University of Blida 1

Faculty of Sciences

Department of Informatics

Master Thesis

Option : Informatic Systems and Networks

Bioinspired Metaheuristic based Big Data Uncertain Itemset Mining Framework

---

Presented By :

Kahlia Dounia

Kheniche Ikram

In front of a jury composed of:

Ms. HIRECHE Celia	President
Ms. TEBBI Hanane	Examiner
Ms. ZAHRA Fatma Zohra	Supervisor

2021/2022

Acknowledgement

In the name of Allah, the most Beneficent and most Merciful

We thank Allah for giving us health, strength to survive, as well as the audacity to overcome all the difficulties and the will to start and complete this modest work.

We thank **Ms. ZAHRA** very warmly for having supervised us, as well as for her availability and her constructive remarks which were very useful to us throughout our project.

We thank all of our Department of Informatics teachers of the Faculty of Sciences of **BLIDA 1**.

We would like to thank the jury members for their interest in judging our work.

We also thank **Zakaria Ziraoui** and **Abdenour Aba** for their sincere contribution which helps us to complete this project.

Finally, our thanks go to everyone who has contributed, directly or indirectly away, to the culmination of this work.

We address a particularly emotional thought to our friends who have made our long years of study enjoyable.

Al Hamdulilah

". . Thank you Allah for the beautiful life that you have granted me . . "

On successfully completing my graduation, I dedicated the people dearest to me, and that is the least I can do for them, because without their encouragement I will never reach this important moment in my life.

To my father, my king **Omar**, who made all sacrifices with a smile on his face, he worked hard for me for so many years, he is my strength in difficult times, you are the one who chose this specialty for me, so the least I can do is dedicate to you this success, and wherever life takes me, I will always be thankful to you my Dad.

To my beautiful mother, my loving and selfless **Amel**, there is no one else in this world who will be as happy as you will be on this day. I appreciate how you always worked so hard to show your love for me. I want you to always know that I love you and cherish you more than life itself, and wish to become half of the person you are!

To my big sister, **Roumaissa**. Thank you sister for always watching out for me, and protecting me from others, letting me tag along even though you didn't really want to, and being simply the best big sister ever. I love you and am so glad that we are sisters.

To my little sister, **Cheyma**. Thank you sister for always being there for me when I needed you most, you have always loved me during my worst period, for your patience and understanding, for bringing so much love and laughter into the world.

To my most special aunt, **Nacera**. Thank you for all of your years of love and support. I appreciate everything you have ever done for me. You are the most supportive, reliable and most kind hearted woman.

To my binomial and my bestie, **Dounia Kahilia** is the sister of my soul, and her beautiful family.

To my lovely cousins, which are special to me, **Imen**, **Lilya**, **Meriem** and **Nesrine**.

To my dearest friends **Djalila**, **Ahlem** and **Chaimaa**.

To my lovely cats, **Oreo**, **Mchimech** and **Minou**.

And for all **Kheniche** and **Aksouh** families.

Ikram Kheniche. . .

'The first thing is to thank Allah who has enabled me to reach this stage of my life after effort and trouble and fatigue..'

I want to thank my family who has supported me throughout my life..

To my mother, **Fatiha**. You are the definition of irreplaceable, mom. Thank you for all that you sacrificed for me, and for always loving me and believing in me. I love you. I couldn't have accomplished everything I have without you. You have cared for me like no one else ever could. I will forever appreciate you.

To my late father, **Mohammed**. Thank you for giving me the most wonderful and unforgettable memories anyone could ever have. I know you will be proud of me, I can imagine how big your smile would be on my graduation day. I did it Dad..

To my lovely sister, **Amira**. You give so much and always encourage me to be my best self. Thank you for always showering me with love and support, my darling sister.

To my brothers, **Abdallah** and **Rayan**. I am the luckiest person in the whole world to have brothers like you. I am thankful to have you in my life.

And thank my sister who was not born by my mother **Ikram Kheniche** and her adorable family.

To my aunts, **Malika**, **Ahlem** and **Zahia**. I am so fortunate to have such amazing Aunts like you. Thank you for being the greatest around..

To my beautiful cousins, **Manal**, **Wissam** and **Amina**.

To **Yahia**. Thank you for your support. You are a member of my family and I really appreciate that you were always there for us.

To my best friend **Zakaria Ziraoui**.

Dounia Kahilia. . .

Abstract

Uncertain pattern mining is considered as an NP-Hard problem due to its complexity and its execution time consummation. The problem is amplified in the Big Data era. Thus, we need to use techniques that don't require prior knowledge of the search space as the metaheuristics algorithms, which use natural theories based on randomness.

This work deals with the uncertainty of data when extracting frequent patterns from big uncertain (probabilistic) Datasets (BDUPM for Big Data Uncertain Pattern Mining). In addition to that, the BDUPM task is addressed as a combinatorial optimization problem in this study. In fact, we proposed three metaheuristic-based algorithms that are inspired from the Particle Swarm Optimization (PSO), Bee Swarm Optimization (BSO) and Genetic Algorithms (GA), for the purpose of extracting unexpected useful frequent patterns that help to get useful pieces of information to make trusted decisions.

The proposed algorithms MRPSO-UFIM, MRBSO-UFIM and MRGA-UFIM are employed with the MapReduce programming model in a parallel and distributed environment, and examined based on the number of frequent itemsets retrieved, and computational time. The experiments have shown the efficiency of our proposed solutions when tested with several uncertain datasets.

Key words : Frequent Pattern Mining, Uncertain Data, Big Data, Particle Swarm Optimization, Genetic Algorithms, Bee Swarm Optimization.

Résumé

L’extraction de motifs incertains est considérée comme un problème NP-Hard, en raison de sa complexité et de sa consommation de temps d’exécution. Le problème est amplifié à l’ère des Big Data. Ainsi, nous devons utiliser des techniques qui ne nécessitent pas de connaissance préalable de l'espace de recherche comme les algorithmes métaheuristiques, qui utilisent des théories naturelles basées sur le hasard.

Ce travail traite l’incertitude des données lors de l’extraction de motifs fréquents à partir de grands ensembles de données incertaines (probabilistes) (BDUPM pour Big Data Uncertain Pattern Mining). En outre, la tâche BDUPM est abordée comme un problème d’optimisation combinatoire dans cette étude. En fait, nous avons proposé trois algorithmes métaheuristiques inspirés de l’optimisation des essaims de particules (PSO), de l’optimisation des essaims d’abeilles (BSO) et des algorithmes génétiques (GA), dans le but d'extraire des modèles utiles inattendus qui aident à obtenir des informations utiles pour faire des décisions fiables.

Les algorithmes proposés MRPSO-UFIM, MRBSO-UFIM et MRGA-UFIM sont utilisés avec le modèle de programmation MapReduce dans un environnement parallèle et distribué, et examinés en fonction du nombre d'itemsets fréquents récupérés et du temps de calcul. Les expérimentations ont montré l’efficacité de nos solutions proposées lorsqu’elles sont testées avec plusieurs ensembles de données incertains.

Mots-clés : Extraction des motifs fréquents, Données incertaines, Big Data, Optimisation des essaims de particules, Algorithmes génétiques, Optimisation des essaims d’abeilles.

ملخص

يعتبر التعدين الغير المؤكد مشكلة صعبة بسبب تعقيده واستهلاك وقت تنفيذه. حيث تتضخم المشكلة في عصر البيانات الضخمة. وبالتالي، نحتاج إلى استخدام تقنيات لا تتطلب معرفة مسبقة بمساحة البحث كخوارزميات ميتاهيرستيك، والتي تستخدم النظريات الطبيعية القائمة على العشوائية.

يتعامل هذا العمل مع عدم اليقين في البيانات عند استخراج الأنماط المتكررة من مجموعات البيانات الكبيرة غير المؤكدة (الاحتمالية) (BDUPM لتعدين الأنماط الغير المؤكدة للبيانات الضخمة). بالإضافة إلى ذلك، يتم تناول مهمة BDUPM كمشكلة تحسين إندماجي. حيث اقترحنا ثلاث خوارزميات قائمة على ميتاهيرستيك مستوحاة من تحسين سرب الطيور (PSO) وتحسين سرب النحل (BSO) والخوارزميات الجينية (GA), لغرض استخراج أنماط مفيدة غير متوقعة تساعد في الحصول على الأجزاء المفيدة من المعلومات المقدمة لإستخراج قرارات موثوقة.

تم استخدام الخوارزميات المقترحة MRPSO-UFIM و MRGA-UFIM و MRBSO-UFIM مع نموذج البرمجة MapReduce في بيئة متوازية و موزعة ، وتم فحصها بناءً على عدد العناصر المتكررة التي تم استخراجها والوقت المستغرق. أظهرت التجارب كفاءة حلولنا المقترحة المختبرة على بيانات ضخمة ,مختلفة, غير مؤكدة و ومتعددة.

كلمات مفتاحية : تعدين الأنماط المتكررة، البيانات غير المؤكدة، البيانات الضخمة، تحسين سرب الطيور، الخوارزميات الجينية، تحسين سرب النحل.

Contents

List Of abbreviations

ARM : Association Rule Mining.

BSO : Bee Swarm Optimization.

EA : Evolutionary Algorithms.

ECA : Evolutionary Computation Algorithm.

Eclat : Equivalence Class Transformation.

EFIM : Efficient Frequent Itemset Mining.

EP : Evolutionary Programming.

ES : Evolutionary Strategies.

expSup : Expected Support.

FIM : Frequent Itemsets Mining.

FP-growth : Frequent Pattern growth.

FPM : Frequent pattern Mining.

FP-tree : Frequent Pattern tree.

GA : Genetic Algorithm.

Gbest : Global Best.

GP : Genetic Programming.

GPU : Graphic Processing Units.

HDFS : Hadoop Distributed File System.

HEWI-UApriori : High Expected Weighted Itemset Uncertain Apriori.

HPI : High Potential Itemset.

HTWPUI : Transaction Weighted Probabilistic and Utilization Itemset.

HUI : High Utility Itemset.

HUI-Mine : High Utility Itemset Miner.

HUPM : High Utility Pattern Mining.

HUPs : High Utility Patterns.

IDE : Integrated Development Environment.

JDK : Java Development Kit.

LGS : Local-trimming, Global-pruning and Single-pass patch-up.

MBP : Multi Back Propagation.

minSup : Minimum Support.

MR : MapReduce.

MRBSO-UFIM : MapReduce Bee Swarm Optimization Uncertain Frequent Itemset Mining.

MRExpSup : MapReduce Expected Support.

MR-Growth : MapReduce Growth.

MRGA-UFIM : MapReduce Genetic Algorithm Uncertain Frequent Itemset Mining.

MRPSO-UFIM : MapReduce Particle Swarm Optimization Uncertain Frequent Itemset Mining.

MR-PUFGrowth : MapReduce Uncertain Frequent Pattern Growth.

MR-UV-Eclat : MapReduce Uncertain Vertical Equivalence Class Transformation.

NP-hard : Non deterministic Polynomial time hardness.

Pbest: Personal Best.

PbM : Population based Metaheuristics.

PFIs : Probabilistic Frequent itemsets.

PHUI-List : Potential High Utility Itemset List.

PHUIM : Potential High Utility Itemset Mining.

PHUI-UP : Potential High Utility Itemset Mining Upper bound based mining.

PNDUA : Parallel Normal Distribution Based UApriori.

PSO : Particle Swarm Optimization.

PuFIM : Parallel Uncertain Frequent Itemset Mining.

PUF-Tree : Uncertain Frequent Pattern Tree.

RAM : Random access memory.

RDD : Resilient Distributed Dataset.

SI : Swarm Intelligence.

Tidset : Transaction Identifiers set Table.

TWPUDC : Transaction Weighted Probabilistic and Utilization Downward Closure.

TWU : Transaction Weighted Utility.

UApriori : Uncertain Apriori.

UF-growth : Uncertain Frequent Pattern growth.

UFIM : Uncertain Frequent Itemset Mining.

UFPM : Uncertain Frequent Pattern Mining.

UF-tree : Uncertain Frequent Pattern tree.

UF-Trees : Uncertain frequent Trees.

UHUPM : Uncertain High Utility Pattern Mining.

UP-Growth : Utility Pattern Growth.

Utidset : Uncertain Transaction Identifiers set Table.

UV-Éclat : Uncertain Vertical Equivalence Class Transformation.

List Of Figures

List Of Tables

GENERAL INTRODUCTION

Nowadays, since we are in the information age, all domains of life such as big companies, health sectors, public or financial utilities …etc, necessitate to receive, store, retrieve, transmit, and manipulate a huge amount of clients rich information, they may be structured, semi-structured or

unstructured, and these collections are combined together to constitute massive imperfect data, also known as the Big Data.

One of the concepts of Big Data is the veracity that deals with the uncertainty of data via generally the probabilistic theory where the possibilistic basis handles data imprecision. On the other hand, evidence theory can treat uncertainty, imprecision and incompleteness of data together. All of the mentioned concepts make Big Data something too large to be analyzed by humans. Best reduced by ex-McKinsey consultant Allen Bonde, “Big data is about machines, while small data is about people”.

To process and explore these huge amount of data, we need to use one of the important and well known Data Mining tasks, which is the Frequent Pattern Mining technology(FPM), it is a process for extracting valuable and pertinent information from datasets to generate novel, non-trivial, implicit and interesting patterns that help in predicting decisions and strategies depending on a user specified threshold taking a pattern as an itemset, it is a necessary data analytics concept that helps the other data mining tasks like clustering, indexing, classification…etc, but we need to keep in consideration its complications, where we cannot use the traditional FPM techniques to address Big Data FPM, as a reason to:

- The immensity of the datasets volume, high level of velocity and variety.
- The size of the output of frequent itemsets mining algorithms which is larger than the input in many cases.
- The uncertainty of data which is an inherent property in various emerging applications such as geolocation-based services, sensor-based, monitoring systems and data integration.
- The huge count of items required to represent an object in many domains which means an exponential size of itemsets search space.
- The need for low frequency threshold itemsets in Big Data.

Therefore, these difficulties attracted researchers to develop a specific exact and traditional approximation algorithms using one of the most Big Data paradigms, the MapReduce to be able to address one or more of the aforementioned issues but not all, and they still suffer in many terms, the larger the data, the more complex the solution space, they becomes slower and more difficult to validate, especially the multiple scans of the data, consequently to that, the FIM problematic become an NP-Hard problem and to solve it we need more non traditional techniques and strategies

to boost up the performance of the mining process for large, uncertain data sets by transforming this issue into a combinatorial optimization problem to search for the best and optimal resolution out of all possible solutions.

We need to always keep in mind that Big Data volume grows with time, that makes this optimization problem inconsistent. It is not possible to use heuristics to handle it, so a different optimization approach is employed, known as metaheuristics, where their global idea was extracted from our various natural elements such as evolutionary or swarm intelligence procedures.

In our case, all the previously mentioned methodologies and drawbacks inspired us to take on the Big Data challenge, and to address the probability theory that means we will deal with Uncertain Frequent Pattern Mining (UFPM) Big Data. Where we didn't employ just one method, but, we combined several ideas that help us to propose three efficient frameworks that perform the UFPM task for the purpose of maximization, our contributions is as follows:

The first proposed algorithm was the MRPSO-UFIM that refer to MapReduce Particle Swarm Optimization framework for Uncertain Frequent Itemsets Mining, the second one is the MRBSO-UFIM that means MapReduce Bee Swarm Optimization algorithm for Uncertain Frequent Itemsets Mining, and lastly the MRGA-UFIM that is identified as MapReduce Genetic Algorithm for Uncertain Frequent Itemsets Mining. These frameworks are based on three well-known metaheuristics PSO, BSO and GA algorithms with more enhancements, in a distributed Hadoop environment via a modified MapReduce echnique that also includes an extra couple of parallelization models.

The rest of this thesis is divided into five chapters and structured as follows:

Chapter I : Uncertain frequent pattern mining

This chapter defines the uncertain data based on the probabilistic theory and reviews the existing frequent and high utility pattern mining algorithms from uncertain datasets.

Chapter II : Big Data Uncertain pattern mining

This chapter is mostly a survey that introduces the uncertain Big Data and its different available paradigms, it also expresses the algorithms suggested by researchers, which work with one or more presented paradigms, followed by a simple comparison between them.

Chapter III : Metaheuristics

This chapter is related to the natural phenomena technique, the metaheuristics, where it details its methods, and explains the three based metaheuristic algorithms.

CHAPTER IV : THE PROPOSED SOLUTION

This chapter describes our proposed solution, it summarizes the steps that we have followed to design our frameworks, with a detailed explanation of all the implemented ideas.

CHAPTER V : TEST AND VALIDATION

In this chapter we present our tests and validation of the proposed solution and report the obtained results through performance tables and plots.

Chapter I:

Uncertain Itemsets Mining

I. Introduction

Many existing data mining algorithms search interesting patterns from transactional databases of precise data. However, with the rapid development of data acquisition and data processing technologies, various forms of complex data have emerged, like uncertain data. Thus, in many applications, the existence of an item in a transaction is best captured by a likelihood measure or a probability, because we are living in an uncertain world, in which uncertain data can be found almost everywhere.

In this chapter, we review some basic concepts and rules with the recent algorithmic development on mining such itemsets from data with the two mining techniques: frequent itemset mining, and high utility itemset mining. First, we gave a brief comprehension about precise data and their frequent pattern mining algorithms, the Apriori, the FP-growth, the Eclat algorithms, and their high utility pattern mining algorithms, the HUI-Miner, and the Utility pattern growth algorithms. Since our paper focus on uncertainty, therefore, we presented the uncertain data and discussed the frequent pattern mining algorithms from uncertain data proposed by researchers, the UApriori, the UF-Growth, the UEclat algorithms, and the algorithms for mining potential high utility itemsets, the PHUI-UP, the PHUI-List algorithms.

II. Precise Data Mining

A. Precise Data

Precise, certain, or deterministic data is the traditional type of data in which the presence and the absence of items in transactions of a database is certain, an example of this type of database is illustrated in *Figure.1* **(Chauhan, 2019)** which describes lines as transactions and columns as the total items, where the existence of an item in a transaction is 1 else is 0.

Transaction ID	Grapes	Apple	Mango	Orange
1	1	1	1	1
2	1	0	1	1
3	0	0	1	1
4	0	1	0	0
5	1	1	1	1
6	1	1	0	1

Figure.1: Deterministic basket market database table.

To mine patterns from precise databases, researchers were interested in two well known data mining tasks which are the frequent pattern mining and the high utility pattern mining.

B. Frequent Pattern Mining from Precise Data

1. Definition

Frequent pattern mining (FPM) is a fundamental data mining task that aims to discover implicit, previously unknown and potentially useful information and valuable knowledge in terms of sets of frequently occurring sets of items **(Aggarwal & Han, 2014**), objects or events. Most frequent pattern mining algorithms find patterns from traditional transaction databases, in which the content of each transaction of items is definitely known and precise.

2. Frequent Pattern Mining Algorithms

2.1. Apriori Algorithm

Apriori (**Agrawal & Srikant, 1994)**, is an algorithm that mines precise static databases. Basically it can be described recursively in level-wise search technique. Let's give a summary, the Apriori algorithm comprises two steps as presented in *Figure.2* **(Jain, 2017)**, which are ‘join’ and ‘prune’ operations, which are repeated over and over again. First, it generates candidates then checks the support values from the occurrences of those candidates within the database. A minimum support threshold *minSup* is defined before the algorithm execution. Therefore, all candidates with a support value that is greater than or equal to the *minSup* will be returned as frequent patterns. According to **MacKinnon (2015)**, one of the big Apriori-based algorithm challenges is that the candidate generation process is

a bottleneck of all Apriori-based algorithms, which is not present in the tree-based algorithm FP-Growth presented below.

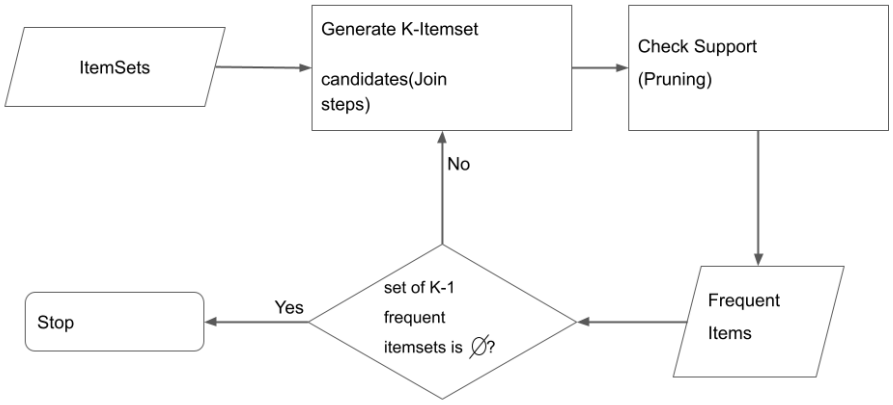


Figure.2: The Apriori Algorithm process.

