qui combine deux ou plusieurs individus sélectionnés (les soi-disant parents) pour produire un ou plusieurs nouveaux individus (les enfants ou la progéniture), et la mutation, qui permet l'apparition de nouveaux traits chez la progéniture pour promouvoir la diversité [17].

La Fitness (la qualité des solutions) des solutions résultantes est évaluée et une stratégie de sélection appropriée est ensuite appliquée pour déterminer quelles solutions seront maintenues dans la génération suivante. Comme condition de terminaison, un nombre prédéfini de générations (ou d'évaluations de fonctions) de processus évolutif simulé est généralement utilisé, ou certains critères d'arrêt plus complexes peuvent être appliqués.

Parmi les approches basées sur l'évolution les plus connues, on peut citer :

1. ***Genetic Algorithm (GA) :*** est sans doute l'algorithme basé sur l'évolution le plus connue et le plus utilisée. Il est expliqué en détail dans [19].

2. ***Evolution Strategy (ES) :*** Semblable à GA, ES imite les principes de l'évolution naturelle comme méthode pour résoudre les problèmes d'optimisation. Il est expliqué en détail dans [20].

3.2.2. Approches basées sur l'intelligence en essaim

Les systèmes Swarm Intelligence consistent généralement en une communauté d'agents ou d'organismes de base communiquant géographiquement entre eux et avec leur environnement [21].

Parmi les approches basées sur l'intelligence en essaim les plus connues, on peut citer :

1. L'optimisation par colonies de fourmis (ACO, pour Ant Colony Optimization) : Cette méthode est inspirée du comportement des fourmis dans la nature. Lorsqu'elles recherchent de la nourriture, les fourmis laissent des traces de phéromones, ce qui permet à d'autres fourmis de suivre leur chemin si elles trouvent de la nourriture. Dans l'ACO, une solution à un problème est analogue à un chemin que suit une fourmi, et la quantité de nourriture est analogue à la qualité de la solution. Pour plus de détails voir [22].

2. L'optimisation par colonies d'abeilles (BCO, pour Bee Colony Optimization) : Cette méthode est inspirée du comportement des abeilles lorsqu'elles recherchent de la nourriture. Dans le BCO, une solution à un problème est analogue à une source de nourriture, et la qualité de la solution est analogue à la quantité de nourriture disponible. Pour plus de détails voir [23].

3- L'Optimisation par essaim de particules (PSO pour Particle Swarm Optimization) : est une technique d'optimisation basée sur la population, la population étant appelée essaim. Une explication simple du fonctionnement de la PSO est la suivante. Au cours de chaque itération, chaque particule accélère dans la direction de sa meilleure solution personnelle trouvée jusqu'à présent, ainsi que dans la direction de la meilleure position globale découverte jusqu'à présent par n'importe quelle particule de l'essaim. Cela signifie que si une particule découvre une nouvelle solution prometteuse, toutes les autres particules se rapprocheront d'elle, explorant ainsi la région de manière plus approfondie. L'optimisation par essaim de particules a été largement utilisée pour résoudre divers problèmes d'optimisation [24]. PSO a été

introduite pour la première fois par James Kennedy et Russell Eberhart en 1995 dans [25].

4- Optimisation par essaims d'abeilles (BSO pour Bee Swarm Optimization) : est un algorithme d'optimisation métaheuristique inspiré du comportement des abeilles lorsqu'elles recherchent de la nourriture. Dans l'algorithme BSO, la population est divisée en deux groupes : les abeilles employées et les abeilles ouvrières. Les abeilles employées recherchent de la nourriture et partagent ensuite l'emplacement de la source de nourriture avec les abeilles ouvrières dans la ruche. Les abeilles ouvrières évaluent ensuite la qualité de la source de nourriture en fonction de l'information partagée et choisissent d'explorer soit la même source de nourriture, soit une nouvelle source. La recherche de nouvelles sources de nourriture est une forme d'exploration, qui permet à l'algorithme d'éviter de se bloquer dans des optima locaux, tandis que l'exploitation des sources de nourriture existantes permet à l'algorithme de converger vers une solution [26].

Nous présentons dans la section suivante les concepts fondamentaux d'une approche basée sur l'intelligence en essaim relativement récente (introduit en 2014) par rapport aux métaheuristiques présentées précédemment, qui est l'Optimisation des Loup Gris (Grey Wolf Optimizer, GWO). Cette méthode d'optimisation est explicitée en détails parce qu'elle fait partie de l'objet de notre travail.

4. L'optimiseur des loups gris

L'optimiseur de loups gris ou Grey Wolf Optimizer (GWO) est inspiré du comportement des loups gris (Canis lupus) qui appartiennent à la famille des canidés. Ils sont considérés comme des prédateurs au sommet, ce qui signifie qu'ils sont au sommet de la chaîne alimentaire. Les loups gris préfèrent généralement vivre en meute. Le la taille du groupe est de 5 à 12 en moyenne.

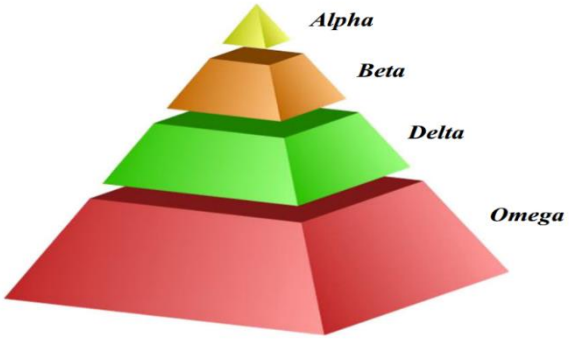

Figure 7: Hiérarchie sociale des loups gris. [27]

Dans chaque meute de loups gris, il existe une hiérarchie sociale commune qui dicte le pouvoir et la domination [27], comme le montre la figure numéro 7.

Les meneurs sont un mâle et une femelle, appelés alphas. L'alpha est principalement responsable de prendre des décisions concernant la chasse, le sommeil lieu, heure de réveil, etc. Le deuxième niveau dans la hiérarchie des loups gris est bêta. Les bêtas sont des loups subordonnés qui aident l'alpha dans la prise de décision ou d'autres activités de meute. Le loup gris le moins bien classé est oméga. L'oméga joue le rôle de bouc émissaire. Les loups Oméga doivent toujours se soumettre à tous les autres loups dominants [2].

Une autre inspiration est l'approche de chasse des loups gris. Lorsqu'ils chassent une proie, les loups gris suivent une série d'étapes efficaces : chasser, encercler, harceler et attaquer. Cela leur permet de chasser de grosses proies [27].

Mirjalili [2] a proposé une métaheuristique immitante le comportement de chasse des loups gris concrétiser par le modèle mathématique illustré dans les sections suivantes.

4.1. Hiérarchie sociale

Afin de modéliser mathématiquement la hiérarchie sociale des loups lors de la conception de GWO, nous considérons la solution la plus adaptée comme alpha (α). Par conséquent, les deuxièmes et troisièmes meilleures solutions sont respectivement nommées bêta (β) et delta (δ). Les autres solutions candidates sont supposées être oméga (ω). Dans l'algorithme GWO, la recherche (optimisation) est guidée par α, β et δ. Les ω loups suivent ces trois loups [2].

4.2 Encerclement de la proie

La première étape de la chasse est chasser et encercler. Pour modéliser mathématiquement cela, GWO considère deux points dans un espace à n dimensions et met à jour l'emplacement de l'un d'eux en fonction de celui de l'autre [2]. L'équation suivante a été proposée pour simuler ceci :

$$X(t+1) = X(t) - A \,.\, D \qquad \text{Équation 1}$$

Où $X(t+1)$ est l'emplacement suivant du loup, $X(t)$ est l'emplacement actuel, A est une matrice de coefficients et D est un vecteur qui dépend de l'emplacement de la proie (X_{p}) [2], et est calculé comme suite :

$D = |C . X_p(t) - X(t)|$ Équation 2

Où :

$C = 2 . r2$

Notez que $r2$ est un vecteur généré aléatoirement à partir de l'intervalle [0,1]. Avec ces deux équations, une solution est capable de se déplacer autour d'une autre solution [2]. Notez que les équations utilisent des vecteurs, cela s'applique donc à n'importe quel nombre de dimensions [2]. Un exemple de positions possibles d'un loup gris par rapport à une proie est montré dans la figure suivante :