2.2. FP-Growth Algorithm

Frequent Pattern Growth (FP-Growth) (**Han et al., 2000)**, is an algorithm that mines frequent itemsets without a costly candidate generation process. It implements a divide-and-conquer technique steps implemented recursively as illustrated by **Zhang (2021**) in *Figure.*3. First, generating a descendingly ordered list of frequency of frequent items from the database scan, then compressing the database into a frequent-pattern tree(FP-tree) using the frequency-descending list. This tree is mined by starting from each initial suffix pattern, constructing its conditional pattern base then constructing its conditional FP-tree, and performing mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

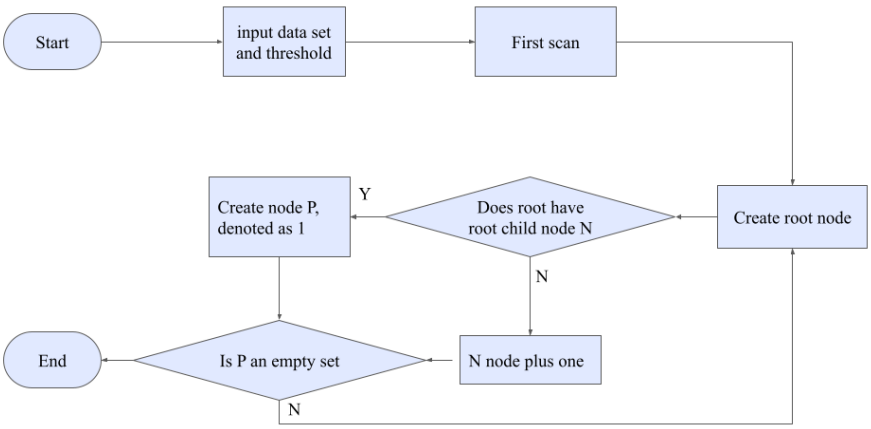


***Figure.3*:** The FP-Growth Algorithm process.

2.3. Eclat Algorithm

Equivalence Class Transformation (Eclat) **(Zaki, 2000)**, is a depth-first algorithm that mines frequent itemsets efficiently. The algorithms introduced above all mine frequent patterns from the transaction set of the itemsets, and they support the horizontal data formats. There are many cases where the second database format is vertical, showing more efficiency than the first format. The two data formats as **Foscari et al., (2004)** presented them in *Figure.4*.

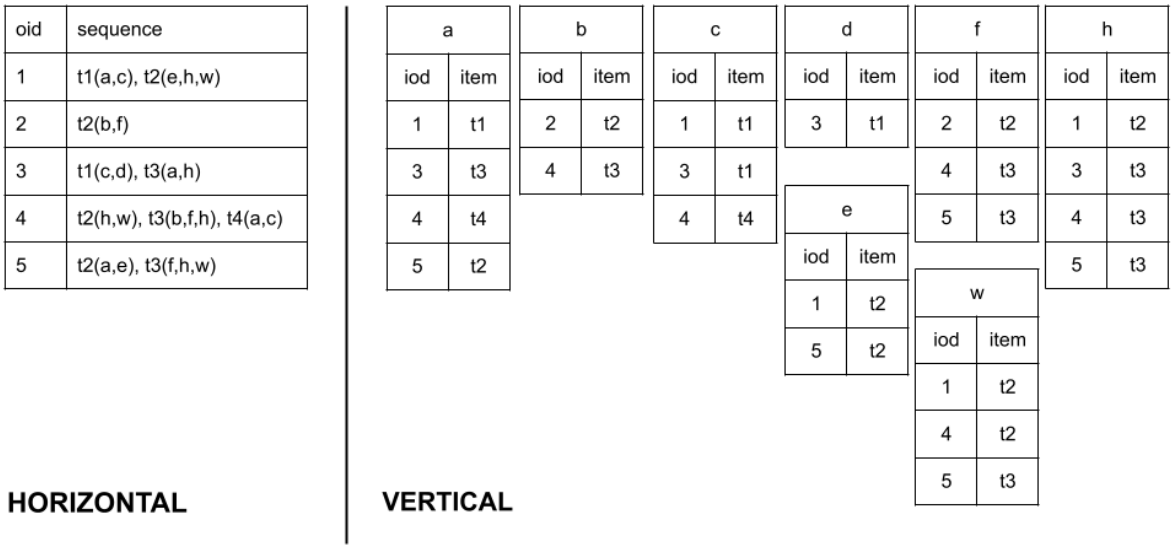


Figure.4: Horizontal and vertical representation of data.

Thus, *Figure.5* illustrates the algorithm process as defined by **Reynaldo & Boy Tonara, (2018)**, we can see that it uses an inverted table to increase the speed of frequent item set generation. It is started by listing the Transaction ID set of each item, then filtering with *minSup* and computing the Transaction ID set of each item pair to filter out the pairs that do not reach the *minSup*, this process repeats until all the frequent itemsets are intersected with one another and no frequent itemsets can be found. However, **Solanki & Soni, (2015)** finds that the Eclat disadvantage is that the deletion of the candidate set is not performed, resulting in a large number of candidate sets, which affects the performance of the algorithm.

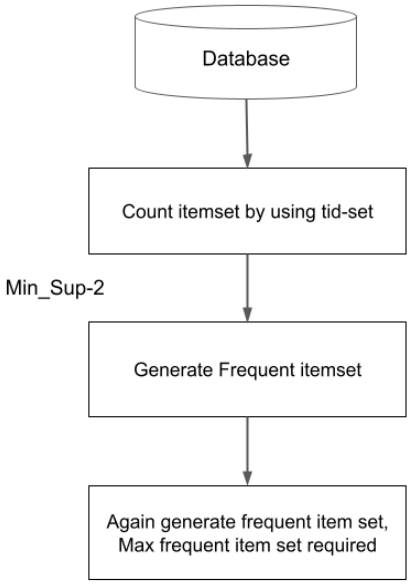
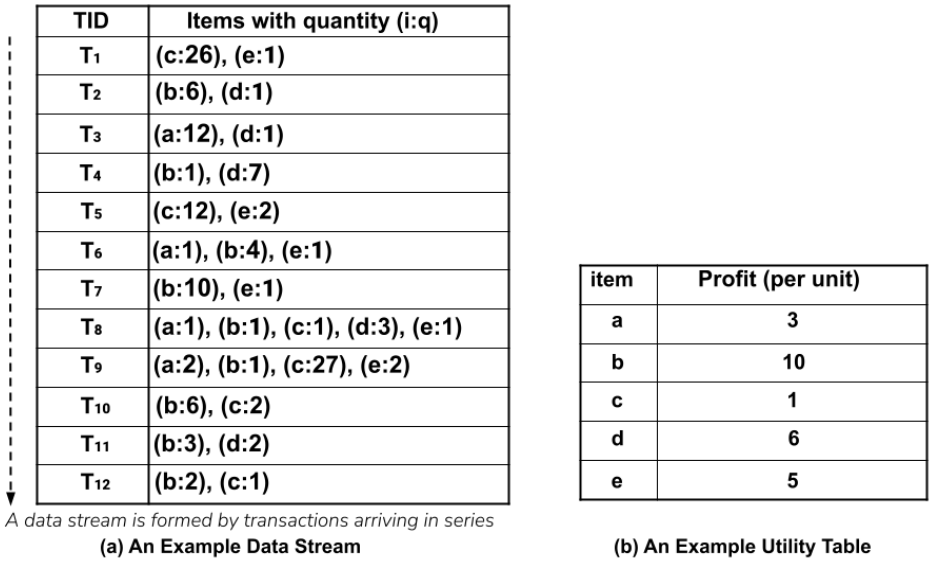


Figure.5: The Eclat Algorithm process.

C. High Utility Pattern Mining from Precise Data

1. Definition

High utility pattern mining **(Fournier-Viger et al., 2019)** is a challenge to find meaningful information from massive amounts of data. Many studies have focused on traditional frequent pattern mining and just concern the occurrence of itemsets, patterns in the database, without considering the internal utility values (i.e., quantity). High utility pattern mining is an emerging data science task, which consists of discovering patterns having a high importance in databases. The utility of a pattern can be measured in terms of various objective criteria such as its profit, frequency, and weight, **Li et al., (2008)** gives an overview about the high utility pattern mining from a precise database as shown in *Figure.6*.



TID	Items with quantity (i:q)
T_1	(c:26), (e:1)
T_2	(b:6), (d:1)
T_3	(a:12), (d:1)
T_4	(b:1), (d:7)
T_5	(c:12), (e:2)
T_6	(a:1), (b:4), (e:1)
T_7	(b:10), (e:1)
T_8	(a:1), (b:1), (c:1), (d:3), (e:1)
T_9	(a:2), (b:1), (c:27), (e:2)
T_{10}	(b:6), (c:2)
T_{11}	(b:3), (d:2)
T_{12}	(b:2), (c:1)

A data stream is formed by transactions arriving in series

(a) An Example Data Stream

item	Profit (per unit)
a	3
b	10
c	1
d	6
e	5

(b) An Example Utility Table

Figure.6: High utility deterministic database table

2. Efficient Algorithms for High-Utility Itemset Mining

Efficient High-Utility Itemset Mining (EFIM) **(Zida et al., 2015)** is a novel algorithm ,which introduces several new ideas to more efficiently discover high-utility itemsets both in terms of execution time and memory. EFIM relies on two upper-bounds named sub-tree utility and local utility to more effectively prune the search space. It also introduces a novel array-based utility counting technique named Fast Utility Counting to calculate these upper-bounds in linear time and space. Moreover, to reduce the cost of database scans, EFIM proposes efficient database projection and transaction merging techniques.

2.1. HUI-Miner Algorithm

The HUI-Miner algorithm **(Liu & Qu, 2012)**, uses a novel *utility-list* structure to store both utility information about itemsets and heuristic information for search space pruning. Whereas, the *utility-list* of items allows directly deriving the utility-lists of other itemsets and calculating their utilities without scanning the database. HUI-Miner avoids candidate generation, it can efficiently mine high utility itemsets to further speed up the construction of *utility-lists.*

2.2. Utility Pattern Growth Algorithm

The Utility Pattern Growth algorithm (UP-Growth) **(Tseng et al., 2010)**, maintains the information of high utility itemsets in a special data structure named UP-Tree that refers

to utility pattern trees such that the candidate itemsets can be generated efficiently with only two scans of the database. **Tseng et al., (2010)** evaluated UP-Growth in comparison with the state-of-the-art algorithms on different types of datasets, they found that UP-Growth not only reduces the number of candidates effectively but also outperforms other algorithms substantially in terms of execution time, especially when the database contains lots of long transactions.

III. Uncertain Pattern Mining

A. Uncertain Data

Many of the real world applications may have not only certain data but also various types of uncertain data, generally most of the collected data is uncertain, imprecise and incomplete **(Hariri et al,. 2019),** which refers to unknown or imperfect information. This uncertainty in data comes from many different sources, such as the environmental conditions and issues related to data collection and sampling, also in the multimodality (e.g., the complexity and noise introduced with patient health records from multiple sensors include numerical, textual, and image data).

Additionally, uncertainty is one of the defining characteristics of data which means that the data may contain hidden relationships, Therefore, the processing using uncertain data may negatively impact the effectiveness and accuracy of the results data mining process. In another manner, if training data is incomplete, or obtained through inaccurate sampling, after training the learning algorithm using corrupted training data will likely output inaccurate results. In computer science, uncertainty deviates the data information from the correct, intended or original values.

Previously, we listed two tasks for itemsets mining with precise data, but in this case, the challenge is different with uncertain data. Well, it is not an easy task to mine uncertainty in real datasets, especially when the data may have been collected in a manner that creates bias and items within each transaction of the database have their own probability values not binary, instead of exact existence or nonexistence information. Thus, to remove the many forms of uncertainty that exist in data like the weather data example defined by **Lee et al., (2015)** that is shown in *Figure.7*, not to zero, but to mitigate it, there are many theories and techniques that have been developed to model its various forms. We will describe below two common itemsets mining techniques and their Algorithms: Frequent pattern mining and High Utility pattern mining.

City	Weather				
	Severe heat	Deluge	Dense fog	Windstorm	Thunder and lightning
A	70%	10%	20%	10%	5%
B	30%	80%	50%	10%	60%
C	40%	70%	30%	90%	70%
D	50%	20%	60%	10%	10%

Weather	Accuracy
Sever heat	0.8
Deluge	0.6
Dense fog	0.7
Windstorm	0.4
Thunder and Lightning	0.3

Figure.7: Uncertain weather database and accuracy information.

B. Frequent Pattern Mining from Uncertain Data

1. Definition

Uncertain Frequent Pattern Mining (UFPM) **(Chui et al., 2007)** has the same objective as the FPM strategy, which is a task that finds valid patterns (itemsets) from uncertain databases such as the left side of *Figure.7*.

In uncertain data, the transactions are mostly probabilistic, where to determine frequent itemsets we need to count the probability of presence of an itemset in a given transaction, as it is not possible to definitively count the frequency of itemsets.

In this section, we are interested in the **Chui et al., (2007)** expected support-based model to mine frequent patterns. Therefore, we will present some basic concepts related to existential probability and expected support introduced by **Chui et al., (2007)**. Let's get *S,* a set that contains *m* items of a dataset (*m* : the total number of items), and *X* describes a pattern with *X={x1,x2,...,xn}* '*n-itemset'*, considering a pattern is composed of *n* items, where $X \subseteq S$ and $1 \leq n \leq m$, we have a transactional data *D={t1,t2,...,tk};* which is a set of *k-transactions,* think about *tj* which is a transaction from *D,* where each transaction $tj \subseteq D$, where all transactions containing *X* are collected in the obtained set *D*. Contrary to precise data, each item *xi* from *X* in a transaction *tj={x1,x2,...,xp}*; in uncertain data is associated with an existential probability value *P(xi, tj)*, which is equal to the possibility of the presence of *xi* in *tj.* An additional note that the probability of an item lies between *0* and *1* as shown in *Rule.1*.

Rule.1: $0<P(xi, tj)\leq 1$

When the items within *X* are independent, then the product of the existential probability values corresponding to the items in the *n-itemset X* is the existential probability

P(X, tj) for the pattern *X* in *tj*. Therefore, we have the following relationship presented in *Rule.2*:

Rule.2: $$P(X, tj) = \prod_{x \in X} P(x, tj)$$

In precise data, the support is the number of occurrences of the pattern, but with uncertainty, the support needs to be expected, and it can only be counted in probabilistic value, which is the sum of the existential probabilities of an itemset. For better understanding, suppose *X* is a pattern, let *exSup(X)* be the support of this itemset *X* in the database is the sum of *P(X, tj)* over all *k* transactions in the database, which is defined below in *Rule.(3)*:

Rule.3: $$exSup(X) = \sum_{j=1}^{k} P(X, tj) = \sum_{j=1}^{k} \left(\prod_{x \in X} P(x, tj) \right)$$

According to **Chui et al., (2007)**, a pattern *X* is frequent on the condition that $exSup(X) \geq minSup$, where the expected support of *X* is larger than the defined threshold *minSup* which refers to the user specified minimum support threshold. Considering we have a probabilistic database and a user defined support value, the challenge is to extract an efficient, complete and definite set of frequent patterns from this uncertain data. Therefore, in the next section, we will introduce the UFPM algorithms, which can satisfy the condition $expSup \geq minSup$.

2. Frequent Pattern Mining Algorithms

2.1. UApriori Algorithm

UApriori is the first based level-wise algorithm with a candidate generate-and-test paradigm proposed by **Chui et al., (2007)** that extracts frequent itemsets from uncertain databases. It is a modification of the classical Apriori identified above. Where, it depends on the Apriori property, presented by **Agrawal & Srikant, (1994)** which known as the downward closure property as defined in ***property.1*** below:

Property.1: *"If a pattern I is frequent in expected support, then all the non-empty subsets of the pattern are also frequent in expected support."*

Property.2: *"If any subset sb of an itemset S is not frequent, then the superset S itself cannot be a frequent itemset."*

Equivalently, from *property.1* in a reverse manner we can extract *property.2*. For the uncertain problem, researchers **Agrawal & Srikant, (1994)** proved that we can use the downward closure principle without a peril of losing true frequent patterns. Thus, according to *Rule.2*, the existential probability of X is lower than or equal to his subsets, since we are multiplying probabilistic values, imply that every subset X^{k-1} of X^{k} , where k is the total number of items within X, for any transaction T, must satisfy this condition shown in *Rule.4*:

Rule.4: $p(X^{k-1}, T) \geq p(X^{k}, T)$

Since the existential probability is a product function, the superset existential probability gets smaller as we increase the multiplied subsets probabilities. Therefore, the expected support of the subset is larger than the expected support of the superset pattern as presented in *Rule.5*:

Rule.5: $exSup(X^{k-1}) \geq exSup(X^{k})$

First, UApriori scans the database to get the expected support (*expSup*) of each *1-itemset*, this *expSup* of *1-itemset* is compared with the minimum support (*minSup*) to extract the frequent *1-itemset*. From the *1-itemset* it generates the *2-itemset* and prunes the non-frequent itemset using the frequency condition. The database is scanned once again to count the exSup of candidate *2-itemset*, and apply the frequency condition. Similarly, the procedure is repeated until no more frequent item is generated.

As it is known, there is always a difficulty while dealing with uncertain databases (UDBs). Therefore, even the UApriori, UDBs exceeded its capabilities, well, he suffered from the inherent problem from the original Apriori, which is handling large datasets, as long as the algorithm approach is level-wise generate-and-test framework and it requires multiple scans of UDBs, because the same pattern with different support values are considered as different candidates.

The new complication was while dealing with very small existential probabilities of most items of a pattern to generate the expected support in the pruning step, which ended up to be even worse than that of the original Apriori because of the effect of multiplying small

numbers several times. **Chui et al., (2007)** tried to remedy these problems by proposing a trimming strategy called LGS (Local-trimming, Global-pruning and Single-pass patch-up), to reduce the database by removing items with low probability. It is still Apriori based, but this improvement helps reduce the number of candidate patterns being counted during the mining process.

2.2. UF-Growth Algorithm

UF-Growth is a well known uncertain itemset mining and a tree-based algorithm **(Leung et al., 2007)**, which follows the basic framework of the FP-growth algorithm explained in the previous section, during the mining operation, the algorithm does not generate any candidate patterns unlike the level-wise method UApriori. Thus, it relies on two essential operations: the construction of UF-trees and the mining of frequent patterns from UF-trees. Hence, UF-growth and FP-growth share the same approach, which is pattern growth approach, they construct their trees in two scans of the input database, but UF-growth does not use the FP-tree. It uses another tree data structure called UF-tree in which each node contains three parameters: the item, its existential probability, and its occurrence count in the path. The UF-tree is constructed in the same manner of FP-tree, taking account the following properties defined by **Leung et al., (2007)**:

Property.3: *"A new transaction is merged with a child node only if the same item and the same existential probability exist in both the transaction and the child node."*

Since the data is probabilistic the ***Rule.3*** is used to count the expected support with some adaptation, as indicated down in ***Property.4***:

Property.4: *"The expected support of X is the sum of the product of the occurrence count and existential probability of every item within X."*

Secondary, as the UF-tree is constructed, to mine frequent patterns from it, the UF-growth used a recursive algorithm to break the tree into smaller pieces with a similar way as the FP-growth algorithm, but also with some modification, which is listed down in *property.5*:

Property.5: *"When forming a UF-tree for the projected database for a pattern X, we need to keep track of the expected support (in addition to the occurrence) of X."*

According to *property.4,* **Tong et al., (2012)** made a comparison and showed that UApriori outperformed UF-growth, because there is a small percentage for paths to share the same node as they necessary have the same existential probability, and this probability is between [0, 1], so the UF-tree will have a large number of subtrees which increased memory requirements, **Tong et al., (2012)** comparison is not certain, because its seems that they didn't implemented the two optimizations of the UF-growth algorithm proposed by **Leung et al., (2008)** to maintain its problem, which are discretizing and rounding of the expected support values up to k decimal places, and limiting the maximum depth of recursion to two which limit the construction of UF-tree.

2.3. UEclat Algorithm

UEclat is a vertical uncertain data mining algorithm **(Abde-Elmegid et al., 2010)**, it is an adaptation of the traditional Eclat algorithm introduced by **(Zaki, 2000)** which helps finding frequent itemsets from uncertain databases. The operation will start by generating the transaction id set known as *Tidset* table from the database, where each item *x* is combined with its set of transaction identifiers *Tidset* at which *x* appear, but in case of probabilistic data, the presence of items in transactions is presented by a existential probability value between [0, 1]. UEclat implements a different structure known as the *Utidset* table, which means the uncertain transaction identifiers set table that is a derivation from the *Tidset*. Thus, this table stocks the item, its existential probability in every transaction from only a single scan of the database, known as the *Utidset*(*Table.1*).

Table.1 : Deterministic *Utidset* vertical representation of all items.

TID	Items
T1	A(0.9), B(0.8), C(0.7), F(0.8)
T2	A(0.9), C(0.7), D(0.6), F(0.1)
T3	B(0.9), C(0.5), E(0.4)
T4	B(0.9), E(0.2)
T5	A(0.9), C(0.7), D(0.6), E(0.3)

Once the *Utidset* is built, the next step is to generate the *Tidset* table of frequent items, which called the '*prune'* step, the *exSup* of an item *x* is calculated using *Rule.3*, that is

the sum of all existential probabilities of item *x* in all the transaction within the database, to check if item *x* satisfy the frequency condition. If it is respected, then x is a frequent item and will be added in the *Utidset* table, this process is repeated for all remaining items.

Finally, for the *'mining'* step, the *exSup* of a pattern *X* or *k-itemsets* where $k \geq 2$, as presented in *Rule.3* is the sum of the product of all existential probabilities of items within *X* in all the transactions, the obtained *exSup* is compared with the *minSup*, and only the *k-itemsets* which serve the frequency condition will be added to the *k-itemset* projected database, and this operation is done recursively until all frequent possible *k-itemsets* are mined, and added to the final projected DB.

Additionally to UEclat, there are a lot of improved algorithms such as U-Eclat **(Calders et al., 2010)**, UV-Eclat **(Leung & Sun, 2011)** that mine frequent items vertically and have shown an effective and usually outperform horizontal approaches because of the feature of vertical mining, which is using the independence of classes term, where each frequent item is a class that contains a set of frequent *k-itemsets.*

C. High Utility Pattern Mining from Uncertain Data

1. Definition

High Utility Pattern Mining (HUPM) **(Lin et al., 2016)** is the second technique defined in this chapter, where it is an utilizable and effective task for mining and analyzing data with another concept. HUPM's purpose is to discover patterns in transactional databases, which considers both quantity and profit of items, an example of an uncertain quantitative database with its probabilistic values is shown in *Table.2*. and its corresponding profit table is presented in *Table.3*.

Table.2: A Transactional uncertain database *D*.

TID	transaction (item, quantity)	probability
T1	{A,3},{B,2},{D,3}	0.8
T2	{C,1},{B,3},{A,1}	0.9
T3	{C,3},{D,1}	0.5

Table.3: The profit table.

item	Profit
A	2
B	3
C	10
D	15

Previously, in the high utility section, HUPM and their mining algorithms that handle precise databases which consist only of the utility function, because data is binary. But, with

uncertain databases known as UDBs, the measurement is different. Unlike UFPM which considers only the objective measure which is probability of existence of a pattern, UHUPM tries to combine both probability and utility, that is a semantic criterion to measure the utility of a pattern based on items of high importance provided by the user.

In another manner, consider a pattern of itemsets depending on frequency as well as utility to predict more profitable itemsets from UDBs, because for the uncertain databases, itemsets with high utility and high existential probability are useful to users, not itemsets with only one of them. The traditional HUPM algorithms do not consider the existential probability, and they are insufficient to process UDBs. The development of other algorithms for UHUPM is still in progress and has not yet been considered. Thus, no algorithm has yet been proposed for exactly mining high utility patterns (HUPs) in an uncertain database.

Therefore, according to our research, we found that **Lin et al., (2016)** supposed a framework known as the Potential High-Utility Itemset Mining(PHUIM) model to mine UDBs with both probability and utility. Thus, down from this section, let's take a brief look at the proposed **Lin et al., (2016**) PHUIM model and their two algorithms.

2. Potential High Utility Itemset Mining Model

Potential High-Utility Itemset Mining (PHUIM) **(Lin et al., 2016)**, aims to discover itemsets with high utility that depend on the two criterions utility and probability.

First, the probability measure which is similar to the expected support notion defined in the part of the uncertain frequent pattern mining problem is the same adopted in the **Lin et al., (2016)** proposed PHUIM framework, and the name was changed to the potential probability measure. However, to understand the second measure, which is utility, let's present below some basic concepts.

Consider I a set of n definite items $I=\{i1 , i2 , ..., in \}$, an uncertain database D , which contains a set of m-transactions $D= \{t1 , t2 , ..., tm\}$, where for each transaction tj that contain a subset of items, we have $tj \in D$, an item ix in a subset tj is categorized by its quantity $q(ix, tj)$, and for every transaction tj from D is identified by a unique value which is its existential probability $p(tj)$ in D. Thus, each item ix has its profit value pfx, that is in the created profit where $pfTab=\{pf1, pf2, ..., pfn\}$, and let $X=\{i1 , i2 , ..., ik\}$ is a *k-itemset*, where $1 \leq k \leq n$, this X is a part of the subset tj only if X is a subset of or equal to tj: $X \subseteq tj$.

Since we have two measures, then we have two thresholds, the minimum utility threshold ε, and the minimum potential probability threshold μ.

Down to this part, to clarify the concept, we will introduce the basic rules that are proposed by **Lin et al., (2016)** which helps finding the high utility itemsets and the potential utility itemsets. First, let's start by the utility of an item, an itemset, a transaction in the database, then both its probability and potential probability.

The utility of an item *ix* in a transaction *tj* is defined in *Rule.6:*

Rule.6: $u(i_x, t_j) = q(i_x, t_j) \times pf(i_x)$

The utility of an itemset *X* in a transaction *tj* is defined in *Rule.7:*

Rule.7: $u(X, t_j) = \sum_{i_k \in X \wedge X \subseteq t_j} u(i_k, t_j)$

The utility of an itemset *X* in a database *D* is defined in *Rule.8:*

Rule.8: $u(X) = \sum_{X \subseteq t_j \wedge t_j \in D} u(X, t_j)$

The transaction utility of transaction t_j is defined in *Rule.9:*

Rule.9: $tu(t_j) = \sum_{k=1}^{m} u(i_k, t_j)$

The total utility of transactions in database *D* is defined in *Rule.10:*

Rule.10: $TU = \sum_{t_j \in D} tu(t_j)$

The probability of an itemset *X* in a transaction *tj*, where $X \subseteq tj$ is presented in *Rule.11:*

Rule.11: $p(X, tj) = p(tj)$

The potential probability of an itemset *X* in the database *D* is presented in *Rule.12:*

Rule.12: $Pro(X) = \sum_{X \subseteq t_j \wedge t_j \in D} p(X, t_j)$

According to **Lin et al., (2016)** an itemset *X* in a database *D* is defined as a high utility itemset(*HUI)*, if it satisfies the *Rule.13*, else it is called a low utility itemset.

Rule.13: $u(X) \geq \varepsilon \times TU$

Moreover, an itemset *X* in a database *D* is defined as a high potential itemset(*HPI)*, if it satisfies the *Rule.14*, which is shown below:

Rule.14: $Pro\ (X) \geq \mu \times |\ D\ |$

Finally, an itemset *X* is defined as a potential high utility itemset *(PHUI)*, only if it respects the two rules above, that is an *HUI* from *Rule.13*, and is an *HPI* from *Rule.14*.

Now, let's know and define the two algorithms proposed by **(Lin et al., 2016)**, which helps the potential high utility mining process from uncertain databases.

2.1. PHUI-UP Algorithm

PHUI-UP is the first potential high utility algorithm proposed by **(Lin et al., 2016)** to solve problems for PHUM in an uncertain database, and it is based on the level-wise Apriori approach, which means that is work with the same probability properties of the already presented probability count of the Apriori algorithm, but with some different terms.

Therefore, let's present the Apriori extended downward closure properties for both *HPI* and *HUI*, which are considered as basic concepts that are used by this algorithm. First, the downward closure property of the high probability itemset (*HPIDC)* is the same as defined in *Rule.4*. Thus, as mentioned in *Rule.15*:

Rule.15: $Pro\ (X^{k-1}) \geq\ Pro\ (X^{k})$

Which implies that if X^{k} is an *HPI* than necessarily X^{k-1} respects the *HPI* condition. Next, to mine *HUIs*, we cannot use the *ARM* properties. Therefore, **Lin et al., (2016)** proposed a transaction weighted probabilistic and utilization downward closure *(TWPUDC)* property, let's present the basic rules of this property:

The transaction weighted utility *(TWU)* of an itemset *X* in *D* is defined in *Rule.16*:

Rule.16: $TWU(X) = \sum_{X \subseteq t_j \wedge t_j \in D} tu\ (t_j)$

However, in the same manner as the *HPI* defined in *Rule.15, TWDC* is obtained if the *Rule.17* is implemented:

Rule.17: $TWU(X^{k-1}) \geq TWU(X^{k})$

An itemset *X* in *D* is considered as a high transaction weighted utilization itemset *(HTWUI)* if it respect the condition presented in *Rule.18:*

Rule.18: $TWU(X) \geq \varepsilon \times TU$

An itemset *X* in *D* is defined as a high transaction weighted probabilistic and utilization itemset *(HTWPUI)*, if it satisfies the two mentioned rules, *Rule.14* and *Rule.18* respectively, and the downward closure property *TWPUDC* of *HTWPUIs* is examined only if the *HPIDC* and *TWDC* in *Rule.15* and *Rule.16* are respected. Let's now link between the previous *PHUI* and the *HTWPUI* as shown in *property.6:*

Property.6: *"The transaction-weighted probabilistic and utilization downward closure (TWPUDC) property ensures that PHUIs ⊆ HTWPUIs. It indicates that if an itemset is not a HTWPUI, then none of its supersets are PHUIs.***(Lin et al., 2016)***"*

From ***property.6*** we can extract *Rule.19*:

Rule.19: $u(X^{k-1}) \leq TWU(X^{k-1})$

After implementing all the necessary rules, let's move to the PHUI-UP algorithm process.The proposed *PHUI-UP* algorithm has two phases. During the first phase, it scans the database to find the TU, *TWU* values and the probabilities of all 1-itemsets within the database, next it checks if the *TWU* and probabilities *(Pro)* for every itemset satisfied the *HTWPUI* conditions implemented in *Rule.14* and *Rule.18.* and putting them into the first set of $HTWPUI^{k}$where *k* is initially set to 1. Then the algorithm start generating candidate set *C2* where *k* is set to two, and the database is then rescanned to calculate the *TWU* and *Pro* values of each itemset in *C2*, this phase repeated to generate the next candidates C*k+1* for discovering $HTWPUI^{k+1}$ in a level-wise way until no candidate is generated and a complete set of *HTWPUIs* is built. Next, in the second phase, the *HTWPUIs-set*. An additional database scan is done to find the final *PHUIs* from the *HTWPUIs-set* based on *property.6*.

A complete set of *PHUIs* discovered by the PHUI-UP algorithm is classed in *Table.4* where these results are obtained using the previous uncertain database coordinations

presented in *Table.2* and *Table.3,* the final *PHUIs* set contained only the itemsets *{C}, {D} and {AB}*.

Table.4: The final derived *PHUIs* for the example uncertain database.

itemset	Utility	probability
{C}	40	1.4
{D}	60	1.3
{A, B}	23	1.7

We cannot disagree that the PHUI-UP algorithm extracts the *PHUIs*, but it has a long execution time problem, because it scans the data several times. Therefore, in order to address this problem, **Lin et al., (2016)** proposed an improved algorithm called the PHUI-List.

2.2. PHUI-List Algorithm

PHUI-List is a potential high utility algorithm proposed by **Lin et al., (2016)** to solve the previous PHUI-UP algorithm problem. However, this algorithm mines the uncertain data within three phases, from a vertical data structure, known as the *PU-List*, which means the probability utility list, to building a tree, then starting the pruning technique. Before introducing these three steps, first, let's define some rules and properties, which are proved by **Lin et al., (2016)** and they are used in the PHUI-List algorithm.

To count the utilities ***iu*** of itemset *X* in *tj*, we use *Rule.20:*

Rule.20: $$X.iu = \sum_{i \in X \wedge X \subseteq t_j} u(i, t_j)$$

To calculate the remaining utilities ***ru*** of itemset *X* in *tj*, we use *Rule.21:*

Rule.21: $$X.ru = \sum_{i \notin X \wedge i \in t_j \wedge X \prec i} u(i, t_j)$$

The sum of the utilities ***IU*** of an itemset *X* in *D*, is defined in *Rule.22:*

Rule.22: $$X.IU = \sum_{X \in t_j \wedge t_j \subseteq D} (X.iu)$$

The sum of the remaining utilities *RU* of an itemset *X* in *D*, is defined in *Rule.23:*

Rule.23: $$X.RU = \sum_{X \in t_j \wedge t_j \subseteq D} (X.ru)$$

During the first phase, considering an itemset *X,* the algorithm first scans the original uncertain database *D* once to extract the necessary informations such as the *TU,* the *TWU*, and the *Pro* values of each item in *X*, then using the *Rule.18* to take only the *1-itemsets* that satisfy the *HTWPUI* condition, these $HTWPUIs^1$ are sorted in ascending order by their TWU values or by alphabetical order. Next, the PHUI-List scans *D* once more to generate the *PU-List* of each *1-item* in the *1-itemsets*. Where each row in the *PU-List* of an itemset *X* in transaction *tj* contains four fields, the transaction identifier '*TID'*, where $X \subseteq tj \in D$, the probability *'prob'* of *X* in *tj*, the utility *'iu'* of *X* in *tj*, the remaining utility *'ru'* for *X* in *tj*. However, to count the *prob* we use *Rule.11*, for the value of *iu* we use *Rule.20*, then to calculate the *ru* we use *Rule.21*. This stage is summarized in *Figure.8,* which define the constructed PU-lists of $HTWPUIs^1$, where we applied the PHUI-List algorithm in the same uncertain database *D* presented in *Table.2.*

{C}			
TID	prob	tu	ru
2	0.9	10	11
3	0.5	30	15

{A}			
TID	prob	tu	ru
1	0.5	6	51
2	0.9	2	9

{B}			
TID	prob	tu	ru
1	0.8	6	15
2	0.9	9	0

{D}			
TID	prob	tu	ru
1	0.5	45	0
3	0.5	15	0

Figure.8: *The constructed PU-lists of* $HTWPUI^1$.

Since we constructed the *PU-Lists,* we start the second phase, which starts the construction of the *Set-enumeration* tree from the sorted *itemsets*, where each node in the

tree contains an itemset, which is the extension of its parent node. Then, we will use a depth-first search to discover the *Set-enumeration* tree. The third phase is the pruning step, two pruning strategies are applied to decide whether the extensions of the processed node need to be explored, the first pruning strategy is defined in *property.7*, and for second is presented in *property.8*.

Property.7: *"When traversing the Set-enumeration tree using a depth-first search strategy, if the sum of all the probabilities of a tree node X in its constructed PU-list is less than the minimum potential probability, then none of the descendant nodes of node X is a PHUI."*

Property.8: *"When traversing the Set-enumeration tree using a depth-first search, if the sum of X.IU and X.RU in the constructed PU-list is less than the minimum utility count, then none of the descendant nodes (extensions) of node X is a PHUI."*

If the processed node satisfies the two conditions identified in *property.7* and *property.8*, then the extensions of the descendant nodes of the processed node will be explored recursively, until we discover *PHUIs* directly without scanning the data every time, and without candidate generation. According to **Lin et al., (2016)**, the PHUI-List algorithm is correct and complete, because it discovers all *PHUIs* and only the *PHUIs*.

IV. Conclusion

This chapter has provided a brief analysis of the technologies used and developed recently to mine patterns from uncertain databases. We can conclude that all the previously mentioned algorithms still have their drawbacks, like consuming memory, taking a long execution time, because the data complexity grows exponentially with time, this problem enters us in a new era known as the big data, which is introduced in the next chapter.

Chapter II:

Big Data Uncertain Pattern Mining

I. Introduction

Since we are in the big data era, which means that big data is everywhere, traditional pattern mining algorithms are not sufficient to handle problems associated with such applications and complex sources. Therefore, the level of difficulty is augmented, and the previous proposed algorithms which handle the uncertain databases such as UApriori, UV-Eclat, UF-Growth, ... etc. are ineffective according to the increasing needs of managing a large quantity of data, because these algorithms execute the mining process sequentially on a single local computer. Thus, they cannot result in the same performance made in previous simple uncertain data with these huge datasets, which generate a new challenge known as mining patterns from uncertain big data. Therefore, to solve this complexity many researchers started searching for solutions and techniques to minimize the big data issues.

In this chapter we will present some of these proposed solutions and make a comparative review between these different parallelized frequent pattern mining algorithms based on several measures, parameters, and understanding the advantages and limitations of each algorithm. Moreover, the first section contains an abstract definition of the uncertain big data and explain its various paradigms and mechanisms that are used in recent years in both parallel or distributed data mining platforms, these techniques have been further designed to accomplish a high performance data mining, next we briefly introduced a set of the analyzed algorithms that used some of the researchers techniques which are frequently used and outperformed in all kind of uncertain big data in a distributed environment known as the MapReduce and Spark models such as MR-Growth, MR-PUFGrowth, BigAnt, ApproxFP…etc, then in the comparison section we studied and discussed the above listed algorithms based on a set of parameters, finally we gave a conclusion as a result of our review.

II. Background

This section introduces background information on the main characteristics of uncertainty in big data, with their techniques and paradigms.

A. Uncertain Big Data

In the previous chapter we presented a brief definition about the uncertain data, but when we have a large dataset then the big data term appears, which is the collection of a huge and complex data and it grows exponentially with time, This term describes datasets which are so large or

complex that traditional data processing applications are inadequate to deal with them Moreover, it created a lot of challenges including analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying and updating information. Therefore, computer scientists have made outstanding contributions to the application of big data and introduced a concept to handle difficulties associated with such applications, known as data mining from uncertain large data, which is the way to explore big data using complex mining tools. Additionally, this technology has been playing an increasingly important role in decision making activities, in order to discover the hidden relationships that exist within the data, and decide the value and meaning of this information.

To understand the existence of uncertainty in large datasets, let's explain the main features of big data. Our interest is in the 4V (**Chebbi et al ., 2015)**, which are the volume, the variety, the velocity and veracity. First, high volumes of valuable data such as texts and documents, medical images, business transactions, banking records, surveillance videos, telecommunication and social media data can be easily collected or generated from different sources, in different formats, which mean that various parts can be managed with different tools. However, this refers to the large amount of data involved, the size of available data is growing at an increasing rate, which leads to the transformation from local servers to cloud or external partners and old approaches cannot analyze large sets of data which give inconsistent results.

Secondally, variety refers to the different types of the collected sources, structured databases such as text files, csv, excel..etc and unstructured databases like videos, text, graphics.. etc which are more useful to humans and that’s mean that create more work and requires more analytical skills to decipher it. next for the velocity, it is extremely important, it shown how fast the data is entered and processed, to get a better comprehension, let’s give an example of a massive data, just consider the social media case, when there are 300,000 status updates, 140,000 photos uploaded, and more than 500,000 comments made every minute, adding to this the video calls which needs a real time transmissions which have to be processed just as quickly. The last one is that veracity is the most important “V” of big data, which indicates data honesty and accuracy, where the data veracity is attributed to the source from which it comes. Thus, in this age data counts are collected from different sources, constituting heterogeneity in the data which results in a difficulty to know about the quality of data, that leads to the overlapping of the rapid changes between the old data and the updated one, and this creates uncertainty.

As a consequence to the definitions given in this above section, we can signify that uncertainty is presented when data is big, fast, diverse and incorrect. As we identified situations which result in the appearance of inconsistent big data, the manner to fix these challenges is noted down in the paradigms of big data.

B. Paradigms of the Big Data

Because of the above listed features: huge volume, high level of velocity, variety, and veracity, all standard tools are not applicable, which motivates researchers to develop some new approaches oriented to interact with sources of big data. In this section, let's define these useful paradigms that are implemented in uncertain large data and show an effective progress. Many developers used them to introduce new massive incomplete data algorithms that gave good results.

1. **MapReduce***:* this paradigm was published by google **(Dean & Ghemawat, 2004)**, it is the heart of the apache hadoop ecosystem, where it is the most common software framework that facilitates the user to write applications that can process vast amounts of data on large clusters. However, it is a batch processing technique and a program model for distributed computing written in java, it creates a two-stage execution graph consisting of data mapping and reducing, where each task has a certain operation, the map function which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples where the input is a couple of a key and its value, secondly, the reduce step, which it input is the takes the shuffled output of the map step and combines those data tuples into a smaller set of tuples, these two functions are repeated continuously, after each map and reduce the data is sent back to the disk part, this is considered as both a limitation and an advantage, the back to the disk made the map and reduce much slower, but the good point that is a more fault-tolerant, it offers a better security by using all the hadoop protection features.
2. **Spark***:* (**Zaharia et al., 2010)** is an open-source project that is under the wing of the apache software foundation, which can quickly execute distributed processing tasks across multiple computers on very large datasets. However, spark comes with in-built versions of java, scala and python, which make it easier to use, also it runs tasks one hundred times faster in-memory and ten times faster on disk than MapReduce, especially when compared with multi-stage jobs that require the writing of state back out to disk between stages, it aims to reduce the number of steps in a job, and reuse data across multiple parallel operations in

only one needed step, because we already indicated that MapReduce is a disk-based framework unlike the spark which runs on RAM and execute tasks using a cache to greatly speed up machine learning algorithms that repeatedly call a function on the same dataset, additionally, it is a suitable tool for IoT sensors, social media sites and log monitoring, thus it is using for batch processing as well as near real-time processing. Also, one of the important concepts of spark is the resilient distributed dataset(RDD), most of the spark APi is built based on RDD, taking the traditional map and reduce functions and adding some built-in support for joining data sets, filtering, sampling, and aggregation. Moreover, one of the limitations of the spark framework is that it is less fault-tolerant because when an unexpected event proceeds it has to start the processing from the beginning.