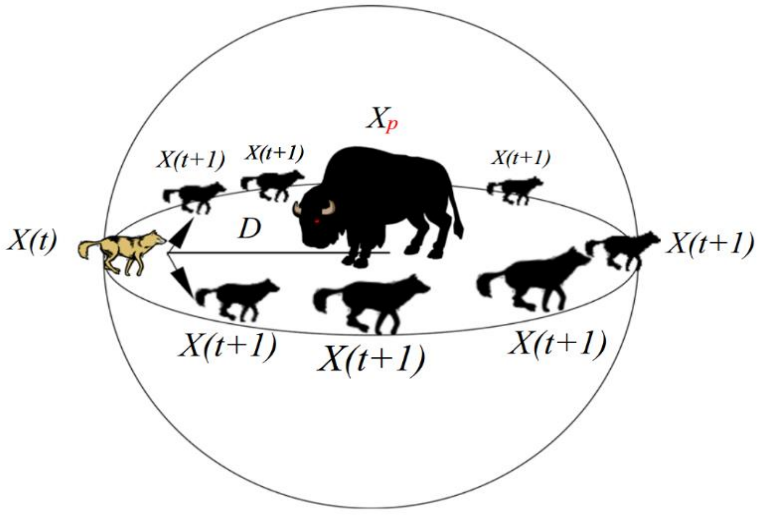

Figure 8: positions d'un loup gris par rapport à une proie. [27]

Les composants aléatoires dans les équations ci-dessus simulent différentes tailles de pas et de vitesses de déplacement des loups gris [2]. Les équations qui définissent leurs valeurs sont les suivantes :

$A = 2a . r1 - a$ Équation 3

Où a est un vecteur dont les valeurs diminuent linéairement de 2 à 0 au cours de l'exécution. $r1$ est un vecteur généré aléatoirement à partir de l'intervalle [0,1], [2]. L'équation pour mettre à jour le paramètre a est la suivante :

$a = 2 - t\,(2/T)$ Équation 4

Où t indique l'itération actuelle et T est le nombre maximum d'itérations.

4.3. La chasse

Dans GWO, on suppose que alpha, bêta et delta sont toujours les trois meilleures solutions obtenues jusqu'à présent. L'optimum global des problèmes d'optimisation est inconnu, il a donc été supposé que alpha, bêta et delta ont une bonne idée de son emplacement, ce qui est raisonnable car ce sont les meilleures solutions dans l'ensemble de la population. Par conséquent, les autres loups devraient être obligés de mettre à jour leurs positions comme suit :

$X(t+1) = \frac{1}{3} X_1 + \frac{1}{3} X_2 + \frac{1}{3} X_3$ Équation 5

Où X_1et X_2 et X_3 sont calculer comme suit avec l'équation (6) :

$X_1 = X_\alpha(t) - A_1 . D_\alpha$

$X_2 = X_\beta(t) - A_2 . D_\beta$ Équation 6

$X_3 = X_\delta(t) - A_3 . D_\delta$

D_α, D_β et D_δ sont calculer avec l'équation (7) :

$D_\alpha = |C_1 . X_\alpha - X|$

$D_\beta = |C_2 . X_\beta - X|$ Équation 7

$D_\delta = |C_3 . X_\delta - X|$

Voici le pseudocode simplifié de GWO [2]:

GWO Algorithm

Initialiser la population de loups (solutions)

Initialisation de a, A, et C

Calculer fitness de chaque loup.

$X_{\propto} = le\ meilleur\ loup$

$X_{\beta} = le\ deuxieme\ meilleur\ loup$

$X_{\delta} = le\ troisieme\ meilleur\ loup$

Tant que ($t < nombre\ Max\ d'iteration$)

Pour chaque loup de la population

Mettre à jour la position du loup courant

Fin pour

Figure 10: Pseudo code de GWO.

6. Conclusion

En conclusion, les métaheuristiques offrent une approche à la fois puissante et flexible pour résoudre des problèmes d'optimisation complexes. Elles peuvent également être appliquées à d'autres types de problèmes, comme l'extraction de motifs fréquents. Dans ce chapitre, nous avons introduit le concept de métaheuristique, exploré les différents types de métaheuristiques, et passé en revue quelques-unes des méthodes les plus connues. Enfin, nous avons détaillé l'Optimisation par Loup Gris (GWO), la métaheuristique que nous avons choisie pour résoudre notre problème. Dans les chapitres suivants, nous verrons comment appliquer GWO à l'extraction de motifs à partir de données massives, et comment adapter et optimiser cet algorithme pour répondre à nos besoins spécifiques.

Chapitre 3 : Approche Proposée

1. Introduction

GWO est un algorithme bio-inspiré qui simule le comportement de chasse des loups gris. Cet algorithme a démontré sa capacité à résoudre efficacement une variété de problèmes d'optimisation complexes, ce qui suggère qu'il pourrait être une solution viable pour l'extraction de d'itemset fréquents dans le Big Data.

Nous proposons donc d'adapter GWO à l'extraction d'itemset fréquents dans le Big Data. Nous pensons que cette approche pourrait améliorer l'efficacité de l'extraction de motifs, tout en gérant efficacement le volume, la vélocité, la variété et la véracité caractéristiques du Big Data.

Cette section présentera en détail notre approche proposée. En premier lieu sera expliquer comment l'algorithme peut être appliqué à l'extraction d'itemsets fréquents et sera présenter notre version séquentielle. Enfin nous expliqueront comment nous avant adapter notre version séquentielle au big data.

2. Motivation de l'adaptation

1. Manque de recherches approfondies sur l'application de GWO a l'extraction d'itemset fréquent dans le contexte du big data.
2. Efficacité et performances : GWO est une métaheuristique récente et puissante qui a démontré de bonnes performances dans la résolution de problèmes d'optimisation complexes [27].
3. Facilité d'implémentation et de compréhension : GWO bénéficie d'une implémentation relativement simple et intuitive [27].

3. Adaptation de GWO à l'extraction d'itemsets fréquents

Dans le contexte de l'utilisation GWO pour l'extraction d'itemsets fréquents, voici comment les concepts clés de GWO sont adaptés :

1. **Loup (Wolf)** : Dans GWO, chaque loup représente une solution candidate à un problème d'optimisation. Lorsqu'il est appliqué à l'extraction d'itemsets fréquents, chaque loup peut être considéré comme une combinaison d'items qui forme un itemset candidat. Chaque loup représente donc un itemset potentiellement fréquent.

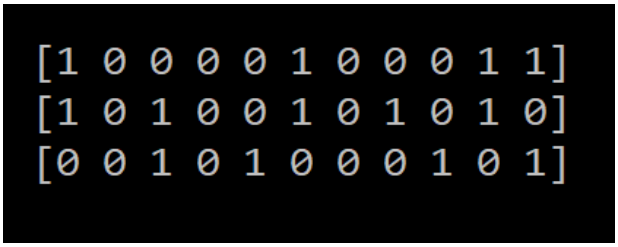

Figure 11: Example d'itemsets candidat

La figure précédente représente une population de trois itemset de longueur 11, tel que la présence de ‘1’ dans une position, signifie qu’un item est présent dans l’itemset et ‘0’ signifie qu’il est absent.

2. **Valeur de fitness (Fitness Value)** : La valeur de fitness mesure la qualité d'une solution candidate dans l'algorithme GWO. Dans le contexte de l'extraction d'itemsets fréquents, la valeur de fitness et la fréquence d'apparition d'un itemset dans un ensemble de données. Plus la valeur de fitness est élevée, plus l'itemset est considéré comme fréquent.
3. **Position d'un loup (Wolf Position)** : Dans GWO, la position d'un loup représente les valeurs des paramètres de sa solution candidate. Dans le contexte de l'extraction d'itemsets fréquents, la position d'un loup correspond aux items qui composent l'itemset candidat.
4. **L'espace de recherche :** représente l'ensemble des solutions potentielles qui seront explorées par l'algorithme

Les concepts d'alpha, beta et delta peuvent être adaptés de la manière suivante :

1. **Alpha (α)** : Dans GWO, le loup alpha est considéré comme le meilleur loup de la meute. Il représente la meilleure solution candidate trouvée jusqu'à présent. Dans le contexte de l'extraction d'itemsets fréquents, le loup alpha peut être associé à l'itemset fréquent le plus prometteur découvert jusqu'à présent. Il représente donc le meilleur résultat obtenu à ce stade de l'algorithme.
2. **Beta (β)** : Le loup beta est le deuxième meilleur loup de la meute dans GWO. Dans le contexte de l'extraction d'itemsets fréquents, le loup beta peut être associé à un itemset fréquent prometteur qui n'est pas aussi bon que l'alpha, mais qui est tout de même considéré comme une solution de qualité.
3. **Delta (δ)** : Le loup delta est le troisième meilleur loup de la meute dans GWO. Dans le contexte de l'extraction d'itemsets fréquents, le loup delta peut représenter un itemset fréquent moins que alpha et beta.

Pour adapter GWO a l’extraction d’itemset fréquent nous utiliseront une version modifier de l’algorithme qui est une version binaire qui sera mieux adapter au type et à la structure des données choisie.