3. **Graphics processing units(GPU):** is the soul of the computer that is used to accelerate processing operations on big data through the harness of the massive parallelism mechanism. Besides, GPU computing consists of using graphics processors which use transistors for the processing step in addition to CPUs that implement transistors for caching data, as a purpose of the use of both GPUs and CPUs was to speed up data science and machine learning applications. Thus, let's clearly understand the GPU parallel architecture, each GPU is composed of a number of cores where each of these cores has a number of functional units. For thread processing, one or more of these functional units can be used. However, these units are known as thread processors, and they all share the same control unit. Therefore, they are able to perform the same instruction.

4. **Multi-Core computing**: is a modern technology based on the improvements in processor and network technologies, since every big data center server has multiple CPUs to maintain the same single task. Therefore, using this technology that it is a parallel processing that has two or more central processing units or cores attached which runs multiple computational threads of a program in order to enhance performance and reduce power where each core executes multiple instructions in parallel, this means that a program is splitted into diverse threads, where each thread will proceed its task on a separate CPU or core. As a consequence, this mechanism implemented two terms, the multi-threading and the shared memory techniques to perform the parallel processing.

5. **Grid computing**: is a special type of distributed computing, which is a bunch of computers geographically distributed and connected by networks that work together like a

super-computer to perform a task that would be difficult for a single machine to handle, where these computers are connected by some bus, making ethernet or sometimes internet. Thus, the machines on that network work under the same protocol, adding to that instead of many CPU cores on a single machine, it contains multiple cores spread across various locations, and the computers communicate and exchange resources with each other, well, this signifies that all computing resources do not have to work on the same specific task, but can work on sub-tasks that collectively make up the end goal, also means that a computer on the node can swing in between being a user or a provider based on its needs. Moreover, there are two kinds of network homogeneous or heterogeneous based on the operating system of the machines in the network, which differ while using grid computing than other distributed computing architectures, as long as it uses middleware to control the network and its resources and authorize any process that is being executed on the network.

III. Parallel and Distributed Frequent Pattern Mining algorithms

A. MR-Growth Algorithm

The MR-Growth algorithm **(Leung & Hayduk, 2013)** extends from the UF-Growth **(Leung et al., 2007)** algorithm and improves its weakness when detecting frequent patterns in large uncertain data. This algorithm uses a different framework which is the MapReduce model by applying two sets of the “map” and “reduce” functions in a tree-based pattern-growth fashion. Additionally, three enhancements to the MR-Growth algorithm were proposed, they started with the ForkJoin approach to complete multiple tasks in a minimized time via multi-core processors. Secondally, the efficient conditional tree construction, which allows MR-Growth to construct conditional trees without constructing projected trees first. The third improvement was given as a solution to the big UF-trees by implementing the concatenating sample method to avoid building UF-trees while the mining process begins.

B. MR-PUFGrowth Algorithm

The initiation point was from the MR-Growth **(Leung & Hayduk, 2013)** UF-tree large size which occurs when the same item has many different probabilities, and this drawback affects the efficiency. Therefore, **Rathan & Rani, (2017)** introduced the MR-PUFGrowth, they merged between the two PUFGrowth **(Leung & Tanbeer, 2013)** and the MR-Growth**(Leung & Hayduk, 2013)** algorithms, it is a MapReduce model of the PUFGrowth based on the MR-Growth, but instead of storing data in the UF-Trees, they are stocked in smaller trees known as PUF-Trees.

C. MR-UV-Eclat Algorithm

The MR-UV-Eclat **(Braun et al., 2018)**, proposed to reduce processing time and memory usage Braun et al. introduced a MapReduce version of UV-Eclat**(Leung & Sun, 2011)**. After vertical data conversion, it has two steps. First, the mapping step which is divided into two processes, the expansion process, by concatenating the current itemset with each suffix node list. Then, the pruning process to verify the new candidates in order to calculate the new support to add it to the new mapped key containing. After this tree expansion, all new candidate vectors are generated, the algorithm extracts the candidate itemset and its support value from the mapped records formed previously in the mapping step. Secondally, the reducing step by summing all the support values and comparing super value with minimum support threshold, to keep in the end only records larger than support threshold.

D. BigAnt Algorithm

The BigAnt was proposed by **Leung & Jiang, (2014)**, refers to an uncertain algorithm that mines frequent patterns from massive data by utilizing the anti-monotonic constraints (AM) via MapReduce framework. Thus, there were many algorithms that handle uncertain databases using the constraints features but no one performed in uncertain big data. Therefore, **Leung & Jiang, (2014)** introduced BigAnt which is based in a pattern growth fashion, (tree based approach) that mines only interesting patterns that satisfy the user-specified constraints of the truly frequent itemsets from inconsistent massive data. Moreover, the traditional process was to extract frequent itemsets and then the user will select his interested itemsets. For that reason, in order to avoid wasting a lot of time and space, the BigAnt gives more options to the user, which are the user-specified frequency constraint such as the *minSup* and the user-specified non-frequency anti-monotonic constraints that checks only the interested user items. Additionally, it has two map and reduce functions, in the map step it checks the non-frequency AM constraints, and in the reduce step it checks the frequency constraint as a result to form valid interesting itemsets that match up the user selected constraints with a high velocity and high variety.

E. PNDUA Algorithm

The PNDUA **(Xu et al., 2015)**, calculates the exact probabilistic frequentness by using MapReduce framework, this last is very complicated in uncertain data, where researchers already proposed some approximation algorithms such as DN **(Calders et al., 2010)**, PD **(Cam, 1960)**...etc, but these algorithms didn’t show acceptably effective while handling very large, dense or sparse

datasets. Therefore, **Xu et al., (2015)** introduced the PNDUA algorithm to solve this problem, it is a parallel normal distributed based UApriori algorithm **(Chui et al., 2007)**, which is splitted into several sequential steps, starting with the first step which contain a single map and reduce functions to calculate the expected support of each singleton frequent itemset, then it takes the output of the step 1 to start second step, which generates candidates of frequent *2-itemsets* in condition that they aren't empty. Thus, in the same manner the next steps will generate frequent *k-itemsets* and stop the process only if there is no more frequent candidate generated list. Furthermore, one of the PNDUA limitations is that it doesn't satisfy the problem mentioned above completely, it is runnable only in dense datasets, not in both sparse and dense.

F. PuFIM Algorithm

The PuFIM **(Ding et al., 2019)**, mines important and interesting itemsets by using probability and weight over Pufim-tree, where its idea was to consider the weight of item **(Lin et al., 2015)**, in other hand, there is also more researchers that presented some algorithms that discover high weight probability patterns based on their importance and weight from uncertain datasets like HEWI-UApriori **(Lin et al., 2015)** algorithm, but not from massive data. Therefore, PuFIM combines the two mentioned ideas to perform a parallel weighted uncertain frequent pattern mining using the spark model from uncertain large data, its operation is considered in three steps, the first takes data as an input to employ different map functions using the extended definitions of the HEWI-UApriori, then scan the data twice to give the key and max weight probability for each item set as an output, the next step is the creation of the PuFIM-Tree that mines recursively all high weight probability patterns. Then, in the last step PuFIM extracted the frequent pattern according to the expected weighted support of each itemset.

G. ApproxFP Algorithm

The ApproxFP **(Xu et al., 2014)**, proposed in term of uncertain frequent itemset, two major measures can be used which are the expected support based frequent itemset applied in some previous mentioned algorithms, and the probabilistic frequent itemset(PFI) which is more complex as a reason that is counted with the probability distribution of item support, according to recent studies researchers aims to use the approximated poisson distribution **(Cam, 1960)** to find frequency to develop efficient approximation algorithms such as MBP **(Wang et al., 2010)**, ApproxApriori **(Calders et al., 2010)**... etc which are much faster and achieved good results in uncertain dense datasets, but not in sparse, big data. Therefore, **Xu et al., (2014)** introduced

ApproxFP to fix this limitation, this approximated frequent pattern algorithm on MapReduce platform examine both expected support and PFIs by using an ApproxFP-tree which is constructed in the reduce task to avoid spend much time in multiple scans of the database, but without building the repeated pattern growth sub-trees because during the PFIs search from the tree, it becomes more smaller by reason of working and comparing generated candidates with the maximum estimated support which is computed with the prefix estimation method this leads to a short runtime and a high speedup and accuracy in mining frequent patterns in sparse and uncertain big data.

IV. Comparison and Discussion

Our comparative study of all the parallel and distributed uncertain pattern mining algorithms mentioned before is based on various measures and parameters such as technique used, data representation, division strategy …etc, which can differentiate from an algorithm to another, as shown in Table 5. Moreover, while comparing the advantages and limitations of the algorithms, we can extract the most efficient, scalable, flexible, and accurate algorithms

(less memory consumption algorithms according to the time consuming between them).

Table.5. Comparative study of different FPM algorithms from big uncertain data.

Aspects of comparison	Names of the algorithms						
	MR-Growth	BigAnt	ApproxFP	PNDUA	MR-PUFGrowth	MR-UV-Eclat	PuFIM
Year	2013	2014	2014	2015	2017	2018	2019
Extends	UF-Growth	UF-Growth	MBP & UF-Growth	UApriori	MR-Growth	UV-Eclat	HEWI-UAprioii & UF-Growth
Division Strategy	Search Space Split	Divide data into several partitions	Search and prune	Divide data into several partitions	Search Space Split	Search Space Split	Search Space Split

Data Representa-tion	Horizontal	Horizontal	Horizontal	Horizontal	Horizontal	Vertical	Horizontal
Storage Structure	UF-Tree	UF-Tree TPC-Tree BLIMP-Tree	ApproxFP-Tree	Array	PUF-Tree	Array	PuFIM-Tree
Technique used	Pattern Growth fashion	Constraints via Pattern Growth fashion	Prefix estimation method without sub-trees	Breadth First (candidate generation)	Prefixed Item cap	Item Centric fashion	Weighted items via Pattern Growth fashion
Framework	MapReduce	MapReduce	MapReduce	MapReduce	MapReduce	Spark	Spark
Used Datasets	.Accidents .Connect4 .Mushroom	.Accidents .Connect4 .Mushroom	.Kosarak .T10I4D100K	.Accidents .Connect .Kosarak	.Connect .T10I4D100K .Retail .Mushroom	.Two unknown datasets	.Retail .Mushroom .T10I4D100K
Operating system	Windows 7 64-bits	Windows 7 64-bits	Ubuntu 12.04 32-bits	Ubuntu 11.04	Ubuntu 14.04	-	-
Main memory	8GB	8GB	4GB/1GB	4GB/1GB	4GB	-	4GB
Programm-ing language	Java	Java	Java	Java	Java	-	Scala

Advantages	.Multi-core Processors in the ForkJoin Framework .Efficient Conditional Tree Constructio n .Sampling the UF-Trees	.Space and time efficient because fewer pairs need to be shuffled and sorted .Small selectivity: short runtime and high speedup. .Flexible: for any databases type precise or uncertain	.Scalable, it avoid multiple data scans and time consuming by using ApproxFP-Tr ees .Fastest running time and high speed because there is no sub-trees are built .High accuracy on sparse uncertain big datasets, PFIs are found with poisson approximatio n	.A parallel normal distribution algorithm. .Better accuracy in denser datasets. .Quite time consuming compared to other approximatio n algorithms.	.Compact data representati on for uncertain pattern	.Databases no need a repeatedly scan each time which make it fastest and more scalable .Consumes the least amount of memory as a reason that it uses the depth-first search method.	.Extracting interesting patterns based on probability and weight measures. .The use of Pufim-tree to reduce the times of scanning the databases. .Scalable, and the frequent patterns are fewer and more precise

Drawbacks	.UF-Trees are not as compact and their size becomes larger for huge datasets	.The results are based and dependent on the selectivity of the user, which needs to be precise and true to get useful and valid patterns to avoid wasting time and memory in processing unsure or wrong interested patterns	.The random process still may have a non accurate results	.Inflexible, it works only in dense datasets, therefore it is ineffective in sparse data .One of its operations failed which made it a limited algorithm.	.The large number of the distributed distinct items leads to worse performanc e when they are in a large set of transactions	.The T-id sets can be long, therefore expensive to manipulate, and consumes more memory especially in dense datasets.	.The runtime of PuFIM is not reduced under the distributed environment because it consider both the probability and weight of the itemset which is more time consuming

In term of memory and time processing, ApproxFP and BigAnt has the less time consuming and achieved a high speed up comparing to the remaining algorithms, such as those who uses the breadth first search, the pattern growth and the weighted fashions, Approx is a tree based algorithm but without building sub trees which takes more time and extra memory space while processing, where BigAnt inverse the traditional pattern growth technique to perform the selectivity, but it can be efficient only when there is such a specific constraints that avoid mining all the frequent itemsets, when there isn't, BigAnt will have the same other pattern growth algorithms limitation.

In terms of scalability and flexibility, the seven algorithms listed above perform well in uncertain big data, but depending on the data types which are several, when data is dense all the listed algorithms run without any kind of failure, but not in the second type which is the sparse, based on **Xu et al., (2014)** the ApproxFP work on the both kind of data, others like PNDUA that has show a big difficulty and fail to handle sparse uncertain Big datasets, and this made it unscalable, for the BigAnt, we cannot judge if it can be more scalable in sparse data, **Leung & Jiang, (2014)** did not give more details about it.

In terms of accuracy, almost all the previous mentioned algorithms have a good performance. PuFIM extracts interested itemsets, BigAnt outputs only the user interested patterns, PNDUA and ApproxFP are approximative algorithms, so the accuracy is based on the recall and precision.

V. Conclusion

In this chapter, we provided a comparison study of parallel and distributed algorithms proposed for mining itemsets from big uncertain data such as MR-Groth, MR-UPF-growth, MR-UVEclat, BigAnt, PNDUA, PuFIM, and ApproxFP algorithm. The objective of all these algorithms is to mine effectively and efficiently frequent itemsets from huge uncertain datasets, there are some who used the same techniques and others who tried different ones aiming to give a solution to overcome the problem of scalability in the context of big data.

But as we defined in the ***Table.5*** there are negative points too, where with the growth of the Big Data 4V's these algorithms necessitate more memory, and they faced difficulties to handle that, these challenges pushed the UFIM researchers to resort to bio-inspired metaheuristic algorithms.

Chapter III:

Bioinspired Metaheuristics and Pattern Mining

I. Introduction

The second chapter reviewed the uncertain big data characteristics, which are huge volume, high level of velocity, variety and veracity. In addition to that it introduced several algorithms that utilized one or multiple big data paradigms, developers of these algorithms clearly defined a set of instructions, equations to solve the incomplete massive data issue. In other words, they tried to transform the issue into a dependent problem by utilizing the exact or the heuristic approaches of the optimization mechanism to achieve a guaranteed optimal solution for the frequent itemsets mining.

Thus, we noticed that the implemented algorithms output differs from one datasets to another, because the methods used which work for one optimization task might not work for another. According to this hypothesis, uncertain big data becomes an NP-hard problem, its features grow exponentially with time, the larger the issue, the more complex the solution space, which makes the exact and traditional approximation algorithms become slower and it is difficult to verify its validity, this means that they work very efficiently only with simple problems. Therefore, to boost up the performance of the mining process for uncertain large datasets, a different optimization approach is used, inspired by natural phenomena known as the metaheuristics, which is used frequently in the data mining field as a fastest, efficient method to find a global solution.

This chapter is formed as follows, we will begin with a definition of the metaheuristic approach, jumping to the next section, we will focus on the population-based metaheuristic solution with an explanation of its methods and algorithms, known as the evolutionary, and the swarm intelligence algorithms.

II. Metaheuristics

The greek word meta-heuristic as defined by **Sörensen & Glover, (2013)** signifies a high level in discovering near-optimal solutions in a reasonable computational time, it had great popularity from the first appearance which was in the 80s and still until now. In computer science, it is defined as a master heuristic process designed to solve difficult optimization problems **(Yang, 2011)**.

Effectively, metaheuristic is a consequence of the failures that have been shown with the old optimization solutions that are exact and heuristic methods, it is a problem independent technique. In other words, they are stochastic iterative algorithms, where they search for the causes of the

problem to find approximate solutions at a specific time **(Tezel & Mert, 2011),** also defined as clever strategies to enhance the efficiency of heuristic procedures, especially with insufficient, bad information or restricted computation capacity.

In recent years metaheuristics plays an important role in data mining, especially in the frequent pattern mining process in an age of over increasing data, where the data mining needs to benefit from the powerful metaheuristics that can deal with huge amounts of data in order to reach a maximized accuracy and a minimized amount of information described as simplicity. Therefore, to address the optimization purpose, metaheuristics techniques are generally divided into two categories, the representation of these two categories differs from the characteristic of the study, represented by single-solution(trajectory) and population-based, natural-inspired and non-natural-inspired, dynamic and static objective, memory-usage and memory less algorithms..etc, as **Balabanov et al., (2020)** presented in *Figure.9*, the different classifications of the metaheuristic approach.

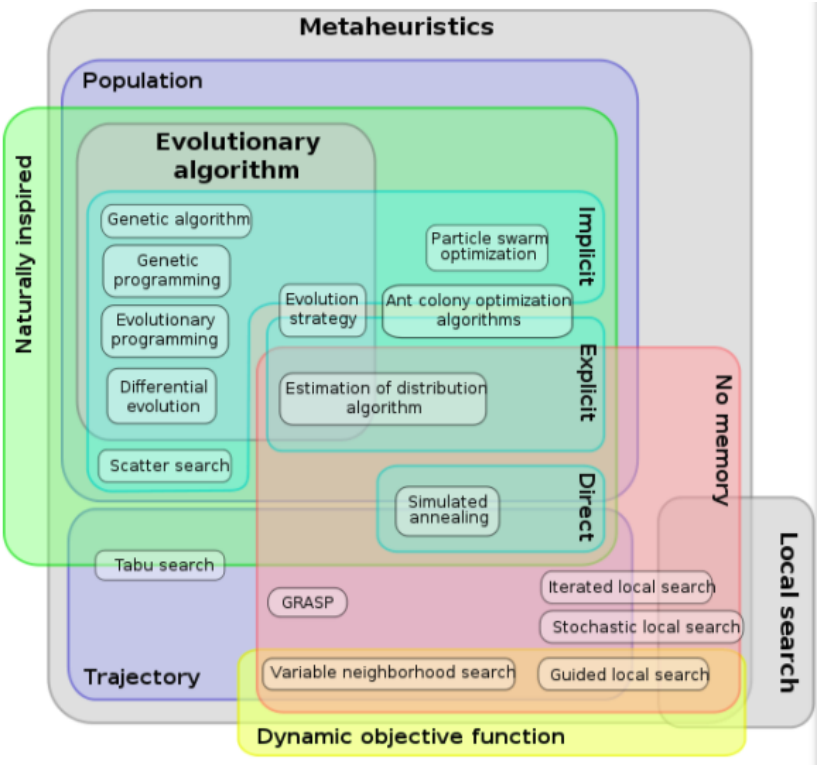


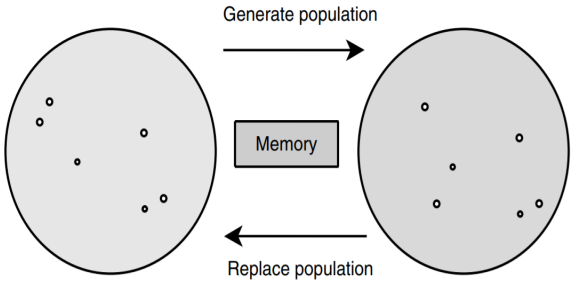
Figure.9: The classifications of the metaheuristic approach

In our study case, we focus on metaheuristic classes depending on the number of solutions used at the same time, they are divided into single based and population based, and we give more attention to the second metaheuristic category, the population based **(Talbi, 2009)** algorithms such

as the Swarm Intelligence algorithms(SIs) **(Beni & Wang, 1989)**, and evolutionary algorithms(EAs) (**Darwin & Kebler, 1859)**, they performed well compared to the single solution during the search process, also showed satisfactory capabilities to deal with high dimensional optimization problems.

III. Population Based Metaheuristics

The Population based Metaheuristics (PbMs) **(Talbi, 2009)** are generally regarded as an iterative enhancement of a population of solutions, the start point is an initial population of solutions, then two major phases are applicated repeatedly, the generation and the replacement, in the first phase a new population is created under the generation procedure, then in the replacement phase a selection procedure is done to replace the current populations with the new ones, to better understand the PbM operation, the *Figure.10* **(Repoussis, n.d.)** illustrates the representation of the PbM process in two manners, via an algorithm and an architecture, the output is given only when the operation is stopped with a satisfied criteria.



Algorithm : High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P'_t); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P'_t$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.