3.1. Binary Gray Wolf Optimization (BGWO)

La version binaire de GWO redéfini l'équation numéro (5) que nous avons vu dans le chapitre métaheuristiques qui est une équation qui servira à mettre à jour de la position du loup courant. L'équation devient :

$$X(t+1) = \begin{cases} 1 \ \ si\ sigmoid\left(\frac{X_1+X_2+X_3}{3}\right) \geq rand \\ 0 \ \ sinon \end{cases} \qquad \text{Équation 8}$$

Ou $rand$ est un nombre dans [1,0]. X_1, X_2, X_3 sont des vecteurs binaires calculer comment définie précédemment et $sigmoid(a)$ est définie comme suite :

$$sigmoid(a) = \frac{1}{1 + e^{-10(a-0.5)}}$$

3.2. Structure de données

Etant donné que l'extraction d'itemset fréquent se base sur la présence ou l'absence d'un ou plusieurs items dans une transaction, nous avons choisie d'utiliser des données binaires. Cette représentation binaire offre des avantages significatifs en termes de stockage, de manipulation et de recherche efficace des itemsets fréquents. Prenant pour exemple le tableau ci-dessous qui illustre une base de données transactionnel contenant 4 transactions :

Tableau 10: Base de données transactionnel

ID		Contenu
1		Pain, Jus, Eau
2		Soda, Pain
3		Eau
4		Pain, Jus, Eau, Soda

Si nous voulions avoir la version binaire du tableau 1 alors Chaque ligne resterait une transaction et chaque colonne deviendra un item. Donc dans une transaction donnée et dans une colonne donnée '1' signifiera qu'un item est présent dans la transaction et '0' qu'un item n'est pas présent dans la transaction. Comme illustrer dans le tableau si dessous :

Tableau 11: Base de données transactionnel binaire

ID		Pain	Jus	Eau	Soda
1		1	1	1	0
2		1	0	0	1
3		0	0	1	0
4		1	1	1	1

3.3. FIM-SBGWO :

FIM-SBGWO est la version séquentielle de notre approche dont voici le pseudocode :

Entrée :

- Dataset.
- M nombre d'itemset dans la population.
- Num_iteration nombre d'itération de l'algorithme.
- Min_sup : seuil minimum.
- N : nombre d'item dans le dataset.

Sortie :

Output : Ensemble d'élément fréquent.

Début

1. Initialisation aléatoire de la population.
2. Calcule du fitness pour chaque itemset de la population (Fréquent d'occurrence de chaque loup de la population dans la base de données)
3. Trie de la population et assignation de :
   Alpha_itemset = l'itemset avec la meilleure valeur de fitness.
   Beta_itemset = l'itemset avec la deuxième meilleure valeur de fitness.
   Delta_itemset = l'itemset avec la troisième meilleur valeur fitness.
4. Pour chaque itemset de la population :
   - Si valeur de fitness supérieur à (min_supp).
   - Ecrire dans output.
5. Fin.
6. Répéter jusqu'à Num_iteration :
7. Pour chaque itemset de la population :
8. - Application de l'équation numéro (8) sur itemset courant.
9. - Calcule de la valeur de fitness pour le nouveau itemset.
10. - si fitness du nouveau itemset supérieur a fitness du loup courant donc Remplacer itemset courant par le nouveau.
11. Fin.
12. Pour chaque itemset de la population :
13. - Si valeur de fitness supérieur à (min_supp).
14. - Ecrire dans output.
15. Fin.
16. Trie de la population et assignation :
    Alpha_itemset = l'itemset avec la meilleure valeur de fitness.
    Beta_itemset = l'itemset avec la deuxième meilleure valeur de fitness.
    Delta_itemset = l'itemset avec la troisième meilleur valeur fitness.
17. Fin.
18. Return Output.

Figure 12 : Pseudocode FIM-SBGWO

4. Adaptation de GWO à l'extraction d'itemsets fréquents dans le contexte de Big Data

La méthode séquentielle présentée précédemment ne peut être appliquée à des ensembles de données très volumineux tout en maintenant un temps d'exécution acceptable, car elle nécessite le chargement en mémoire de l'ensemble des données. Afin d'adapter notre algorithme au Big Data, nous avons développé une approche parallèle et distribuée en utilisant le modèle de programmation MapReduce en utilisant le Framework Spark.

4.1. Spark

Dans le cas d'applications itératives et d'analyses interactives, Hadoop MapReduce n'est pas efficace. Les applications itératives doivent stocker et récupérer des données sur le disque entre chaque itération, ce qui compromet les performances de l'application. Pour remédier à ces limitations, un nouveau cadre de calcul distribué, appelé Spark (Apache Spark, 2016), a vu le jour. Le point clé de Spark est l'abstraction des données à l'aide de RDD (Resilient Distributed Dataset), qui sont des ensembles de données partitionnés par les machines du cluster, et qui peuvent être conservés en mémoire. Il s'agit de collections d'objets immuables, d'objets qui peuvent être récupérés si l'une des machines du cluster tombe en panne. Les applications peuvent réutiliser le même RDD aussi souvent que nécessaire, sans compromettre les performances, puisqu'il n'est pas nécessaire d'écrire et de lire les RDD sur le disque. Par conséquent, les applications développées sur Spark peuvent être jusqu'à 100 fois plus rapides que celles développées sur Hadoop-MapReduce [28].

Les opérations sur les RDD sont de deux types [29]:

1. **Les opérations de transformation :** créent un nouveau RDD en transformant un RDD existant. Par exemple, certaines transformations sont : map, flatMap, filter, reduceByKey.
2. **Les opérations d'action** : écrivent les résultats dans le stockage externe ou retournent au programme Master les résultats, après avoir appliqué la fonction d'action. Par exemple, certaines opérations d'action sont : collect, count, saveAsTextFile.

Spark suit la technique d'évaluation paresseuse sur la chaîne de transformations et d'actions. Tant que l'action n'est pas déclenchée, aucune transformation n'est exécutée.

4.2. L'algorithme SPGWO-FIM

SPGWO-FIM pour "Spark Gray Wolf Optimizer Frequent Itemset Mining", est l'abréviation que nous utiliserons pour notre méthode. Dans cette section, on présente en détails chaque étape de l'approche distribuée parallèle de l'algorithme SBGO-FIM explicitée précédemment qu'on a proposé.

Entrée :

- Dataset.
- Min_sup.
- Nombre d'itération.
- Nombre d'itemset de la population.

Sortie :

- Ensemble d'élément fréquent.

Étape 1 : Lecture du dataset et le stocker dans un RDD

La méthode **textFile(Path, minPartitions)** et utiliser pour lire le dataset de HDFS, minPartitions spécifie le nombre minimal de partitions dans lesquelles le fichier sera divisé.

Lorsque on utilise la méthode textFile de Spark pour lire des données à partir de HDFS (Hadoop Distributed File System), voici ce qui se passe :

1. Spark crée une tâche de lecture de fichier et envoie cette tâche aux nœuds du cluster.
2. Chaque nœud du cluster lit une partie du fichier à partir de HDFS en parallèle.
3. Les données lues à partir du fichier sont divisées en partition (blocs) et chaque partition est traitée par un nœud.
4. Chaque nœud crée un RDD contenant les lignes du fichier qui lui ont été attribuées.

Apres la lecture du fichier on utilise la méthode **persist()** avec Le niveau de stockage MEMORY_ONLY indiquent à chaque nœud de stocker sa partitions en mémoire.

Étape 2 : Initialisation de la population

La population est initialisée à partir du dataset, pour se faire en utiliser l'action take(n) qui prend un nombre comme argument, et renvoie les n premier éléments du RDD_Dataset.

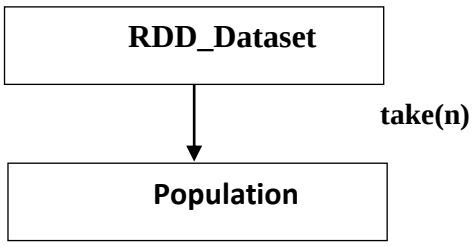

Figure 13: initialisation de la population Etape 2.

La population est ensuite diffusée au diffèrent nœud du cluster ou elle enregistrer en mémoire.

Étape 3 : Calcule de fitness pour chaque itemset de la population

L'étape 3 du processus utilise le modèle MapReduce pour répartir le calcul de la valeur de fitness de chaque itemset de la population sur plusieurs nœuds du cluster, en exploitant la capacité de traitement parallèle.

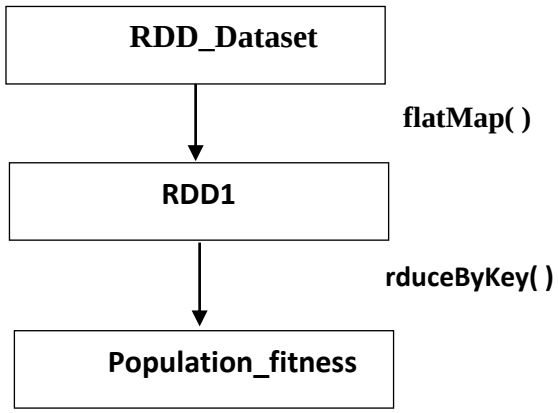

Figure 14: Calcule de fitness Etape 3.