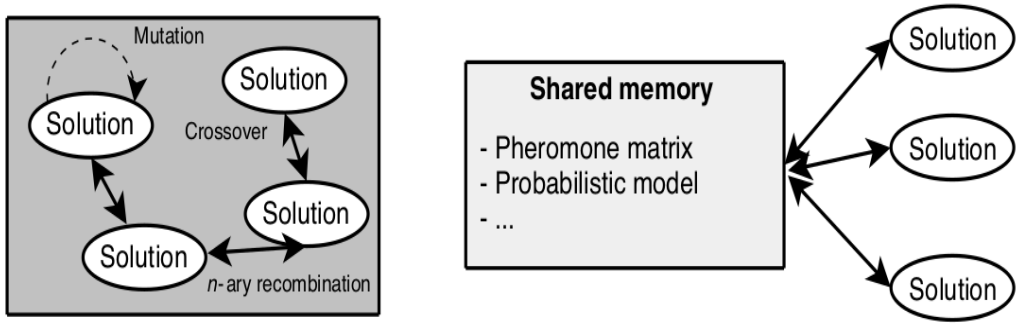
Figure.10: The general process of the Population-based Metaheuristic

Basically, the generation and the replacement phases are memoryless, in some situations they can use a memory to store the history of the search which is used in the PbM phases, known as the memory search strategy. This class is rich in efficient algorithms, which as we said before are mostly nature inspired algorithms, the difference between them is not in the techniques used but in the fashion to implement these techniques. In addition, the search memory strategy varies from an PbM algorithm to another, which means that every algorithm has its manner to represent extracted and memorized information during the search process.

Let's give some details of the population based metaheuristic, mainly it contains four important steps as introduced below, these procedures make the difference between the single-based and the population based solutions.

A. **The Initial Population:** is an important parameter, because a wrong generation of this item will affect the PbM algorithm result, and to find the initial population PbM use strategies that are measured based on three metrics, the diversity, the computational cost and the quality of initial solution.

B. **The Generation:** as presented in *Figure.11* **(Musdholofah et al., 2020)** we have two major categories, the evolution-based which has three operations (reproduction, crossover and mutation), this class includes many algorithms like the EAs **(Darwin & Kebler, 1859)**. The blackboard-based makes an indirect recombination using a shared memory which is an important input during the generation process.



(a) Evolutionary-based P-metaheuristics: evolutionary algorithms, scatter search, ...

(b) Blackboard-based P-metaheuristics: ant colonies, estimation distribution algorithms, ...

Figure.11: The Evolutionary-based PbM versus the Blackboard-based PbM.

C. **The Selection:** as specified in the *Figure.10* in the template representation, after the generation of new populations the selection procedure will select a new solution from the union of the current and the generated populations, from the produced union we can apply several selectivity techniques, we can select the newly generated solution as the new population and this is the traditional manner, remaining techniques prefer to select from the union set the best solutions as the new population.

D. **The Stopping Criterion:** it signifies the ending point of the PbM process, this constraints can be specified statically with a given criteria like the fitness improvement remains under a threshold value for a given period of time, the maximally allowed CPU times elapses, or adaptively when there is a lack of activity or changeability of the solutions the operation is stopped and the output is generated, this makes the iterative process inefficient and wastes more time and memory if it continues while there is no progress.

IV. Algorithms Using Population Based Metaheuristic

In this section we will introduce the metaheuristic algorithms which help to mine frequent and high utility patterns.

A. Evolutionary Algorithms

This type of population-metaheuristic is inspired from the theory of evolution and natural selection **(Darwin & Kebler, 1859)**, in other words, they do what biological nature does, this evolutionary process makes the population adapt to the environment better and better, it is used as the main strategy in a stochastic search manner to find optimal or near optimal solutions of the final survivors applied to optimization problems, this class cover four major and highly successful techniques such as the genetic algorithms(GA) **(Holland, 1984)**, the Evolutionary Programming (EP) **(Fogel, 1962)** and Evolutionary Strategies(ES) **(Rechenberg, 1973)**.

Let's give a general view of the EA algorithm, it will start the initialization phase by generating the first population, known as the randomized initial population which is represented with individuals (chromosomes), a fitness function is applied for the evaluation of quality of the individuals, then a set of reality based evolution operators are applicated on these chromosomes, such as selection, crossover (recombination), mutation, which leads to new individuals generation, but the size is maintained as in nature with the natural selection procedure, old individuals are partially or totally replaced, and this cycle is repeated, when the stopping criterion is satisfied the process finished and the set of solutions is given, *Figure.12* **(Eiben & Smith, 2003)** shows the general EA process, with the pseudo-code schema.

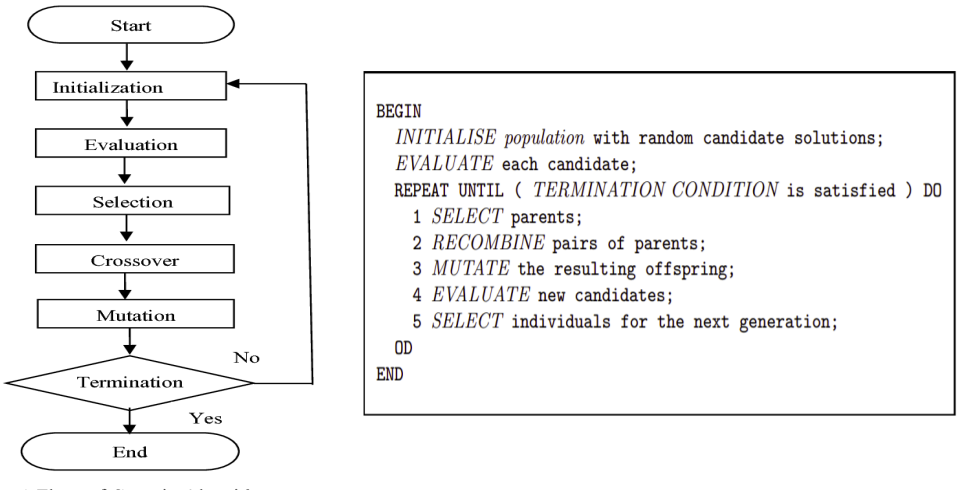


Figure.12: The general process of Evolutionary Algorithms(EAs)

After this brief summary on the EAs general process, it's also helpful to go deeper, with more details. Therefore, in the coming part we focused on the Genetic Algorithm (GA) **(Holland, 1984)**, which is one of the popular classes of evolutionary algorithms and highlighted how it works.

1. Genetic Algorithm

Genetic Algorithm (GA) **(Holland, 1984)**, is the most known branch of the EAs, well John thinked of the idea that machines can imitate the natural biological processes and apply it using a set of instructions and algorithms. However, since EAs simulate the natural evolution strategy, many mechanisms are the same as the biological terminologies. Thus, *Figure.13* defines the major GA steps **(Scrucca, 2013),** and includes some important EAs components and vocabularies that are implemented in GAs process.

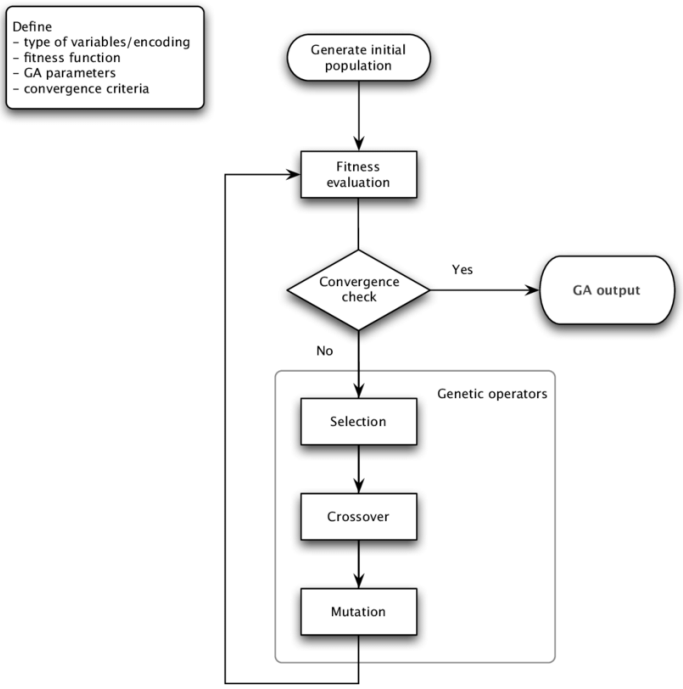


Figure.13: The Genetic Algorithms process

As presented in the picture above, the generation of the initial population is the first step, during this phase it makes the representation of the problem which contains several terminologies, such as chromosomes, genotypes and phenotypes as shown in *Figure.14* **(Massone, 2018)**, the different elements of the problem representation. In biology, genetic information are stored in chromosomes, each of them is made up of proteins and DNA which are encoded into genes, similarly in computer science, the chromosomes also named

individuals, they are given as a group of potential solutions of the problem, and the set of all these chromosomes is called population.

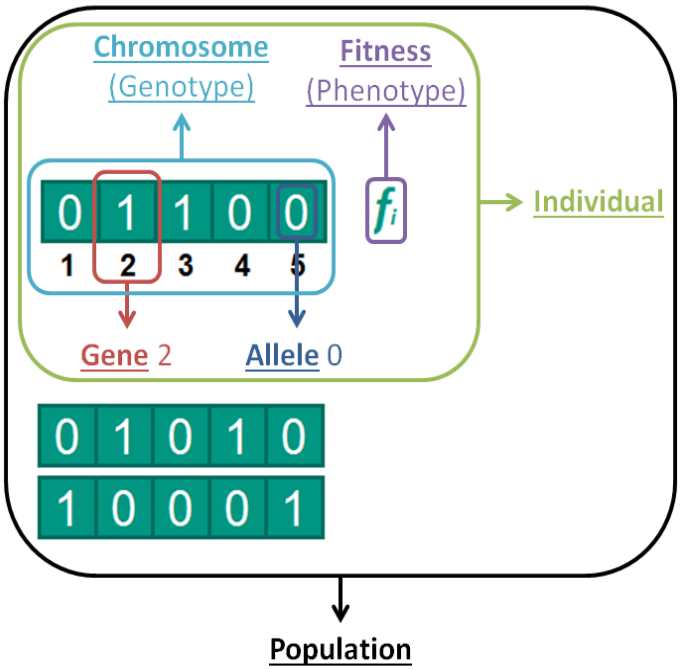


Figure.14: The different terminologies in the representation problem phase

Genotypes and Phenotypes are the link between the Evolutionary Algorithm world and the real world, the phenotype is the actual physical representation of the chromosome, they are these objects that constitute a possible solution in the context of the original problem which have an encoded version called genotypes. Moreover, a genotype is a set of genes that represent the chromosome by a software data structure, and its size is often dynamic or static based on the problem domain. However, when we traverse from phenotypes to genotypes the mapping here is called encoding, the decoding is when inversing from genotypes to phenotypes.

An individual in GAs is generally described as a fixed length vector, the genotypes use generally the binary encoding, the initial population which is generated randomly contains a set of chromosomes that are presented as strings of binary encoding digits, the initial population is seeded with solutions that are known to have good fitness, because the choice of these solutions can affect the GAs near-optimal convergence ability.

Another essential element in the GAs process is the fitness function, this word is taken from the EA theory, it is the quality function that verifies the quality of solution at

each iteration according to the problem to be solved, where every optimization problem has its fitness function.

Furthermore, the GAs will apply this quality function on the population, then a certain pool of calculated fitness values is generated, and the algorithm will start implementing the evolution operators: selection, crossover and mutation.

In the selection, the GA will use the parent selection mechanism based on their quality. Generally, the highest fitness individuals have a greater chance to become parents of the next generation, these parents of chromosomes will help reproduce a new population (offsprings). Next, a fundamental phase will take the offsprings and make a crossover operator, which is explorative, selecting a random crossover point(s) in the range of this point(s) the genes are exchanged and mixed between parents to generate the children's chromosomes.

The mutation operator is exploitative, it is executed on each new offspring, but the probability to mutate a gene is small, the purpose of mutation is to address a high level of diversity, it switches the bit of every gene according to the mutation points range to a randomly chosen neighboring solution.

Thus, as described in *Figure.13* the fitness function will be applied on the new offsprings, then a survivor selection mechanism is implemented, the offsprings will compete with the old individuals, based on their quality values to guarantee a place in the next generation. This whole cycle is repeated, there are cases when it is stopped when the GA reaches the given optimal objective function, but in other situations, the GA process may never stop, as a consequence that is an iterative algorithm and its condition may never get satisfied.

B. Swarm Intelligence Algorithms

Swarm Intelligence(SI) **(Beni & Wang, 1989),** is a population-based metaheuristic branch which has a great popularity because of their flexibility and diversity, they increased their ability to adapt to external differences and self-learning, it refers to one of the most important methodology that swarms use to achieving a common objective, known as the stigmergy, that means communication via cues or signs placed in the environment by one entity, which influence the behavior of other entities encountering them to help perform multiple tasks smartly, under the

stigmergy mechanism we have many effective functions that swarm can make in a cooperation such as the natural construction of homes like ant nests or bees hives, or the foraging techniques that exploits good food sources, adding to that the flocking activities which help traveling in groups, flocks, swarms..etc, they are aligned separated and cohensied at the same time.

Therefore, as we have complex real world problems, these interesting natural behaviors made an impact on computer science. Thus, **Beni & Wang, (1989)** inspired from these collective swarm intelligence activities to reach semi-optimal solutions of NP-hard problems, especially in a short period of time, *Figure.15* **(Brezočnik et al., 2018)** illustrate the general SIs process.

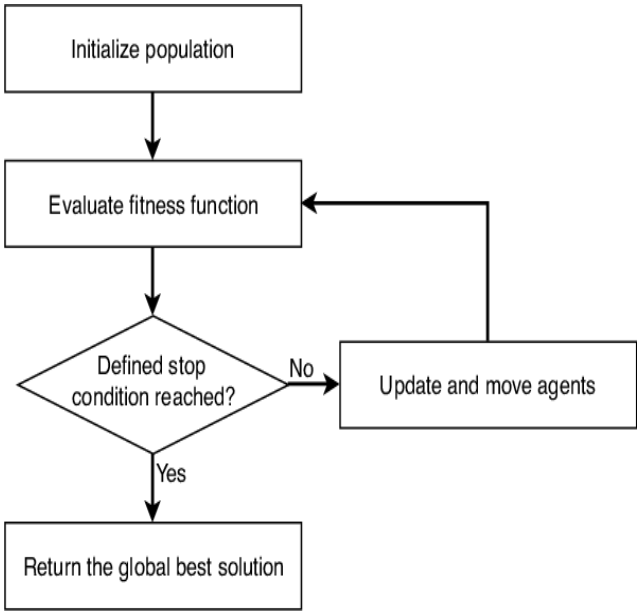


Figure.15: The general process of the Swarm Intelligence branch.

In the next two sections, we will take a look at two different successful algorithms that are inspired from the swarm intelligence framework, the first is the well known Particle Swarm Algorithm(PSO) which is based on the flocking and swarming behavior of birds, and the second is the Bee Swarm Algorithm(BSO) which refer as its name to the bee foraging activity.

1. Particle Swarm Optimization Algorithm

The Particle Swarm Optimization(PSO) **(Kennedy & Eberhart, 1995)**, is an evolutionary computational technique inherited from the population-based stochastic optimization algorithms motivated by the intelligent collective activity of the natural flocking behavior developed from swarm intelligence.

The PSO started the process from the initialization of the population of random potential solutions known as particles which is the first step as presented in *Figure.16* **(Dessouky & Elrashidy, 2016)**. Thus, a swarm consists of *N* particles, these birds are represented in a vectorial form by their position *x* in the search space where they are all flying, similar to the natural flocking activity when searching for the only one piece of food in the search space where each particle dynamically adapts to the speed of its travel in line with its own flight experiences and those of its mates.

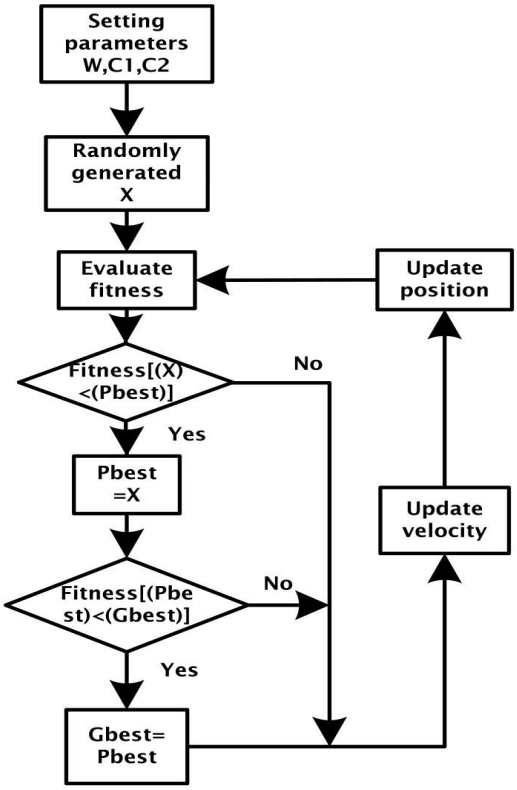


Figure.16: The diagram of the Particle Swarm Optimization Algorithm process

There are two main global values that help a particle track its path in the swarm to continuously update his position according to its previous experience and the experience of its neighbors, it needs to take these values on consideration in each movement, the personal best fitness value reached by this bird, known as the *Pbest*, and the second meter is the *Gbest* which signify that every particle in the swarm is informed about the best position of the best position that any particle has yet discovered, it is stored in a shared vector between all agents.

The next agent parameter is the speed(velocity), which is an essential parameter in the SIs strategy, as a reason that it is a relocation vector that will affect the position of the bird in the next iteration during the search phase, this value is calculated based on *Rule.24*:

Rule.24: $Vi+1 = w \times Vi + c1 \times rand(Pbest - Xi) + c2 \times rand(Gbest - Xi)$

that helps to calculate the new position following *Rule.25*:

Rule.25: $Xi+1 = Xi + Vi$

In summary, a particle has three vectors, the position vector, the velocity vector and the *Pbest* vector, and there are other supplementary parameters used in the velocity count like the inertial weight *w* regarded to the velocity, two accelerations coefficients *c1*, *c2* and two random numbers as *Rule.24* illustrated.

Let's go back to the diagram shown in *Figure.16*, after the swarm initialization of the position and velocity, the evaluation of the population starts with the desired fitness function. This procedure will compare the current fitness value of position with the fitness values of the randomly initial meters of *Pbest* and *Gbest*, and select the fittest value to update with. Then test if the obtained new *Gbest* and *Pbest* are satisfying the termination condition, otherwise it continues the cycle by updating the velocity with *Rule.24*, and then updating the position with *Rule.25*. A new particle is generated and all the steps of this section are repeated until the termination condition is satisfied and outputs the *Gbest* as the solution found.

One of the popular features of the PSO algorithm is its simplicity, and it is different from the GAs as it doesn’t make any kind of selection.

2. Bee Swarm Optimization Algorithm

The Bee Swarm Optimization(BSO) **(Drias et al., 2005)**, is a population based metaheuristic algorithm, which is a member of the swarm intelligence branch, on the top of that, it is one of the favored algorithms that are used in difficult optimization problems, for the reason that is inspired from the bee colonies foraging behavior, this smart strategy made this algorithm useful in various domains.

The process of the BSO starts with the generation of the first reference solution named *Sref* which is similar to the initial population solution in PbM, where a bee called *BeeInit* is responsible to use heuristic techniques to collect the *Sref*, as it is the starting point

it must contain a beneficial parameters, from the defined *Sref* a certain methods are applied to choose and find the supplementary solutions, then all of these solutions including the *Sref* are combined to form the *SearchArea* or also known as the search space.

The *SearchArea* solutions are placed based on an equivalent distance from the *Sref*, according to the flip metric which helps to decide the most converged solutions in the *SearchArea*, the primary phase is finished. Next, the content of the *SearchArea* is splitted where each bee will take a solution to begin the local search phase, then these agents will store the searching result in a table called the *DanceTable* as the famous bees waggle dance, the worker bees can address this table to know about others solutions. In the last step the BSO will select the fittest solution as the new *Sref* from the *DanceTable* and the cycle is repeated and will stop based on the given constraints such as the max iterations or the quality of the food sources reached during the process, the pseudo-code of the BSO is presented in *Figure.17* **(Drias et al., 2005)**.

```
begin
  Let Sref be the solution found by BeeInit;
    While    not condition of stop do
     begin
        insert Sref in taboo list;
              determine SearchArea from Sref;
         affect a solution of SearchArea to each bee;
        for each Bee K do
           begin
               search starting with the solution
                    affected to it;
               store the result in the table Dance;
            end;
        Choose the new solution of reference Sref;
      end;
  end;
```

Figure.17: The Pseudo-Code of the Bee Swarm Optimization Algorithm process

The BSO is different from the other population based methodologies because all the improvements made helps to explain the bees foraging activity deeply, where every bee has its own mission.

V. Conclusion

In this chapter, we presented an efficient approach that gives a global optimization solution to the NP-hard optimization problems, the metaheuristics address the limitation and complexity of the exact and heuristic algorithms, since our study concentrate on the uncertain big data field, the

solution search time needs to be decreased the most, this why we focused and gives more attention to the population-based metaheuristics(PbMs).

We give an overview on two of the well known PbM branches, EAs and SIs algorithms, then we introduced their proficient classes such as the Genetic Algorithm(GA), the Particle Swarm Optimization(PSO) and the Bee Swarm Optimization(BSO), all of the mentioned strategies will help us to generate our proposed solution which is described in the next chapter.

CHAPTER IV:

THE PROPOSED APPROACHES

I. Introduction

Based on the information we provided in the listed above chapters, the difficulty to mine frequent patterns from uncertain big datasets can depend on two major problems which are the big data and the algorithm search space, where almost all the previous researchers work tried to address one of the factors which is generally the big data problem using different techniques and ideas. Adding to that, this factor complexity grows up with time which means the old proposed solutions performance may be decreased, so new solutions need to be developed to handle the problem.

Therefore, we proposed three frameworks that combine clustering and distributed parallelism using a big data paradigm which is the MapReduce with three famous efficient bio-inspired metaheuristic algorithms known as the particle swarm optimization (PSO) **(Kennedy & Eberhart, 1995)**, the bee swarm optimization (BSO) **(Drias et al., 2005)** and the genetic algorithm (GA) **(Holland, 1984**), to improve both factors at the same time.

This chapter contains several parts starting with the proposed approaches, it takes in the first part our basic idea including the common point between the proposed algorithms which is the MapReduce job that calculates the expected support MRExpSup. Next, the implementations of each proposed approach including all the applied improvements are introduced.

II. Proposed approaches

We already expressed the two main problems that occur during the UFIM process for big data. As an inspiration from that, the Big Data problem has a unique resolving method that is shared between all the proposed frameworks, where the mentioned solution is previewed in this section.

Let's take a look at the general main pseudo-code which is illustrated in *Figure.18*. The program is a set of multiple steps, it is globally splitted into three major phases, the initialization, the fitness function calculation and the algorithms jobs. All the given steps are dependent on the framework itself, except the *MapReduce#1* defined in *step.4.1*.

Algorithm 1. General Metaheuristic based UFIM

Global main pseudo-code

1. ***Begin***
2. *Generate the initial population*
3. *Initiate the control parameters*
4. ***while*** *(number of Iterations < MaxIter)*
   - 4.1. ***Start Fitness Evaluation (call Expected Support MapReduce#1)***
   - 4.2. *Start Frameworks jobs(PSO, BSO, GA)*
   - 4.3. *Next iteration until the stopping criterion is met*

   end ***while***
5. *Display the Frequent Itemsets file*
6. ***End***

Figure.18: The proposed main pseudo-code.

A. MRExpSup solution *(MapReduce#1)*

Before starting explanation of the job, the initial *MapReduce#1* is referred to the fitness function calculation, it is designed to handle the first issue, which is the big data problem, because our problematic objective function is the count of the expected support which needs to get the necessary inputs from the dataset. And for each individual, each line of the data is scanned multiple times, as we said before, making the process more time and material consuming.

Due to this, our principal concept was to split our transactional database in a distributed cluster based on the number of machines into mappers, giving this database to the map function as an input, to count the expected support of all individuals of the population and output it in the reduce function, this operation differs from the traditional expected support calculation method as a reason that it doesn't use any sequential operations, it works only in parallel as presented in *Figure.19*.

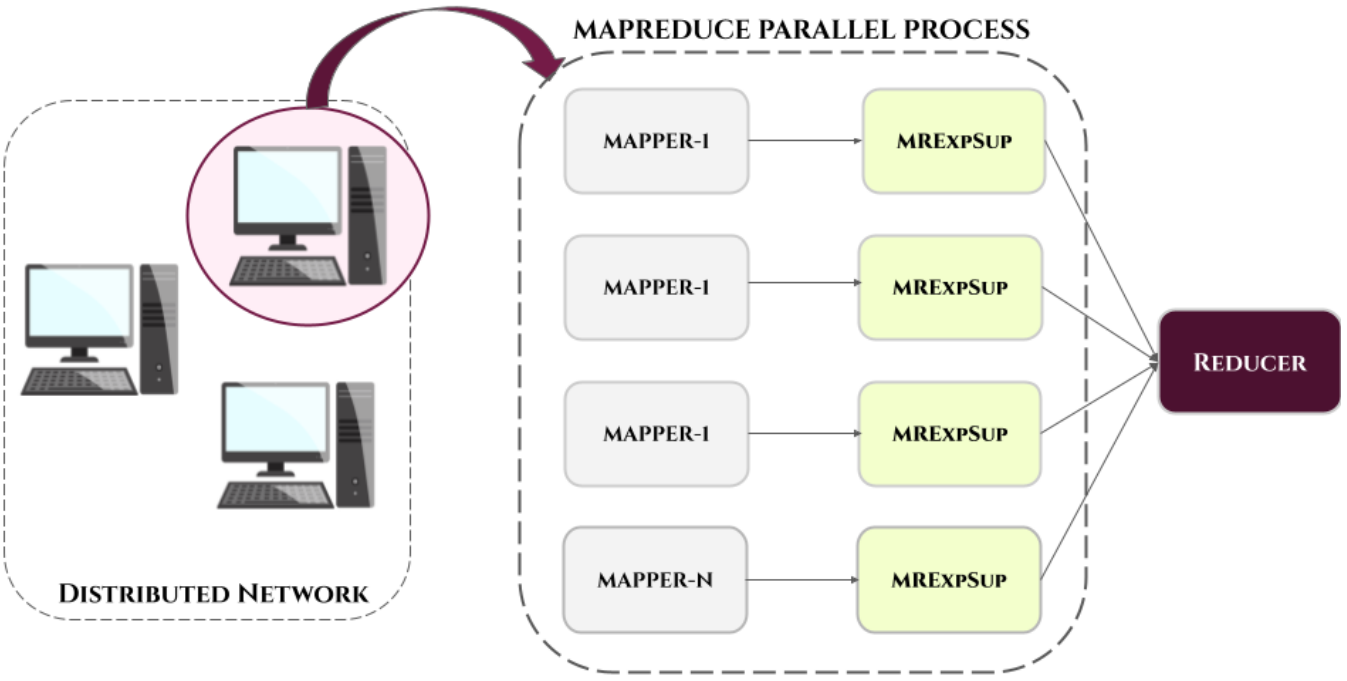


Figure.19: The schematic representation of the distributed cluster.

In essence, we have a tuple of parallelization procedures inside the MRExpSup method, the hadoop MR which is already a parallel distributed framework, and parallelization of the population and the database transactions as summarized in *Figure.*20.

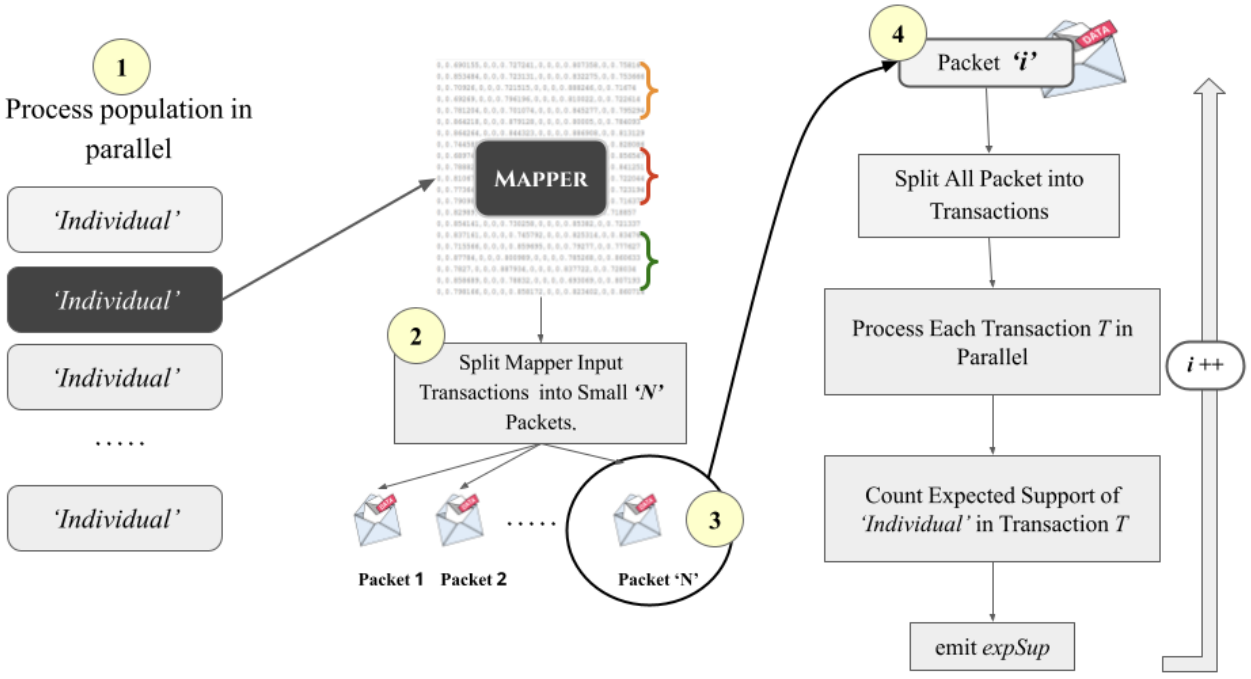


Figure.20: The schematic representation of the MRExpSup solution function.

The population is first splitted into individuals sets, where each individual is handled in parallel, secondary this individual will split all the input context of a specific mapper into '*N*' packets of transactions instead of one at a time, this maximum number of splits *'N'* of a mapper transactions, it is calculated using *Rule.26*, where *S* is a user defined splittable value.

Rule.26: $$N = \frac{maxBlockSize \times numberOfLines}{dataSize \times S}$$

After that, this set of transactions are prepared in parallel, where they are splitted and each record is maintained in parallel as an array of probabilistic values, then since we used the binary representation, it takes the given individual and calculate its objective function based on the comparison between its bits, if the individual i^{th} bit is equal to 1, which proves that this item is presented in this transaction, we take its i^{th} value in the created probabilistic record values array and apply *Rule.2* declared in the first chapter.

Thus, after all the individual bits are processed, the individual obtained *expSup* value becomes an output that is temporarily stored in the HDFS which then sent to the combiner. Whereas, as we said before all the data lines are processed in parallel and distributed over the machines.

Before starting the reducer presented in *Figure.19,* we go through a combiner process to organize the map output records that have the same key, then the reducer will take this output as its input and examine the sum of the set of all its expected support values as mentioned in *Rule.3*. At the end, the reducer writes to the HDFS the final *expSup* value of this particular individual.

Note that this MapReduce job remains the same for all the algorithms, because they share the same problematic and fitness function, this is why it is mentioned in this part not in the algorithms one, because it is a common point between them and we don't need to repeat it three times. In the next parts we will highlight the different proposed frameworks, where each of them has its own procedure.

III. MRPSO-UFIM framework

This framework supposed that each limitation is a MapReduce job. Let's take a look at its general MRPSO-UFIM main pseudo-code which is illustrated in *Figure.21*.

Algorithm 3. MRPSO-UFIM

The proposed solution main pseudo-code

1. ***Begin***
2. *Generate the initial population*
3. *Initiate the PSO control parameters*
4. *Store population in HDFS*
5. ***while*** *(number of Iterations < MaxIter)*
   - 5.1. *Start Fitness Evaluation (call Expected Support **MapReduce#1**)*
   - 5.2. *Combine the MapReduce#1 output with the population in HDFS*
   - 5.3. *Check the range of velocities*
   - 5.4. *Start PSO Job (call **MapReduce#2**)*
   - 5.5. *Next iteration until the stopping criterion is met*

   end ***while***
6. *Display the Frequent Itemsets file*
7. ***End***

Figure.21: The proposed MRPSO-UFIM pseudo-code.

From *Figure.21* we have *step.2* and *step.3* which are considered as a first basic phase of the main program. The generation of the initial population differs from an algorithm to another, the common point is that the individuals are a group of frequent patterns in order to avoid the random generation that leads to non-optimal results, they are presented in the same binary format, an array that is composed of 1 or 0 values, its dimension is the same as the dataset length, these group of frequent items is generated using a heuristic defined method. The first line of the database is token, then for each item if it differs from zero, we set its correspondence array column to 1, if not it remains 0 in the array, until we get the final array that will use recursivity to extract all the possible matching pairs of individuals, as illustrated in Figure.22.

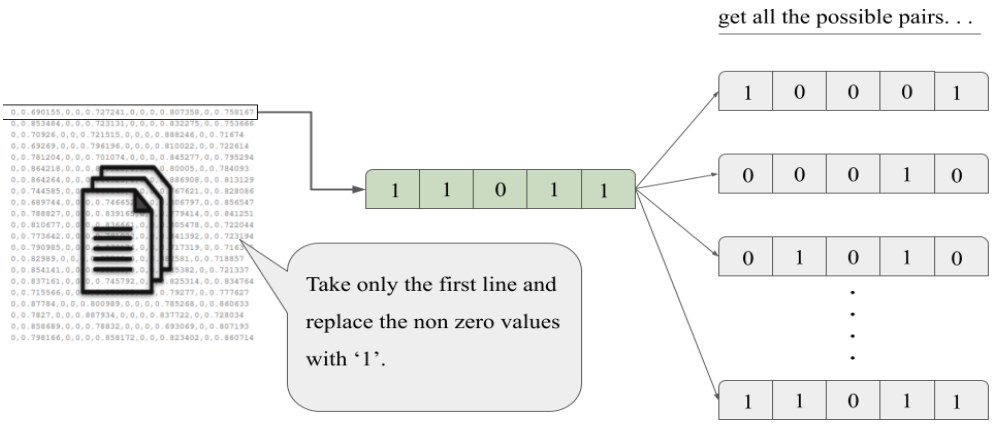


Figure.22: Generating initial population via heuristic method.

As we know a swarm is a collection of particles, assuming that each particle contains several parameters such as its *position* array, which is a binary representation as shown in *Figure.22*, its *velocity* array that is generated randomly in a given interval, both *Pbest* and *Gbest* arrays are arrays of zeros, where *Pbest* and *Gbest* values are set to 0.

When *step.3* starts, it has a significant impact on the outcome of the operation, because convergence behavior may start occurring if wrong initial control parameters are set. Therefore, to be specific there are three major attributes that we need to choose : the acceleration coefficients *c1* and *c2*, the inertia weight *w*, the random values *r1* and *r2*.

After *step.3* is finished, we store the generated population into the Hadoop Distributed File System(HDFS), where this phase is necessary for both algorithms because they use the population file in their work. Then, *step.5* begin, which describes a while loop that defines our stopping criterion, we set a maximum number of iterations(*MaxIter*) and while the iteration is smaller than the *MaxIter*, we enter to the second algorithm phase that contains the MapReduce expected support job(*step.5.1*).

When the already defined MapReduce#1 is done, we move on to *step.5.2* of the main code, we combine here the *expSup* value for each individual with the population file, and restore it again in the HDFS. Moreover, *step.5.3* is processed before calling the MapReduce#2, we need to make some changes. To further understand, PSO early convergence behavior happens when the search process is trapped in a local optimum, where the position of the new particle doesn't change during the time, or it stucks into two positions as pictured in *Figure.23*.

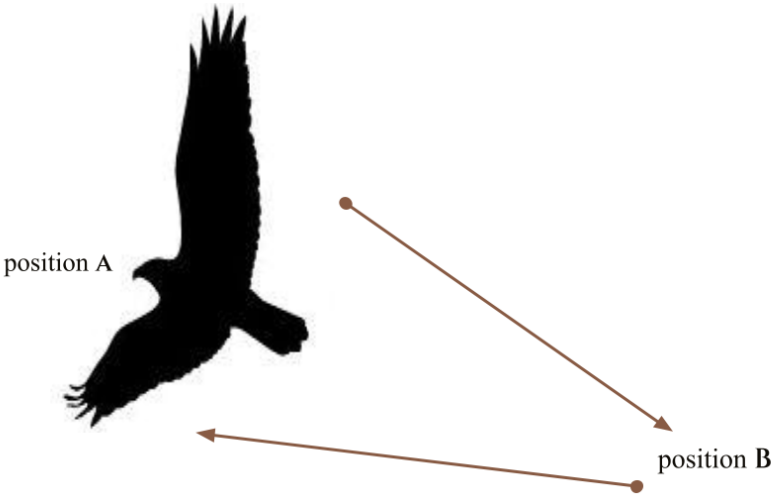


Figure.23: A particle local optimum situation.

Therefore, PSO has a lot of parameters that we've already mentioned, where each of them has a special ethnicity on the algorithm. In our study we made a lot of tests in order to obtain more

valuable results and to know which enhancements are more efficient to our problem, and from that we decided to fix the control parameters *r2*, *c1*, *c2*.

Thus, for the remaining two parameters *w* and *r1*, we suggested to use the well known dynamic parameterization technique **(Khan et al., 2017)**, at each iteration the *w* parameter is changed, to minimize the exceeded velocity as it had a significant effect on the value of the new velocity.

In each iteration we check if the minimum proposed velocity is upper to the obtained max or lower to the obtained min, if it is the first case then we minimize the *w* value to reach the minimum one using a decrease equation presented below in *Rule.27*.

$$\textbf{\textit{Rule.27:}} \quad w = maxW - \frac{(maxW - minW) \times (Iter \bmod NumSeq)}{D}$$

Where *maxW*, *minW, Iter, NumSeq and* D are representing respectively the maximum weight, the minimum weight, the current iteration, the length of the proposed sequence of decreased values and a value that specifies the change of numbers between iterations.

Furthermore, if it is equal to the minimum, the *w* is changed directly to the maximum, for the second case we flip the *Rule.27* in order to decrease we increase..etc and the same operation is repeated during the time.

Note that for the *r1* parameter, since is related to the particle itself, it is modified in the *Step.5.3,* which is designed to fix the second problem, the search space problem, it is solved with another MapReduce job, by using the PSO algorithm, where the output of the reducer is the global frequent itemset in the current iteration, the total file of frequent itemsets is collected in a different manner, let's perceive the MRPSO solution.

A. MRPSO solution *(MapReduce#2)*

The main purpose of the MRPSO is to extract frequent patterns through two functions which are the map and reduce function. According to the number of machines, the operation of the second job is splitted into subtasks launched in parallel in the same manner of the MRExpSup idea, where each one is given to each mapper of the slave node, the following *Figure.24*, represents the basic steps that performs the *MapReduce#2*. Now in the next explanation, we will give an overview of our mapper and reducer contents.

Algorithm 4. MapReduce#2
PSO process: Extract frequent itemsets

Input*:The population file*
Output*: Gbest Coordination*

- *Mapper(key, value):*
  1. *Split the value into valuesArray*
  2. *extract position*, *velocity*, *PbestArray*, *Pbest*, *GbestArray*, *Gbest* and *expSup from valuesArray*
  3. ***if****(expSup>=Threshold)*
     if*(expSup>=Pbest)*
     Pbest=expSup; PbestArray=position
     end ***if***
     if*(Pbest>=Gbest)*
     Gbest=Pbest; GbestArray=PbestArray
     end ***if***
     Store in FIM
     end ***if***
  4. *get r1 value of particle*
  5. ***while****(iter<MaxlocalItr)*
     5.1. ***for*** *bit in velocity*
     update velocity via Rule.24; update sigmoid via Rule.27
     end ***for***
     5.2. *get the maxVelocity, minVelocity, maxSigmoid and minSigmoid values*
     5.3. *get r3 via Rule.28*
     5.4. ***for*** *bit in position*
     if(r3[bit] >= sigmoidArray[bit]) then position[bit]=0
     else position[bit]=1
     end ***for***
     5.5. ***if****(existedPositionInBlackList(NewPosition))*
     NewPosition = FlipPosition(NewPosition)
     if*(OldPosition==NewPosition)*
     update r1 via Rule.30
     iter++
     else *break;*
     end ***ifElse***
     5.6. ***else***
     if*(OldPosition==NewPosition)*
     update r1 via Rule.30
     iter++
     else *break;*
     end ***ifElse***
     end ***ifElse***
     end ***while***
  6. *update population file*
  7. *store r1, iter, w, OldPosition, and NewPosition in PARAM file*
  8. *emit(key, GbestCoordination)*

- *Reducer(key, GbestCoordination)*
  1. *Split the GbestCoordination into valuesArray*
  2. *extract from the valuesArray the maximum Gbest value and the ArrayOfMaxValue*
  3. *update population file*
  4. *emit(key, MaxCoordination)*

Figure.24: The proposed PSO MapReduce solution pseudo-code.

1. MRPSO mapper function

A map process is a collection of key, value pairs, in this function the value is a line of the combine file that contain all the necessary information of a particle for each subtask (*step.1*), and to be able to perform the PSO steps we need to extract all the useful terms, we have two types of parameters, the static and the dynamic during this map operation. The fixed values such as the *w*, since it is not modified here*, c1*, *c2*, *r2* and the user defined minimum support named as *Threshold*, are obtained from the first initialization in *step.3* of main code in *Figure.24*.

For the rest of the dynamic variables in other words, the *position*, *velocity*, *PbestArray*, *Pbest*, *GbestArray*, *Gbest* and *expSup* are extracted from the value pair of the map object as presented in *step.2*.

As a first algorithm phase, the fitness function is already employed in the MRExpSup job, then in *step.3*, we compare the *expSup* value of the particle with the *Threshold* value, only if it is satisfied, and when this particle achieve its best solution in the search space or it equals to its best previous position, we apply the process that update the *PbestArray* and the *Pbest* by the *position* and the *expSup* respectively. After that, the algorithm starts updating the particle global best, both *Gbest* arguments are replaced with the *Pbest* fields, uniquely when the *Pbest* is greater than or equal to *Gbest*.