Chaque nœud, exploitant les partitions des jeux de données en mémoire et la population préalablement diffusée, exécute l'opération **flatMap()**. Cette opération, pour chaque élément du jeu de données, applique une fonction qui peut générer plusieurs éléments, ces derniers sont ensuite stockés dans un nouveau RDD. Dans notre méthode, l'opération flatMap, pour chaque élément de la population, renvoie 1 si tous les items d'un itemset sont présents dans une transaction, sinon elle renvoie 0.

La figure ci-dessous illustre l’étape 3 du point de vue de notre cluster :

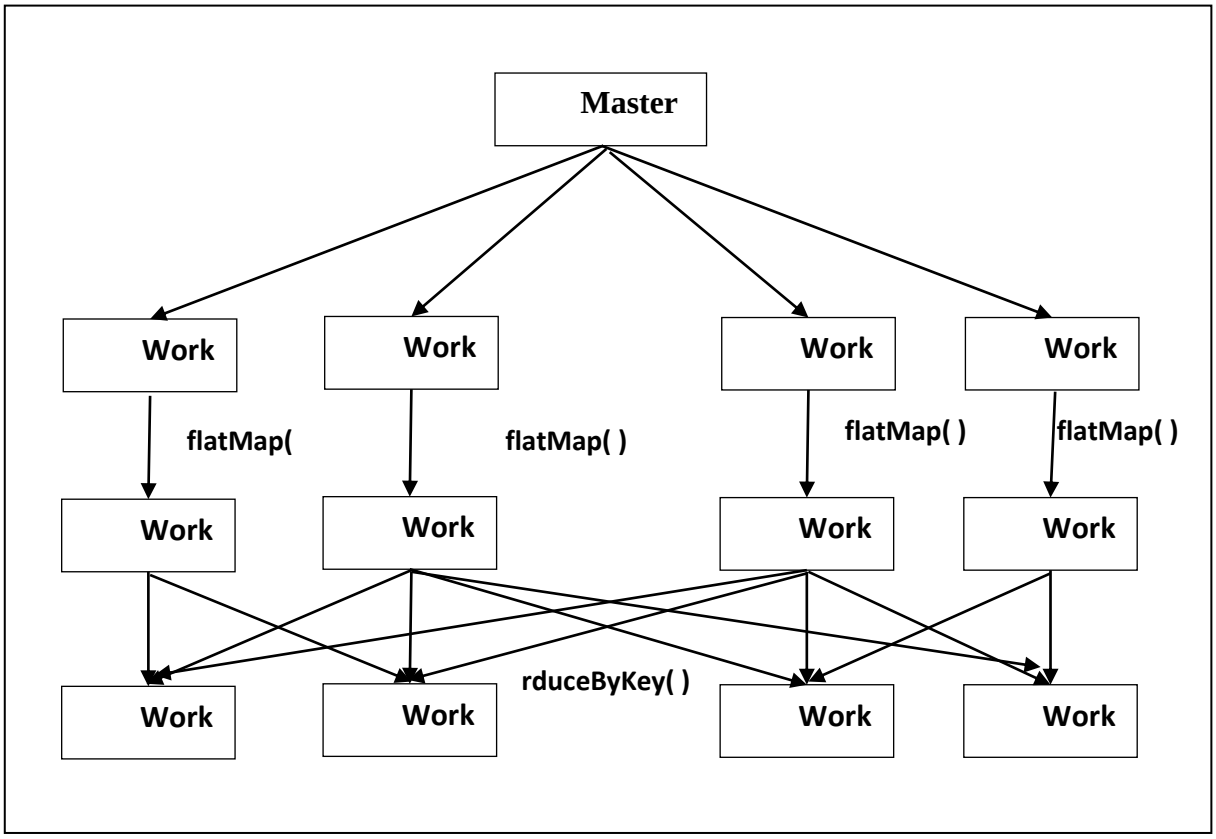

Figure 15: Etape 3 du point de vue du cluster.

Étape 4 : Trie et extraction des trois meilleures solutions

L'action takeOrdered de Spark est utilisée pour renvoyer les trois premiers éléments d'un RDD, classés par ordre croissant ou décroissant en fonction du paramètre d'entrée. takeOrdered exécute d'abord une opération map à travers les partitions du RDD pour trouver les 'n' premiers éléments dans chaque partition, puis il collecte ces éléments vers le Master, où il trouve à nouveau les 'n' premiers éléments. Les données sont triées à l'aide d'un algorithme de tri en tas pour rendre ce processus plus efficace.

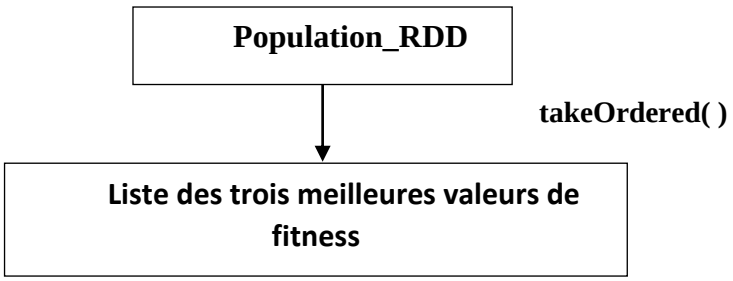

Figure 16: Etape 4.

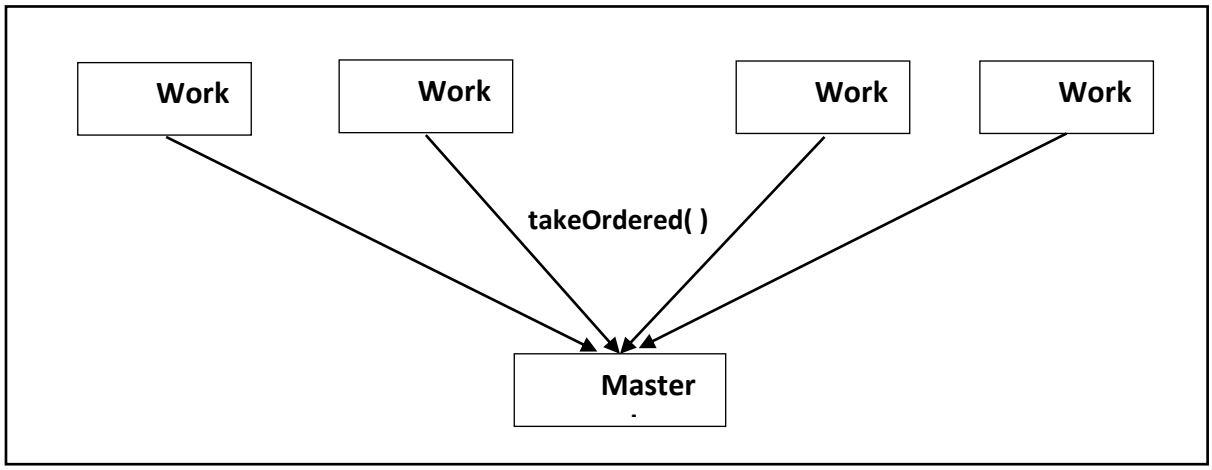

Figure 17: Etape 4 du point de vue du cluster.

Apres application de takeOrdered qui retourne les trois meilleures solutions selon la valeur de fitness, on attribue :

Alpha_item = itemset avec la meilleure valeur de fitness.

Delta_item = itemset avec la deuxième meilleure valeur de fitness.

Beta_item = itemset avec la troisième meilleure valeur de fitness

Ses trois itemset sont ensuite diffuser au nœud et enregistrer en mémoire de chaque nœud.

Étape 5 : Mise a jour des itemset

Mise à jour des itemset en parallèle en utilisant les équations vues précédemment.

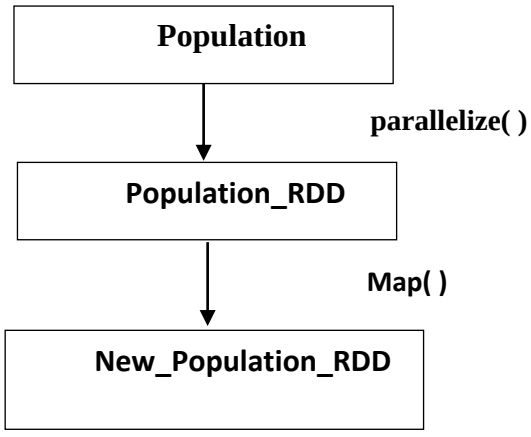

Figure 18: Etape 5

Le transofrmation map( ) pour chaque element du RDD applique une fonction donnée, et renvoie une nouvelle collection qui a la même taille que la collection d'origine.

Étape 6 : Calcule de fitness des nouveau itemset

Calcule de la valeur de fitness de la nouvelle population avec le même processus de calcule de l'étape numéro trois.

Étape 7 : Sélection des meilleurs itemset

Pour chaque itemset de la population (Population_RDD) correspond un itemset de la nouvelle population (New_Population_RDD), la valeur de fitness de ces deux itemsets est alors comparée, et l'itemset avec la meilleure valeur de fitness est sélectionné.