Since our goal is the maximization, there is an additional action in this step, which is to store in the HDFS the *position* and the *expSup* of this current particle in a shared file called FIM, which refer to frequent itemsets mining, this file will maximize the results and to not consider only the highest *Gbest* as the final algorithm results. In contrast, if the frequency rule is not satisfied, the above stages will be ignored.

The functionalities left are the *velocity* and *position* updates, these are simple PSO steps, but in our case, there are some changes and improvements, to understand that we will take this phase line by line.

We will give a particle the chance to eliminate the convergence by applying the idea of chances.

Then, the new *position* is calculated using the current *velocity* according to *Rule.24*. And since the output of the i^{th} *velocity* value is binary. Therefore, as indicated in *step.5.1,* we use the sigmoid function equation defined in *Rule.28* **(Kennedy & Eberhart, 1997)** :

Rule.28: $$sigmoidArray[i] = \frac{1}{1+e^{-velocity[i]}}$$

This sigmoid value is compared with a new variable named *r3*, and it is generated randomly following *Rule.29*:

Rule.29: $$r3[i] = rangeMin + (rangeMax - rangeMin) \times randDouble()$$

Where *rangeMin* is initiated using *Rule.30*, the *rangeMax* is the maximum value of the *sigmoidArray*, we didn't take its minimum value, the reason is that the uncertain datasets are almost fully of zeros, we don't need a *position* that is rich of ones to avoid multiplication with zero values, the *U* is a value that helps augmenting *rangeMin*.

Rule.30: $$rangeMin = rangeMax - \frac{rangeMax - rangeMin}{U}$$

After updating *position* by doing a comparison between the *r3* and the *sigmoid* arrays(*step.5.4*), we will check the presence of this obtained *position* in a file that contain all the already obtained positions to avoid the problem presented in *Figure.23*, if it is true, we will flip a bit in the new position and check similarity with the previous one, if it is true and they are the same, which means that the particle didn't change its position yet in early stages, the r1 of this particle is updated using *Rule.31*, it is inspired from the sinusoidal wave equation, where *U* helps to obtain the user wanted dynamic sequence:

Rule.31: $$r1 = \left| A \times Sin(2\Pi \times iter \times U + \phi) \right|$$

Let's explain our overall contribution to this phase, when the local *iter* augments, this leads to restart the *step.5* processes again with a modified *r1* value for this particle, until we get a new different *position* from the old one. As shown in *step.6, a*ll the used individual information is updated in the population file: the *position*, *velocity*, *PbestArray*, *Pbest*, *GbestArray* and *Gbest* in the HDFS.

Finally, when *step.8* is executed, the current GbestArray and Gbest coordination are sent to the reduce function, to connect the particles with these values.

2. MRPSO reduce function

As we spoke earlier that before starting the reduce function we must pass to the combiner function to order the output of the mapper function and emit it to the reduce function to get both key and value as input, whereas in the commencing we extract the parameters from the input value which are all the *GbestArray's* and their *Gbest* values, we only need to compare the values of the *Gbest* among themselves to outcome the greater one in the swarm. This reduce object facilitates the communication between particles, because when it is finished and the highest value is selected, we update the *GbestArray* and its value in the population file with the one generated from the reducer output for all the particles.

The aforementioned operations presented in *step.5* of *Figure.21*, are repeated based on the maximum number of iterations. Thereafter, when the stopping criterion is met, which leads to the end of all the jobs, we can extract our results by displaying all the obtained frequent itemsets from our FIM file that is located in the HDFS.

IV. MRBSO-UFIM framework

This MRBSO uncertain frequent pattern mining framework is different from the others, its essence can be resumed as it uses only one MapReduce that is MRExpSup as opposed to MRPSO or MRGA which require at least one or more additional MR functionalities. *Figure.25* introduces the different MRBSO-UFIM main steps and we can notice that all its processes are made in this main code.

Algorithm 5. MRBSO-UFIM

The proposed solution main pseudo-code

```
1.  Begin
2.  Generate the initial Sref
3.  Initiate the BSO control parameters
4.  Store Sref in the Taboo list
5.  while (number of Iterations < MaxIter)
    5.1.  while(MaxChances>0)
          5.1.1.  MaxChances - -
          5.1.2.  Determination of SearchAreaI, SearchAreaII(Figure.26)
          5.1.3.  Start Fitness Evaluation (call Expected Support MapReduce#1)
          5.1.4.  Store the generated max-bees in the Dance table
          5.1.5.  get the MaxChildbeeCoordination from the Dance table
          5.1.6.  if(MaxChildBeeValue > SrefValue)
                        if(MaxChildBeeValue NOT existed already in Taboo)
                              SrefValue=MaxChildBeeValue
                              SrefArray=MaxChildBeeArray
                        else
                              change the flip of the SearchAreaI
                        end ifElse
          5.1.7.  else
                        change the flip of the SearchAreaI
                  end ifElse
          end while
    5.2.  if(MaxChildBeeValue==SrefValue)
                  NewSrefCoordination=SelectDistantBee()
          end if
    5.3.  Store the NewSref in Taboo
    end while
6.  Display the Frequent Itemsets file
7.  End
```

Figure.25: The proposed MRBSO-UFIM pseudo-code

We will start generating the initial population, which refers in BSO to the *InitBee* as known as the *Sref*, it is composed of two parameters, its identifier array followed by its expected support value. Since we didn't start our mining then we need to identify a bee as the *Sref*, but for the maximization purpose we set the bee array and *expSup* to zeros.

After *step.02*, we move on to the next step which is the control parameters initialization (*step.3*). As we know, each algorithm contains information that greatly improves its work. In our case, we have three parameters such as the *MaxChance* which is a user-determined value, and based on our research the BSO algorithm contains one flip value, but in our solution, we proposed two *flip* values which are the *MotherFlip* and *ChildFlip* values. Next, when both *step.2* and *step.3* are done, we will store the *Sref* as the population in the HDFS.

Our algorithm starts the work to extract the frequent itemset for each iteration, and while there are more chances start the determination of the search areas based on the *Sref* to output a set of bees as presented in *Figure.26* using one of the impressive strategies called the *Next strategy* (Djenouri et al., 2014).

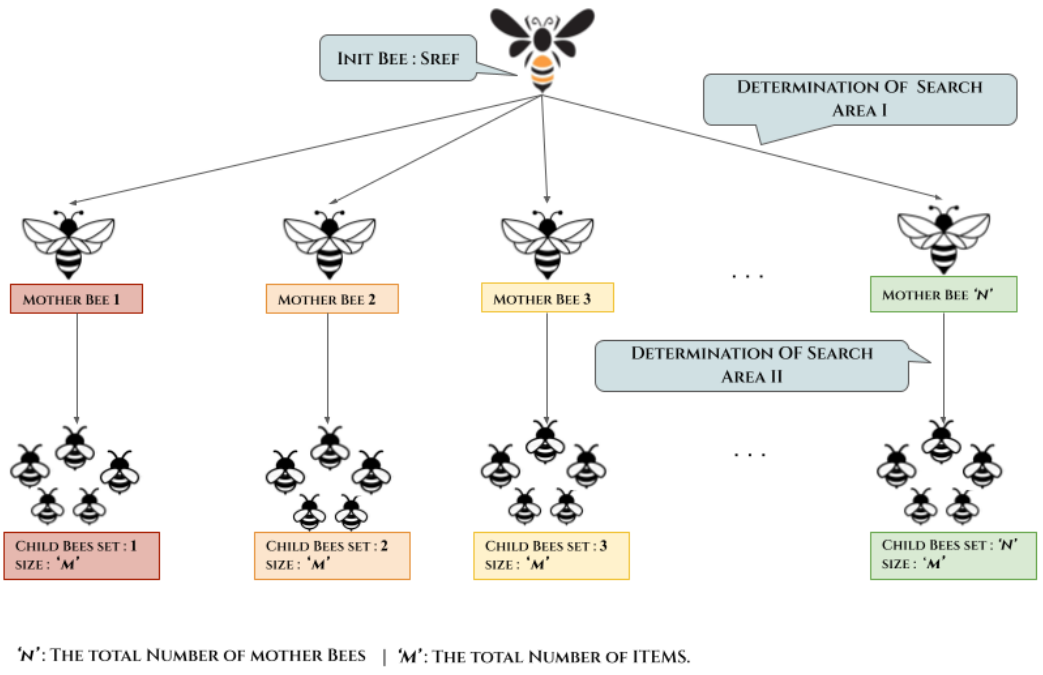


Figure.26: The Determination of the Search Area.

Now we possess the final search area of bees, we go to the next stage which is calculating their *expSup* values by calling *MapReduce#1*. Before applying *step.5.1.4* we will store the frequent bees that satisfy the frequency condition in the FIM file, then from each child bees sets of a specific mother bee we extract the maximum child coordination and store it in the dance table, and from these maximum children bees we select only the highest one as a representative bee '*MaxChildBee*' like written in *step.5.1.5*, where the main objective of this process is to let this representative bee be the new *Sref* of the swarm.

Then we will check if the *MaxChildBee* is valuable, if true, we have to do a quick check if it doesn't exists already in the *Taboo* list or not, we don't want to make a test with an already existed *Sref*, if true, this *MaxChildBee* become the new *Sref* for the next iteration. But, if it existed before or the *MaxChildBee* is smaller than the *Sref*, we will make a random modification on the *MotherFlip* and re-enter the second while loop again until the end of the *MaxChances*.

When our MRBSO satisfy *step.5.2*, this means that all the chances are finished and no *MaxChildBee* could take the place of the *Sref*, therefore, we will change the current *Sref* by taking the most distant *childBee* from the *Dance* table compared to the *Srefs* stored in the *Taboo* list, then replacing this *Sref* with the obtained *childBee* from the distant selection operation and store it in the *Taboo* list.

The process will be continued until the stopping criterion is met, and then we will be able to view all of our possible frequent bees collected by our MRBSO-UFIM.

V. MRGA-UFIM framework

The MRGA-UFIM algorithm general idea is similar to our first proposed framework the MRPSO-UFIM, we may found the same used techniques, where it contains two MR jobs, the first one is the *MapReduce#1*, that calculates the fitness function and the second job is for the MRGA process as figured in *Figure.27*.

Algorithm 6. MRGA-UFIM

The proposed solution main pseudo-code

1. ***Begin***
2. *Generate the initial population*
3. *Initiate the GA control parameters*
4. *Store population in HDFS*
5. ***while*** *(number of Iterations < MaxIter)*
   - 5.1. *Start Fitness Evaluation (call Expected Support **MapReduce#1**)*
   - 5.2. *get the list of max chromosomes : create parents*
   - 5.3. *Start GA Job (call **Map#3**)*
   - 5.4. *copy chromosomes to the population*
   - 5.5. *Next iteration until the stopping criterion is met*

   *end **while***
6. *Display the Frequent Itemsets file*
7. ***End***

Figure.27: The proposed MRGA-UFIM pseudo-code.

To generate the initial population(*step.2*) we need a set of initial chromosomes obtained from the same heuristic technique defined in *Figure.22*, furthermore, the GA require some specific parameters to be initiated in *step.3*, including the crossover type to define which type the user selected to perform the crossover operation, and the crossover points which refers to the parameters of each crossover operation.

We will skip *step.4*, *step.5* and *step.5.1* as long as they are identical to the presented steps in the first algorithm, when the *MapReduce#1* is finished, we extract frequent itemsets from the output of *MapReduce#1*, then we jump to the *step.5.2*, where it is an important phase that is composed of multiple operations as pictured in *Figure.28*, we have to combine, sort, trim and shuffle the old population with the new one to create a random sequence of pairs, these pairs will be examined as the new parents which are given next to the MRGA job as inputs.

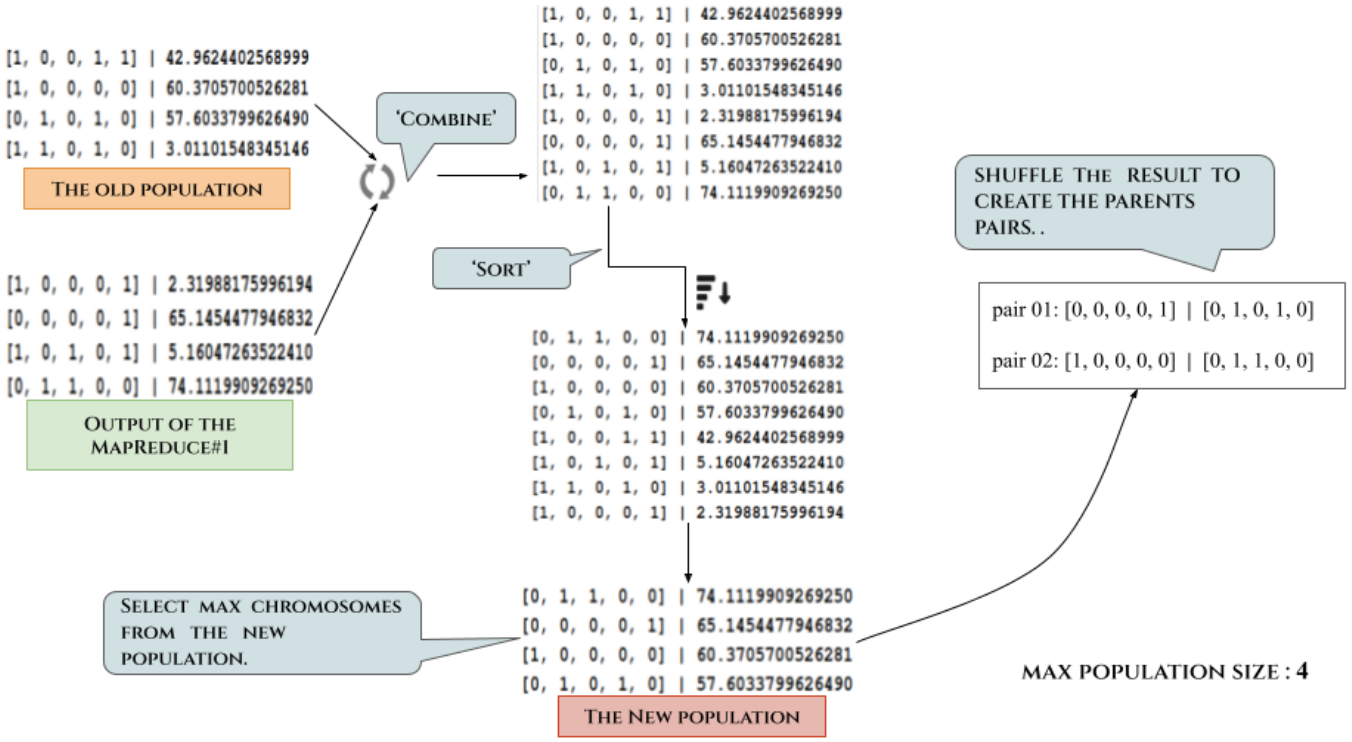


Figure.28: The basic steps for the creation of new parents.

Note that the MRGA is formed from only a mapper without a reducer, since there is no need for it, where this function is written in the pseudo-code defined in *Figure.29*.

Algorithm 7. Map#3

GA process: Generate new chromosomes

Input*: The parents file*

Output*: Set of new chromosomes*

- *Mapper(key, value):*
  1. *Split the value into valuesArray*
  2. *extract parent1 & parent2 from valuesArray*
  3. *get the crossoverType, crossPoints. . .*
  4. ***while****(chance<MaxlocalChances)*
     - 4.1. *offspring1, offspring2 = crossover(crossoverType, parent1, parent2, crossPoints..)*
     - 4.2. *offspring3, offspring4 = mutation(offspring1, offspring2)*
     - 4.3. *check existence of offspring1, offspring2, offspring3 and offspring4 in the population*
     - 4.4. ***if*** *(existence >= 3)*
       change the crossoverType
       chance ++
     - 4.5. ***else***
       emit(key, Non-Existed-Offsprings)

     end ***ifElse***

  end ***while***

Figure.29: The proposed GA Mapper solution pseudo-code.

A. MRGA solution*(Map#3)*

The main purpose of the *Map#3* is to produce new chromosomes through the mapper function, and as we said before, this job is also splitted into subtasks launched in parallel in the same manner of the MRPSO solution, we start by extracting the parents, the control parameters of the GA.

In the same manner of the Mapper PSO we give a maximum local number of chances to help exiting minimize the opportunity to get the same parents, in *step.4.1* a crossover function is performed, as it is known as an important variant of the GA, and we didn't want to limitate the transformation of chromosomes on one method. Therefore, we included the three famous crossover types functions, so according to the *crossoverType* variable, the algorithm will choose its corresponding crossover techniques that are employed such as the single point, the two points and the uniform crossover.

Step.4.2 shows that the first offsprings are ready to perform the mutation, after that we have to check if all the obtained offsprings(from the crossover and the mutation) are existed in the current population, if plus to three of them are presented there, which drop us in a bad situation, where the new offsprings are the same as the old ones, this case needs to be avoided giving a chance to the parents to re-generate new offsprings by changing the *crossoverType* to be a different from the last one, until they either change completely the offspring or there is no more chances left, remember that our goal is the maximization therefore we took all the offsprings as the new collected chromosomes not only the mutation ones.

Let's back to our global MRGA-UFIM algorithm presented in *Figure.27*, the algorithm will copy the output of the GA mapper to the population path stored in the HDFS, where a new set of chromosomes will be used in the next iteration *MapReduce#1*. All the remaining steps are previously expressed in the earlier sections.

VI. Conclusion

In this chapter, we presented our three proposed solutions the MRPSO-UFIM, the MRBSO-UFIM and the MRGA-UFIM they are based on a famous and efficient metaheuristic algorithms PSO, BSO and GA respectively, to mine from uncertain big datasets in the purpose of maximization. Furthermore, the selection of their parameters, tools and tests including the results are seen in the coming pages.

Chapter V:

Tests AND Validation

I. Introduction

In this chapter, the experience results of three based metaheuristic algorithms are compared to find out the best one among them to solve the UFPM from Big Data, based on enhancement we've discussed in the former chapter. We will initially talk about the environment and mechanisms, datasets used in this work, including all the performed tests based on different metrics and benchmarks.

II. Experimental Environment

A. Hardware Environment

All the experiments were performed on a computer configured as follows:

- Device : HP-Laptop.
- Processor : 4-Core 2.50GHz CPU.
- RAM : 7GB.
- OS : Ubuntu 20.04 64-bit.

B. Software Environment

Our work was implemented using the Apache Hadoop software (v-3.3.2), which is a set of open source frameworks, to solve big data problems, it provides distributed processing by using either a single or a set of computers in the same network, to perform a certain process locally. We employed the Hadoop MapReduce framework, using the Java programming language (JDK-11.0.16), which is developed via the Eclipse IDE platform.

The initial objective was to run our algorithms on a multi-nodes Hadoop cluster to make realistic tests on a rich of CPUs machines. Unfortunately, we had some external difficulties, where the biggest problem was the lack of the desirable materials and the search for backup solutions, all this caused us to delay our work. Therefore, we had to use the Hadoop single-node in our proper machines independently.

III. Experimental datasets and parameters

A. Presentation of the used Datasets

In the making of the performance evaluation of the proposed frameworks, we tried to vary between the categories of the dataset to almost examine all the important cases in order to not to exceed the time available to us for testing and validation.

Therefore, we acquired three databases of different characteristics, collected from the SPMF open-source data mining library (Fournier-Viger et al., 2016) and transformed with our generator to get the uncertain databases as presented in *Table.7*, the description of the different datasets used later in the experimentation, they are divided based on their coordination size.

Table.6: The used datasets in the experiments.

Datasets	Avg.Trans.Len.	#Items	#Trans	categorie
UBchess	37	75	3,196	small
UBMushroom	23	119	8,125	small
UBRecordLink	10	29	574,913	large

The studies made are evaluated relying on the database categories, the results are illustrated in the next sections, where for each dataset we vary in the algorithms parameters.

B. Parameters initialization

As we indicated in the previous chapter, there are a set of fixed control parameters that need to be identified and remain unchangeable for the whole testing cases, they affect the optimization algorithms results and selecting the best parameters is still challenging, it necessitates much time. The listed parameters below in *Table.7*, are given based on a lot of tests, generally this list helped our algorithms avoiding the local optimum problem.

Table.7: The fixed control parameters.

MRPSO-UFIM	*c1*	*c2*	*r2*
	0.8	0.8	0.25
MRGA-UFIM	*Single Point Crossover*	*Two Points Crossover*	*Uniform Crossover*
	3	[3, 3]	2
MRBSO-UFIM	*Child Flip*		
	-1		

For the rest of the global fixed parameters such as the initial population size, the minimum user threshold, ….etc. There is a terminal display that offers the user the ability to set its own parameters in each test made. As an overview we have *Figure.30* which shows the user terminal display of the MRPSO-UFIM algorithm, where the user selected the UBChess dataset, and has filled the number of iterations, the threshold percentage and the initial population size manually with 25, 50 and 100 respectively.

```
The MRPSO Uncertain Frequent pattern mining algorithm STARTED . . . .
----------------------------------------------------------------------------------
These are the available datasets :
1 : UBaccidents      , Items: 468  ,  lines: 340185
2 : UBPOWERC      , Items: 140  ,  lines: 1040002
3 : UBRecordLink      , Items: 29  ,  lines: 574913
4 : UBmushroom      , Items: 119  ,  lines: 8125
5 : UBPAMAP      , Items: 141  ,  lines: 1000002
6 : UBchess      , Items: 75  ,  lines: 3197
7 : UBSkin      , Items: 11  ,  lines: 245058
You need to select one from the listed above datasets to perform tests..
Enter the data name number : 6
----------------------------------------------------------------------------------
You chose the 'UBchess' data.
----------------------------------------------------------------------------------
You need to select the maximum number of iterations..
Enter the maximum iterations number : 25
----------------------------------------------------------------------------------
You need to select your minimum support value, to calculate the 'Threshold' you have to :
Enter your Threshold percentage % : 50
----------------------------------------------------------------------------------
You need also to select the population size..
Enter the population size between 10 and 500 : 100
----------------------------------------------------------------------------------
```

Figure.30: The MRPSO-UFIM program simulator.