Étape 8 : Rediffusion de la nouvelle population

Suppression de l'ancienne population de la mémoire de chaque nœud et diffusion de la nouvelle population avec les itemset mise à jour.

Étape 9 : Trie et extraction des nouvelles trois meilleures solutions

Application de l'étape numéro trois sur la population avec les itemset mise à jour, mise a jour de Alpha_item , Beta_item et Delta_item et rediffusion des nouvelle valeur.

Étape 10 : Enregistrement des ensemble d'itemset freqeunt

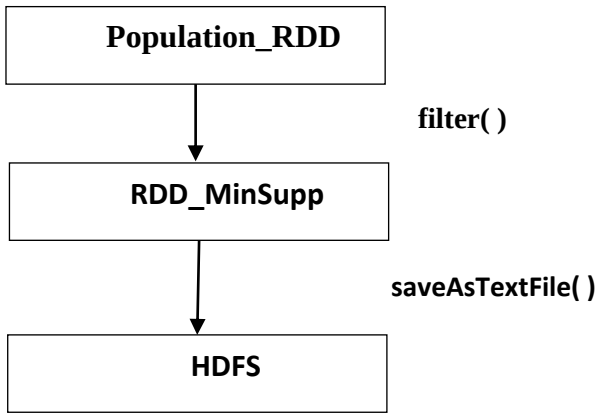

Figure 19: Etape 10

La méthode filter est une opération de transformation qui permet de sélectionner des éléments d'un RDD qui satisfont une condition donnée. Pour notre cas la condition est que la valeur de fitness pour un itemset soit supérieur au nombre minimum d'occurrence calculer en utilisant le support minimum.

saveAsTextFile()permet d'écrire les données du RDD dans HDFS.

Étape 11 : Répéter jusqu'à atteindre le nombre maximum d'itérations

Répéter les étapes 5, 6, 7, 8, 9, 10 jusqu'à atteindre le nombre maximum d'itération.

Conclusion

Dans ce chapitre, nous avons détaillé et explique en premier lieu comment GWO peut être utilisé pour FIM, explication de notre première approche développer qui est une version séquentielle. Enfin, une explication détailler de chaque étape d'exécution de notre approche proposée pour l'extraction d'itemsets fréquents à partir de grands volumes de données.

Chapitre 4 : Testes et Evaluation

1. Introduction

Ce chapitre est consacré à la présentation des résultats de l'application de notre approche distribuée et parallèle pour d'extraction d'itemsets fréquents sur différents datasets. On présente dans un premier temps l'environnement de développement du prototype de l'approche. Ensuite, on décrit les des jeux de données utilisés dans les tests. En dernier, une discussion basés sur différentes métriques des résultats obtenus est explicitée afin d'évaluer l'efficacité de l'approche.

2. Présentation de l'environnement d'implémentation

Le développement de l'algorithme d'Optimisation basée sur le Grey-Wolf (GWO) pour l'extraction de motifs fréquent à partir de Big Data a été réalisé en deux étapes. Initialement, une version séquentielle de l'algorithme a été mise en œuvre en Python, un langage de programmation haut niveau populaire pour son expressivité et sa facilité d'utilisation dans le domaine du traitement des données. Cette première version de l'algorithme a permis de vérifier la validité de l'approche avant d'entreprendre l'implémentation parallèle et distribuée.

2.1. Environnement matériel

Les expérimentations en étais effectuer sur un Cluster d'ordinateur constituer de 1 machine maitre (master node) qui est charger de planifier et lancer les taches et récupérer diffèrent résultat durant l'exécution de l'application et 4 machines de travail (workers nodes) qui sont charge d'effectuer différentes tâches.

Les caractéristiques matérielles sont dans le tableau suivant :

1 Master node	CPU	Intel(R) Core(TM) i5-7200U CPU, 2.59GHz
	Nombre de cœurs	4
	Mémoire	16 GB
	Stockage	250 GB

Tableau 12: caractéristiques du nœud maitre.

4 workers nodes	CPU	Intel(R) Core(TM) i5-10400 CPU, 2.90GHz
	Nombre de cœurs	12
	Mémoire	4 GB
	Stockage	250 GB

Tableau 13: caractéristique des nœuds de travaille.

2.2. Environnement logiciel

Notre travail a été réalisé en utilisant PySpark, qui est l'interface Python pour Apache Spark, une plateforme populaire de calcul distribué. PySpark, avec sa gamme complète de fonctionnalités Spark, nous a permis d'exploiter la puissance du calcul distribué et de gérer efficacement de grands ensembles de données. Le langage de programmation Python, grâce à sa syntaxe claire et sa vaste bibliothèque standard, a été le choix idéal pour développer notre solution.

Pour le stockage des données d'entrée, nous avons utilisé Hadoop Distributed File System (HDFS). Il s'agit d'un système de fichiers distribué, conçu pour être déployé sur des matériels bas de gamme. HDFS fournit un accès haut débit aux données d'application et est adapté aux applications ayant de grands ensembles de données. En outre, HDFS est tolérant aux pannes et conçu pour être déployé sur des architectures matérielles à faible coût. Le tableau ci-dessous contient les versions des logiciels utilisés :

Tableau 14: Logiciels et versions.

Logiciel	**Version**
Os Ubuntu	20.04
Spark	3.4.0
HDFS	3.3.5

3. Jeux de données et paramètres expérimentaux

3.1. Présentation des datasets utilisés

Lors de l'évaluation des performances des cadres proposés, nous avons essayé de varier entre les catégories de l'ensemble de données pour examiner presque tous les cas importants afin de ne pas dépasser le temps dont nous disposons pour les tests et la validation.

Par conséquent, nous avons acquis trois bases de données de caractéristiques différentes, collectées à partir de la bibliothèque d'exploration de données open-source SPMF[**30**] et transformées en binaire pour pouvoir appliquer notre approche sur ses dataset. Les datasets utilisés sont présentés dans le tableau ci-dessous :

Dataset	**Nombre de transaction**	**Nombre d'item**	**Taille de fichier**
RecordLink	574 914	29	32.3 MB
Kddcup	1 000 000	135	258 MB
USCensus	1 000 000	396	756 MB

Tableau 15: Dataset utiliser pour lors de nos tests.

3.2. Initialisation des paramètres

Les paramètres nécessaires à l'initialisation sont : le nombre de loups dans la population, le nombre d'itérations et les vecteurs binaires des loups de la première génération ainsi que le MinSup (seuil minimal pour la fréquence).

Pour ce qui est du nombre de loups dans la population et le nombre d'itérations et le MinSup, ceux-ci sont initialisés par l'utilisateur. Nous utilisons les mêmes valeurs pour comparer le comportement de notre approche sur différents jeux de données. Lors des premiers tests, nous avons observé que l'initialisation aléatoire de la population entraînait une stagnation des valeurs de la fonction de fitness à 0 lors de 3 tests sur 5, c'est-à-dire que nous n'avions aucune solution. Nous avons donc décidé d'initialiser en utilisant les transactions du jeu de données. Par exemple, si le nombre de loups est égal à 10, l'initialisation consiste à prendre les 10 premières lignes du jeu de données.

4. Testes et Evaluation

Dans la section suivante, nous présenterons les résultats de notre approche appliquée sur quatre ensembles de données : RecordLink, Kddcup et USCensus.

Nous avons appliqué notre approche (SBGWO-FIM) sur les diffèrent jeux de données en essayant de garder les mêmes paramètres pour observer le comportement de notre approche. Nos expérimentations avaient pour but de réduire au maximum le temps d'exécution tout en maximisant la qualité des résultats obtenus.

Pour estimer la précision de nos résultats, l'algorithme Apriori a été utilisé préalablement sur les jeux de données pour obtenir le nombre précis des itemset fréquent à différents seuils. Le résultat obtenu dans un seuil en utilisant SGWO-FIM est diviser par le nombre d'item exacte obtenus à ce même seuil par Apriori, puis le pourcentage est calculé.

4.1. RecordLink :

Expérimentation 1 :

Tableau 16: Paramètres expérimentation 1.

Support	0.1
Population	50

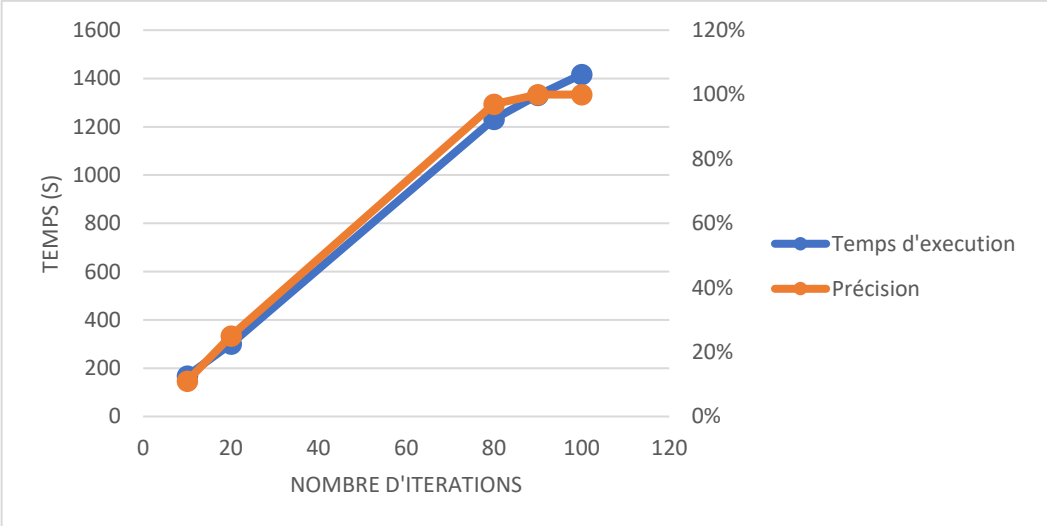

Figure 20: Graphe durée d'exécution et précision de résultats selon le nombre d'itérations expérimentation 1

Comme on peut le voir dans figure précédente, la précision et le temps d'exécution augmente avec l'augmentation du nombre d'itérations.

Expérimentation 2 :

Tableau 17: Paramètres expérimentation 2

Support	0.1
Nombre d'itérations	80

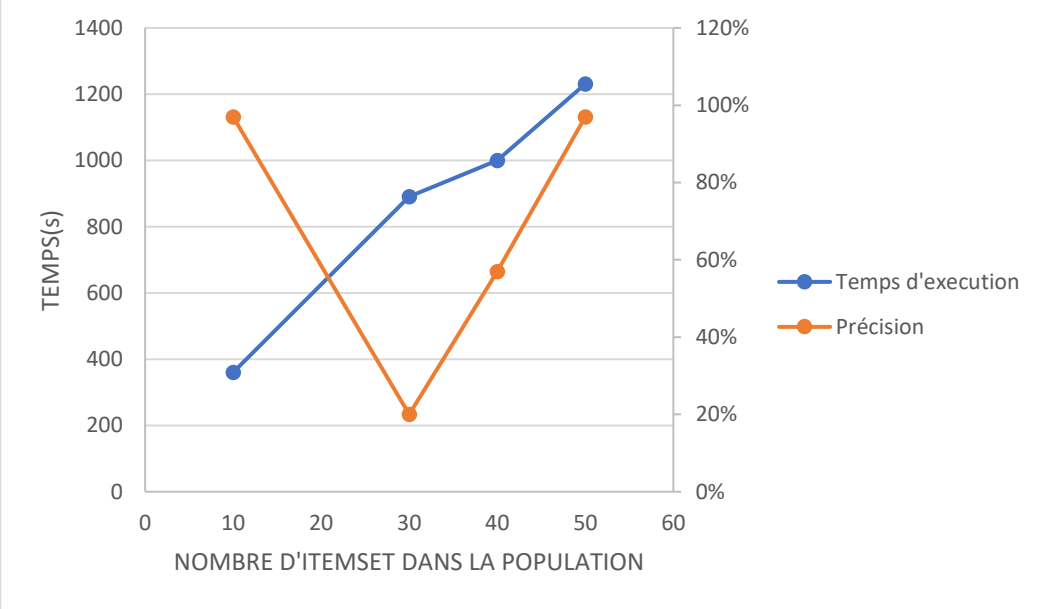

Figure 21: Graphe durée d'exécution et précision de résultats selon le nombre d'itérations Expérimentation 2

D'après la figure 21 on observe une augmentation du temps d'exécution avec l'augmentation du nombre d'itemset de la population, se qui est de la précision elle est diminue puis augment quand dépasse 30 itemset.

Expérimentation 3 :

Population	50
Nombre d'itérations	100

Tableau 18 : Paramètres Expérimentation 3

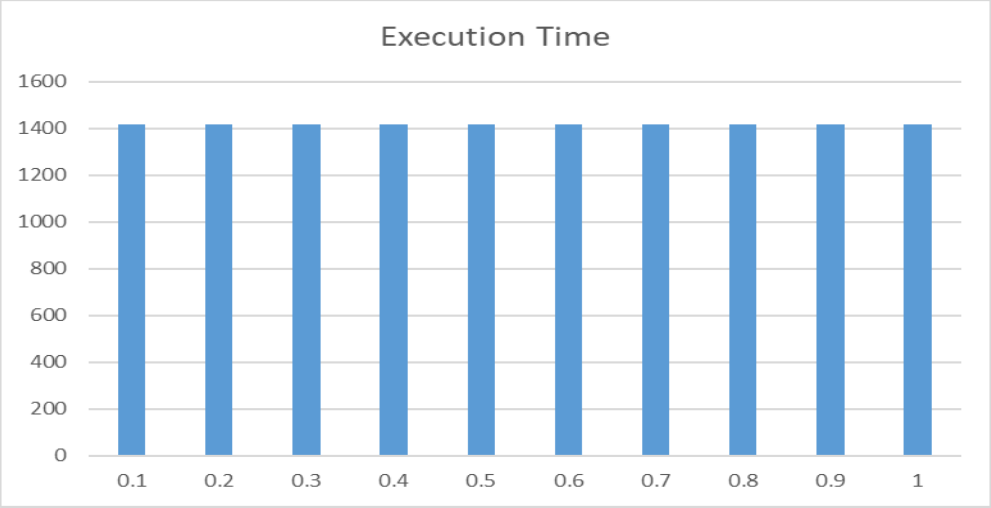

Figure 22: Histogramme calcule de support par rapport au temps d'exécution – Expérimentation 3

Comme on peut le voir dans histogramme précèdent, le temps d'exécution reste stable pour tous les supports.

4.2. Kddcup

Expérimentation 4 :

Tableau 19: Paramètres Expérimentation 4

Support	0.6
Population	10

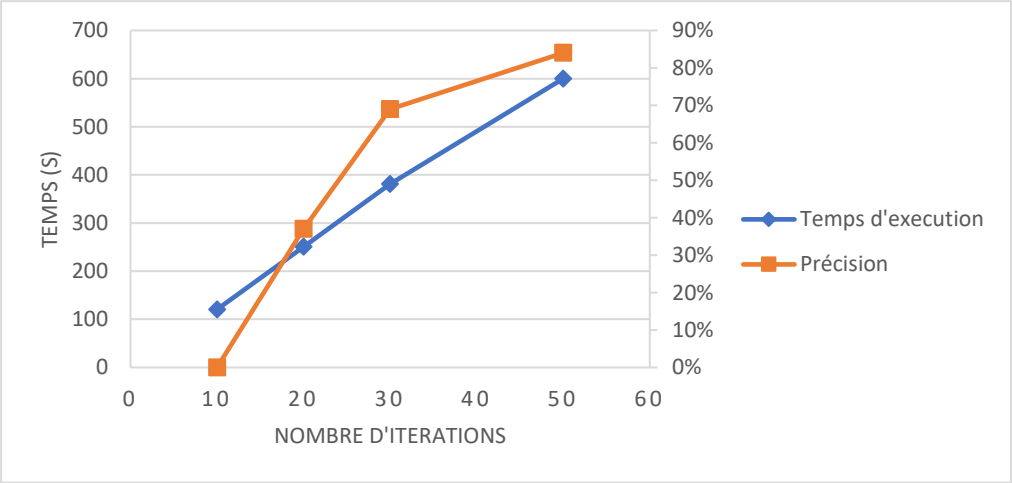

Figure 23: Graphe temps d'exécution et précision de résultats par rapport au nombre d'itérations Expérimentation 4

Comme on peut le voir dans figure précédente, la précision et le temps d'exécution augmente avec avec l'augmentation du nombre d'itérations.

Expérimentation 5 :

Support	0.6
Iteration 10	10

Tableau 20: Paramètres Expérimentation 5

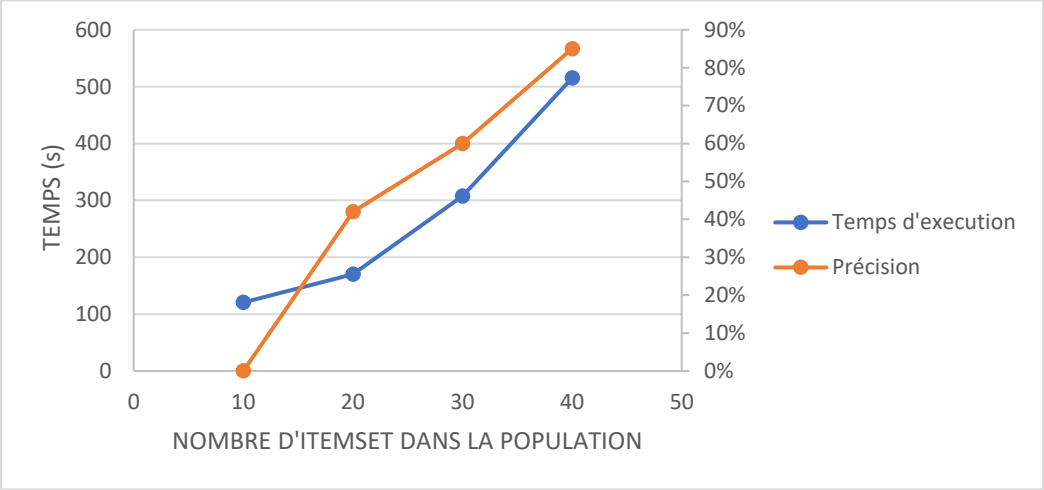

Figure 24: Graphe temps d'exécution et précision des résultats par rapport au nombre de d'itemset dans la population - Expérimentation 5

D'après la figure 24 on observer une augmentation du temps d'exécution et de la précision avec l'augmentation du nombre d'itemset dans la population.

Expérimentation 6 :

Population	50
Nombre d'itérations	100

Tableau 21: Paramètres Expérimentation 6

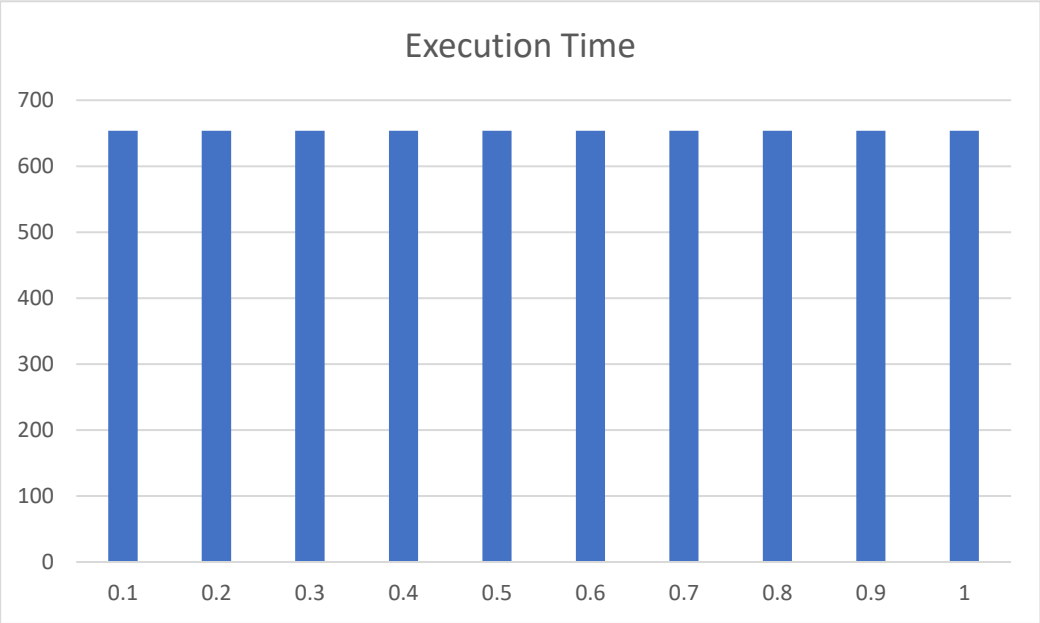

Figure 25: Histogramme calcule d'ensemble d'itemset fréquent sellent MinSup par rapport au temps d'exécution – Expérimentation 6

Comme on peut le voir dans histogramme, le temps d'exécution reste stable pour tous les Seuil minimum.

4.3. USCensus

Expérimentation 7 :

Tableau 22: Paramètres Expérimentation 7

Support	0.9
Population	10

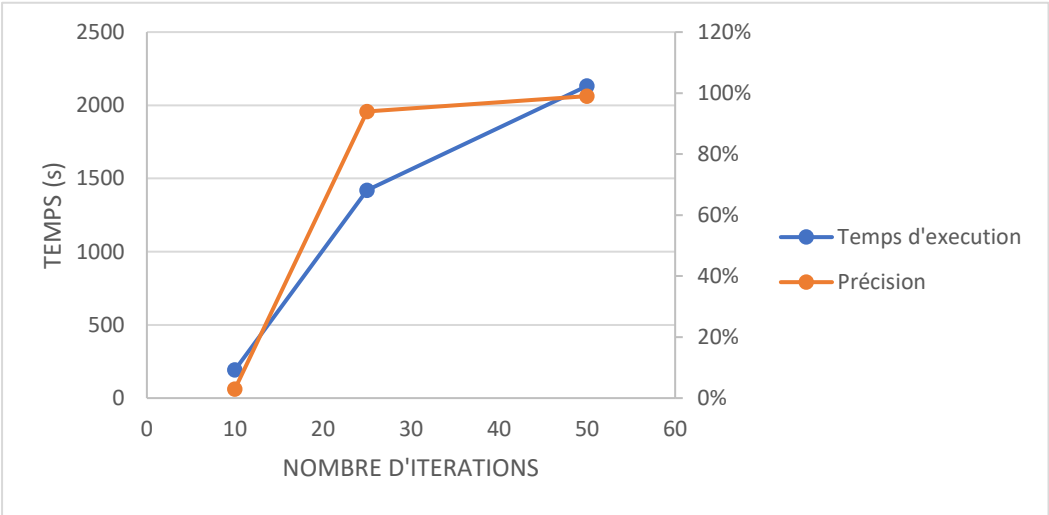

Figure 26: Graphe temps d'exécution et précision de résultats par rapport au nombre d'itérations Expérimentation 7

D’après la figure 26 on observer une augmentation du temps d’exécution et de la précision avec l’augmentation du nombre d’itérations.

Expérimentation 8 :

Tableau 23: Paramètres Expérimentation 8

Population	50
Nombre d’itérations	100

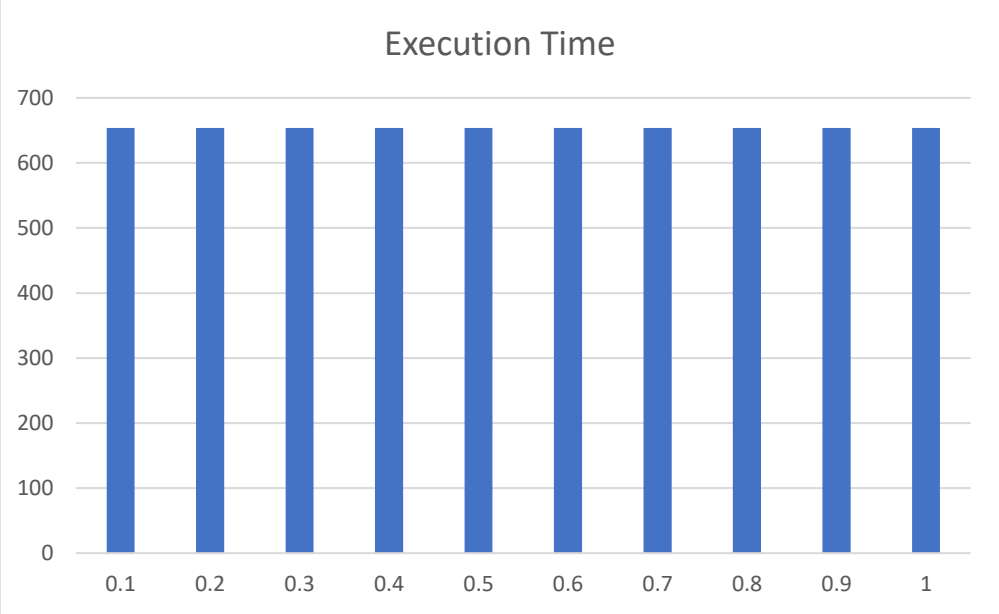

Figure 27: Histogramme calcule d’ensemble d’itemset fréquent sellent MinSup par rapport au temps d'exécution – Expérimentation 8

Comme on peut le voir dans histogramme, le temps d’exécution reste stable pour le calcul de diffèrent seuil.

Conclusion :

En conclusion, ce chapitre a été consacré à la mise en œuvre et à l'évaluation de notre approche proposée. À travers une série de tests et d'expérimentations sur différents jeux de données, nous avons évalué la performance et l'efficacité de notre méthode d'extraction d'itemsets fréquents. Les résultats obtenus mettent en avant les avantages significatifs de notre approche en termes de temps d'exécution et de qualité des motifs extraits. Ces résultats prometteurs renforcent la pertinence de notre approche dans le contexte du Big Data.

CONCLUSION GENERALE

L'objectif de notre travail était de développer une méthode approximative pour l'extraction d'itemsets fréquents à partir de données volumineuses en utilisant l'optimisation basée sur le Grey-Wolf (GWO). Plus précisément, l'objectif était de concevoir une version distribuée et parallèle de cette méthode pour assurer la prise en charge de volumes de données encore plus importants.

Pour surmonter ce défi, nous avons tout d'abord commencé par introduire et expliquer le domaine du FIM et son application sur des données certaines et incertaines, en mettant l'accent sur les algorithmes les plus couramment utilisés, et plus particulièrement le FIM dans le contexte du Big Data.

Ensuite, nous avons introduit le domaine des métaheuristiques et, plus particulièrement, le GWO, la métaheuristique choisie pour notre étude.

Les chapitres suivants étaient axés sur l'explication détaillée de notre approche et la présentation des résultats de son application sur trois ensembles de données.

Résultats expérimentaux

Suite aux expériences menées, nous concluons qu'en termes de précision, notre approche fournit des résultats satisfaisants et plus précité avec un nombre élevé d'itemsets dans la population ou un nombre élevé d'itérations. Cependant, l'augmentation de l'une de ces deux variables entraîne une hausse du temps d'exécution. Le temps d'exécution pour différentes valeurs de seuil reste le même, indiquant une indépendance de notre approche vis-à-vis du paramètre de seuil.

Limites

Nous avons rencontré deux contraintes durant ce travail : une limitation matérielle et une contrainte de temps.

En ce qui concerne la limitation matérielle notre principale limitation était la mémoire des machines Worker. Dotées de seulement 4 Go de mémoire utilisée par le système d'exploitation et les divers outils, seulement 1 Go pour chaque machine était réellement exploitable, donc une partition du fichier d’entrée ne devait pas dépasser 1Go. Alors nous n’avons pas pu tester notre approche sur des datasets plus larges.

Quant à la limitation de temps, notre contrainte résidait dans le fait que le cluster était installé dans les salles de travaux pratiques de l'université, nous limitant ainsi dans le temps alloué pour effectuer nos tests.

Travaux futurs

En ce qui concerne les perspectives d'amélioration et les travaux futurs pour notre approche plusieurs pistes peuvent être envisagées :

1. Optimisation des paramètres : L'optimisation des paramètres de l'algorithme tel que le nombre d'itemsets ou le nombre d'itération en utilisant des techniques d'apprentissage automatique (Machine Learning) est une perspective d'avenir très intéressante.
2. Hybridation avec d'autres métaheuristiques : Une autre direction pourrait être d'explorer l'hybridation de GWO avec d'autres métaheuristiques pour combiner leurs points forts.
3. Hybridation avec des approches FIM exacte : Dans un tel système hybride, GWO servirait de méta-heuristique pour guider la recherche dans l'espace des solutions possibles, tandis que la méthode exacte de FIM servirait à évaluer rapidement chaque solution potentielle. De cette manière, le calcul de fitness serait accéléré, ce qui pourrait réduire le temps d'exécution global de l'algorithme.
4. Une autre approche pourrez envisager en plus de la distribution de l'espace de rechercher la distribution d'une partie de la population sur chaque nœud et l'exécution de GWO localement dans chaque nœud, les trois meilleures solutions seront envoyés au nœud maitre pour être comparer et choix des trois meilleur résultat globaux. Cela pourrait permettre une exploration parallèle et plus efficace de l'espace de recherche, ce qui pourrait réduire le temps d'exécution.

References

[1] J. Han, J. Pei, and M. Kamber, Data Mining: Concepts and Techniques, 3rd ed. Elsevier, 2011.

[2] S. Mirjalili, S. M. Mirjalili and A. Lewis, "Grey Wolf Optimizer," in Advances in Engineering Software, vol. 69, pp. 46-61, 2014.

[3] K. Chuang, J. Huang, and M. Chen, "PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via Children–Parent Equivalence pruning," in Expert Systems with Applications, vol. 41, no. 10, pp. 4578-4591, 2014.

[4] J. Han, J. Pei, et M. Kamber, "Data Mining: Concepts and Techniques", 3rd ed., Elsevier, 2011.

[5] S. Abed, A. A. Abdelaal, M. H. Al-Shayeji, et I. Ahmad, "SAT-based and CP-based declarative approaches for Top-Rank-K closed frequent itemset mining," in International Journal of Intelligent Systems, vol. 36, 2021, pp. 112-151.

[6] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Züfle, "Probabilistic Frequent Pattern Growth for Itemset Mining in Uncertain Databases (Technical Report)," 2010

[7] Y. Tong, L. Chen, Y. Cheng, et P. S. Yu, "Mining Frequent Itemsets over Uncertain Databases," arXiv:1208.0292v1[cs], 1 Août 2012

[8] C.-H. Chee, J. Jaafar, I. A. Aziz, M. H. Hasan, et W. Yeoh, "Algorithms for frequent itemset mining: a literature review," Artificial Intelligence Review, vol. 52, no. 4, pp. 2603-2621, Déc. 2019.

[9] G. Lee et U. Yun, "A new efficient approach for mining uncertain frequent patterns using minimum data structure without false positives", Future Generation Comp. Syst., vol. 68, pp. 89-110, 2017

[10] Yazidi, J., Bouaguel, W., & Essoussi, N. (2016). A Parallel Implementation of Relief Algorithm Using Mapreduce Paradigm. In International Conference on Computational Collective Intelligence

[11] Mohamed Aymen Ben HajKacem, Chiheb-Eddine Ben N'cir, and Nadia Essoussi. "MapReduce-based k-prototypes clustering method for big data." 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2015, pp. 1-7

[12] D. Apiletti, E. Baralis, T. Cerquitelli, and P. Garza, “Frequent Itemsets Mining for Big Data: A Comparative Analysis,” *Big Data Research*, vol. 9, pp. 14-30, 2017

[13] S. Kumar and K. K. Mohbey, “A review on big data based parallel and distributed approaches of pattern mining,” Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 5, pp. 1639-1662, May 2022

[14] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, “Data mining with big data,” IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 1, pp. 97-107, Jan. 2014

[15] A. Brahmavar, H. Venkatarama, and G. Maiya, “Mining high utility itemsets with time-aware scheduling using Apache Spark,” Concurrency and Computation: Practice and Experience, Aug. 2022

[16] A. Kaveh and T. Bakhshpoori, Metaheuristics: Outlines, MATLAB Codes and Examples, 1st ed. Cham: Springer International Publishing, 2019

[17] I. Boussaïd, J. Lepagnot, and P. Siarry, “A survey on optimization metaheuristics,” Information Sciences, vol. 237, pp. 82-117, Jul. 2013

[18] K. Logeswaran, R. K. S. Andal, S. T. Ezhilmathi, A. Harshath Khan, P. Suresh, and K. R. Prasanna Kumar, “A Survey on metaheuristic nature inspired computations used for Mining of Association Rule, Frequent Itemset and High Utility Itemset,” IOP Conference Series: Materials Science and Engineering, vol. 1055, no. 1, p. 012103, Dec. 2020

[19] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989

[20] I. Rechenberg, Evolutionsstrategie: Optimierung Technischer Systeme Nach Prinzipien der Biologischen Evolution. Stuttgart: Frommann-Holzboog, 1973

[21] K. Logeswaran, R. K. S. Andal, S. T. Ezhilmathi, A. Harshath Khan, P. Suresh, and K. R. Prasanna Kumar, “A Survey on metaheuristic nature inspired computations used for Mining of Association Rule, Frequent Itemset and High Utility Itemset,” IOP Conference Series: Materials Science and Engineering, vol. 1055, no. 1, p. 012103, Dec. 2020

[22] M. Dorigo, V. Maniezzo, and A. Colorni, “The ant system: optimization by a colony of cooperating agents,” IEEE Transactions on Systems, Man, and Cybernetics – Part B, vol. 26, pp. 29-41, 1996.

[23] D. Karaboga and B. Akay, “A survey: algorithms simulating bee swarm intelligence,” Artificial Intelligence Review, vol. 31, pp. 61-85, 2009.

[24] F. Van den Bergh and A. P. Engelbrecht, “A Cooperative approach to particle swarm optimization,” IEEE Transactions on Evolutionary Computation, vol. 8, no. 3, pp. 225-239, Jun. 2004

[25] J. Kennedy and R. Eberhart, “Particle Swarm Optimization,” in Proceedings of IEEE International Conference on Neural Networks, vol. IV, 1995, pp. 1942-1948

[26] D. Karaboga, “An Idea Based On Honey Bee Swarm for Numerical Optimization,” Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[27] H. Faris, I. Aljarah, M. A. Al-Betar, and S. Mirjalili, “Grey wolf optimizer: a review of recent variants and applications,” Neural Computing and Applications, vol. 30, no. 2, pp. 413-435, Jul. 2018

[28] E. P. S. Castro, T. D. Maia, M. R. Pereira, A. A. A. Esmin, and D. A. Pereira, “Review and comparison of Apriori algorithm implementations on Hadoop-MapReduce and Spark,” The Knowledge Engineering Review, vol. 33, e9, pp. 1-25, 2018

[29] P. Singh, S. Singh, P. K. Mishra, and R. Garg, “A data structure perspective to the RDD-based Apriori algorithm on Spark,” International Journal of Information Technology, vol. 14, pp. 1585-1594, 2022

[30] P. Fournier-Viger, “SPMF: a Java Open-Source Pattern Mining Library,” [En ligne]. Disponible: http://www.philippe-fournier-viger.com/spmf/