IV. Results and Comparison

In the next sections we will preview the results of our proposed bio-inspired algorithms with the three used datasets UBChess, UBMushroom and UBRecordLink. Let's start testing them based on two examinations, which are depending on the threshold value, where Examination#1, Examination#2 are respectively for low and high minimum supports. Additionally, there are some cases when each of them contain two tests, each test is a variation of the initial population size.

Note that each test has its appropriate coordination that are summarized in the tables presented besides the plot's results of the obtained frequent itemsets and the total execution time based on the number of iterations when applying each algorithm, as pictured in the listed figures.

A. Small dataset, Few number of items

The best description for this benchmark is the UBchess dataset, where it contains the highest average size of items compared to the remaining ones. An uncertain transaction example of the first UBchess line is captured in *Figure.31*.

```
1  0.839165,0.690155,0,0.727241,0,0.807358,0,0.758167,0,0.853484,0,0.723131,0,0.832275,0,0.753666,0,0.70926,0,0.721515,0,
   0.888246,0,0.71674,0,0.69269,0,0.796196,0,0.810022,0,0.722614,0,0,0.781204,0,0.701074,0,0.845277,0,0.795294,0,0.864218
   ,0,0.879128,0,0.80005,0,0.784093,0,0.864264,0,0.844323,0,0.886908,0,0.813129,0,0.744585,0,0.860849,0,0.767621,0,0.8280
   86,0,0.689744,0,0.746652,0,0.806797,0,0.856547,0,0.788827
```

Figure.31: The representation of the UBChess first uncertain transaction.

We employed only the examinations without the two couple of tests, the initial population is fixed in the UBchess, where tests are illustrated in the coming figures.

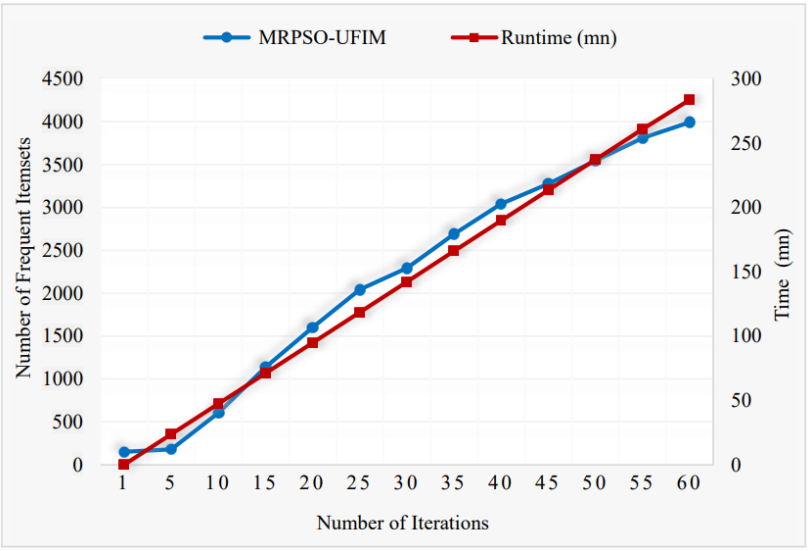


Figure.32: MRPSO obtained frequent itemsets, runtime based on the #iterations (Examination#1, Small).

Table.8: MRPSO Examination#1 coordination.(Small)

Dataset	UBchess
Algorithm	MRPSO-UFIM
Threshold	0.1
Initial Population	150
Iterations	60
Frequency Average	56
Total Frequent Itemsets	3994
Average Time (mn)	4.72
Total Time (mn)	283.69

Table.9: MRGA Examination#1 coordination (Small).

Dataset	UBchess
Algorithm	MRGA-UFIM
Threshold	0.1
Initial Population	150
Iterations	60
Frequency Average	165
Total Frequent Itemsets	9941
Average Time (mn)	3.1
Total Time (mn)	186.06

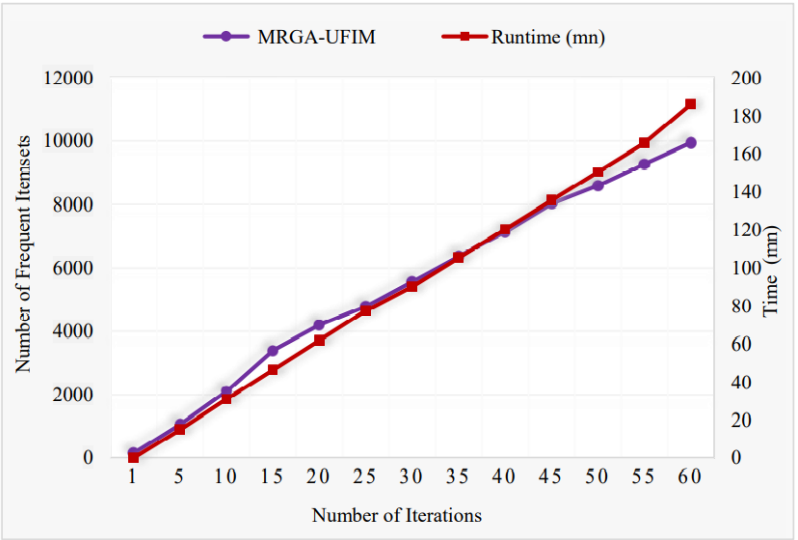


Figure.33: MRGA obtained frequent itemsets, runtime based on the #iterations (Examination#1, Small).

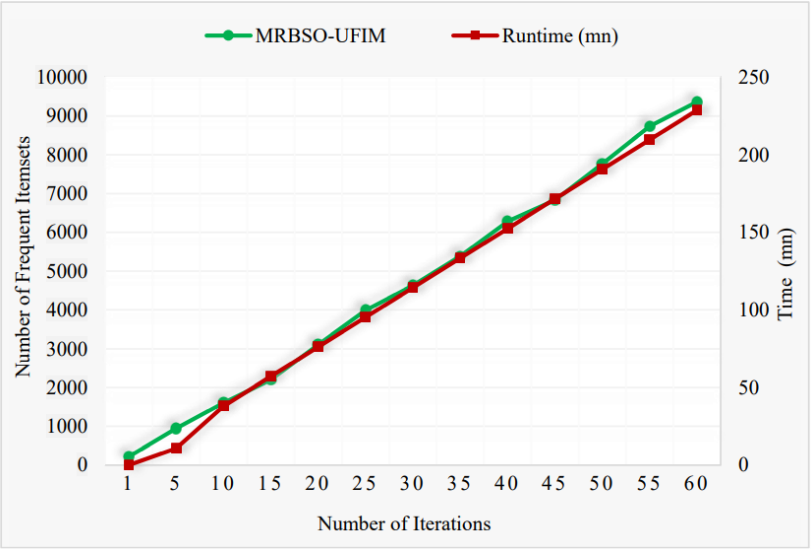


Figure.34: MRBSO obtained frequent itemsets, runtime based on the #iterations(Examination#1, Small).

Table.10: MRBSO Examination#1 coordination(Small).

Dataset	UBchess
Algorithm	MRBSO-UFIM
Threshold	0.1
Initial Population	Bee : 2 Chances : 2
Iterations	60
Frequency Average	156
Total Frequent Itemsets	9364
Average Time (mn)	3.81
Total Time (mn)	228.85

As we can see the number of the obtained itemsets increased at each iteration with the time, the approaches avoided stacking at the same solution.

Table.11: MRPSO Examination#2 coordination(Small).

Dataset	UBchess
Algorithm	MRPSO-UFIM
Threshold	50%
Initial Population	150
Iterations	60
Frequency Average	1
Total Frequent Itemsets	93
Average Time (mn)	3.37
Total Time (mn)	202.32

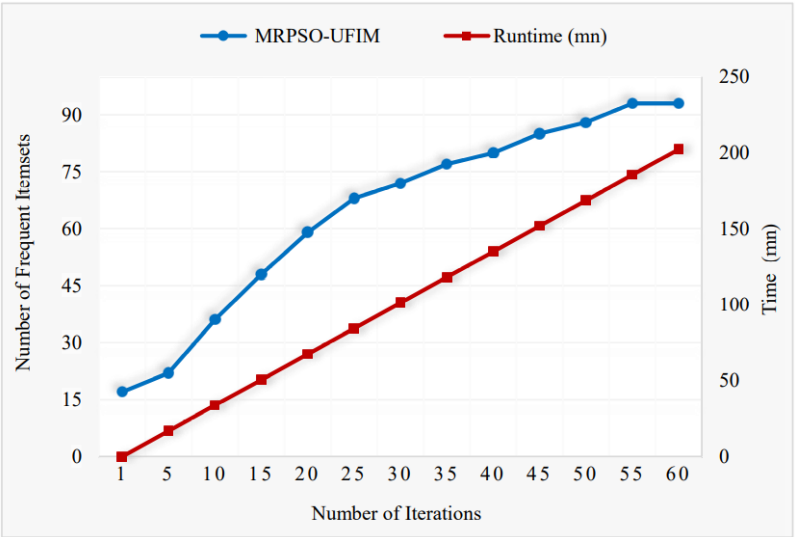


Figure.35: MRPSO obtained frequent itemsets, runtime based on the #iterations(Examination#2, Small).

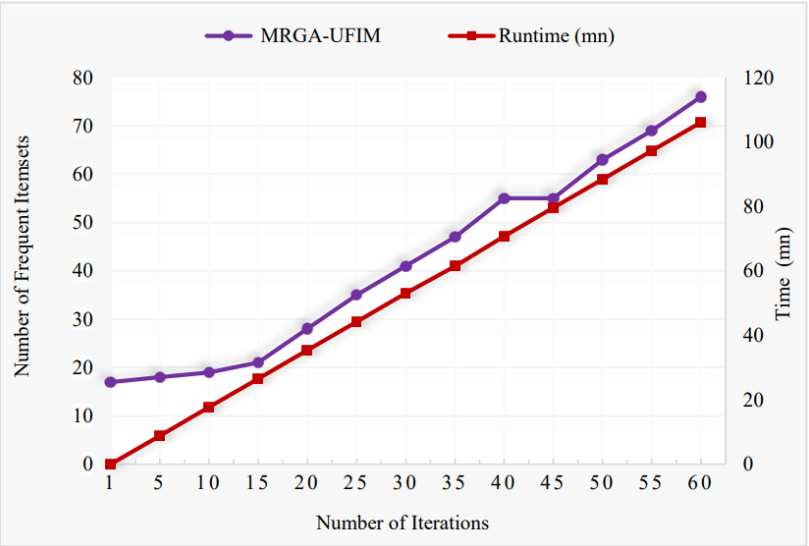


Figure.36: MRGA obtained frequent itemsets, runtime based on the #iterations(Examination#2, Small).

Table.12: MRGA Examination#2 coordination (Small).

Dataset	UBchess
Algorithm	MRGA-UFIM
Threshold	50%
Initial Population	150
Iterations	60
Frequency Average	1
Total Frequent Itemsets	76
Average Time (mn)	1.76
Total Time (mn)	106.12

Table.13: MRBSO Examination#2 coordination(Small).

Dataset	UBchess
Algorithm	MRBSO-UFIM
Threshold	50%
Initial Population	Bee : 2 Chances : 2
Iterations	60
Frequency Average	2
Total Frequent Itemsets	135
Average Time (mn)	4.41
Total Time (mn)	264.8

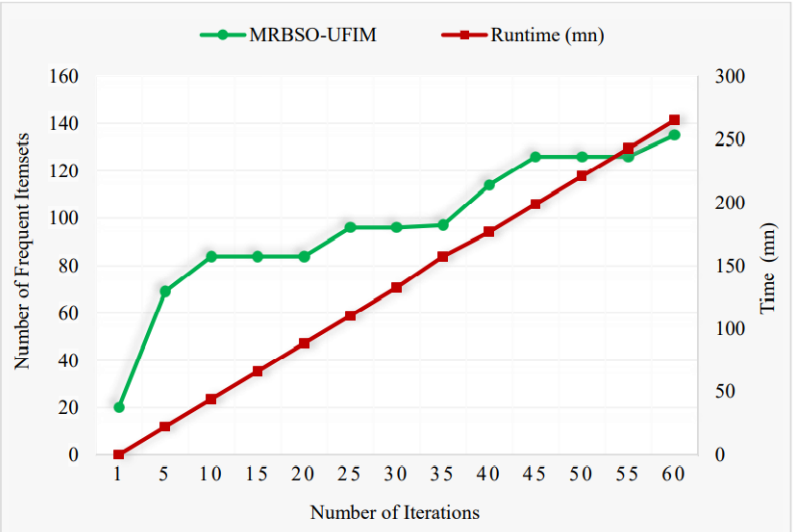


Figure.37: MRBSO obtained frequent itemsets, runtime based on the #iterations(Examination#2, Small)

We can notice that the MRBSO has successfully reached the total number of frequent itemsets with a threshold of 50%, compared to the others.

B. Small dataset, Average number of items

The second benchmark represents the UBMushroom dataset, it is a small dataset but with a significant number of items, all the results, coordination and tests made with datasets are shown in the following figures.

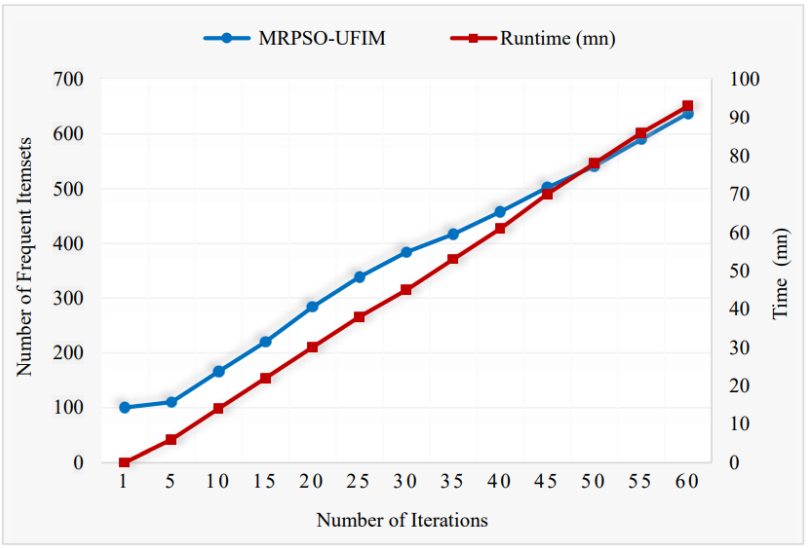


Figure.38: MRPSO obtained frequent itemsets, runtime based on the #iterations(Examination#1, Test#1, Small).

Table.14: MRPSO Examination#1, Test#1 coordination(Small).

Dataset	Mushroom
Algorithm	MRPSO-UFIM
Threshold	0.1
Initial Population	100
Iterations	60
Frequency Average	10
Total Frequent Itemsets	638
Average Time (mn)	1.5
Total Time (mn)	93

Table.15: MRGA Examination#1, Test#1 coordination (Small).

Dataset	Mushroom
Algorithm	MRGA-UFIM
Threshold	0.1
Initial Population	100
Iterations	60
Frequency Average	56
Total Frequent Itemsets	3356
Average Time (mn)	1.89
Total Time (mn)	113.8

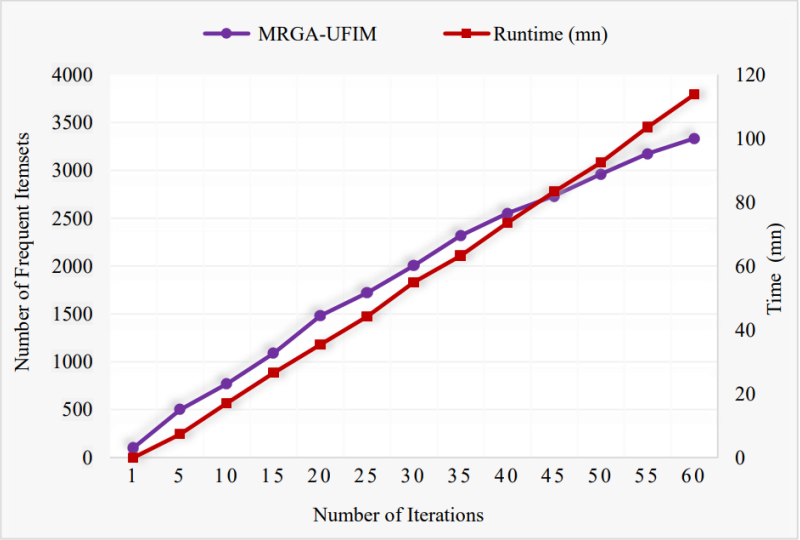


Figure.39: MRGA obtained frequent itemsets, runtime based on the #iterations(Examination#1, Test#1, Small).

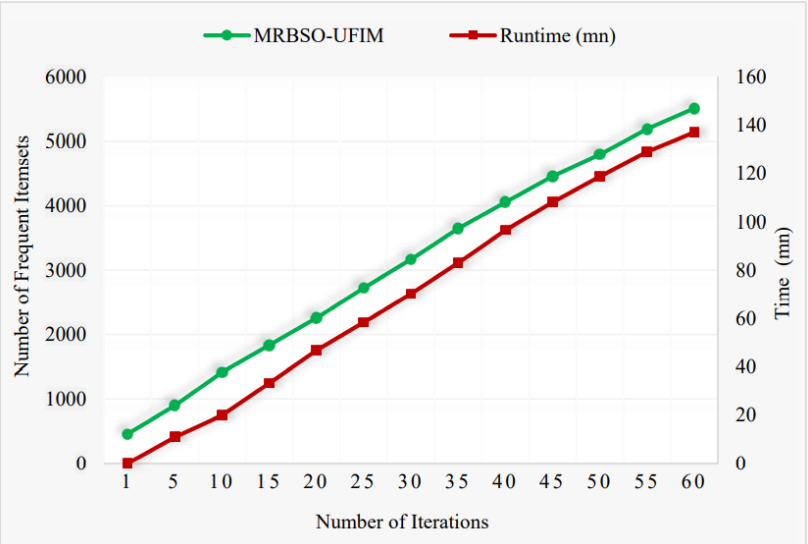


Figure.40: MRBSO obtained frequent itemsets, runtime based on the #iterations(Examination#1, Test#1, Small).

Table.16: MRBSO Examination#1, Test#1 coordination(Small).

Dataset	Mushroom
Algorithm	MRBSO-UFIM
Threshold	0.1
Initial Population	Bee : 1 Chances : 2
Iterations	60
Frequency Average	92
Total Frequent Itemsets	5512
Average Time (mn)	1.24
Total Time (mn)	137.13

Table.17: MRPSO Examination#1, Test#2 coordination (Small).

Dataset	Mushroom
Algorithm	MRPSO-UFIM
Threshold	0.1
Initial Population	200
Iterations	60
Frequency Average	20
Total Frequent Itemsets	1225
Average Time (mn)	2.84
Total Time (mn)	170.4

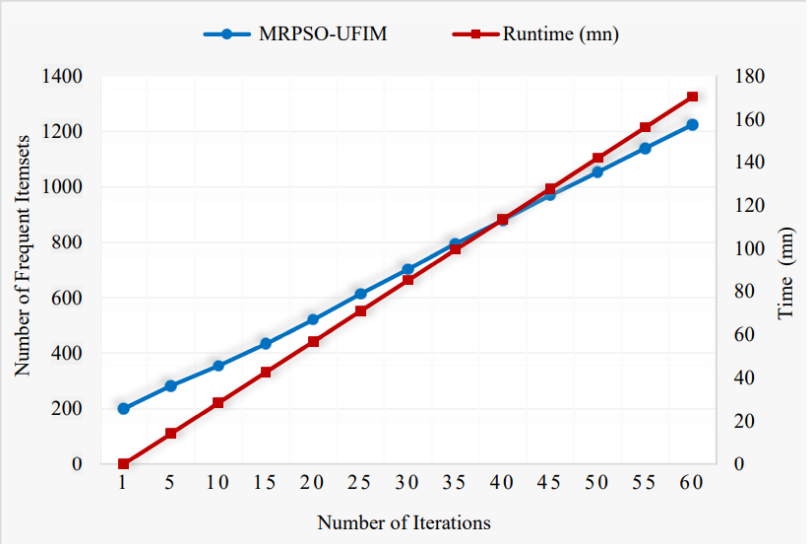


Figure.41: MRPSO obtained frequent itemsets, runtime based on the #iterations(Examination#1, Test#2, Small).

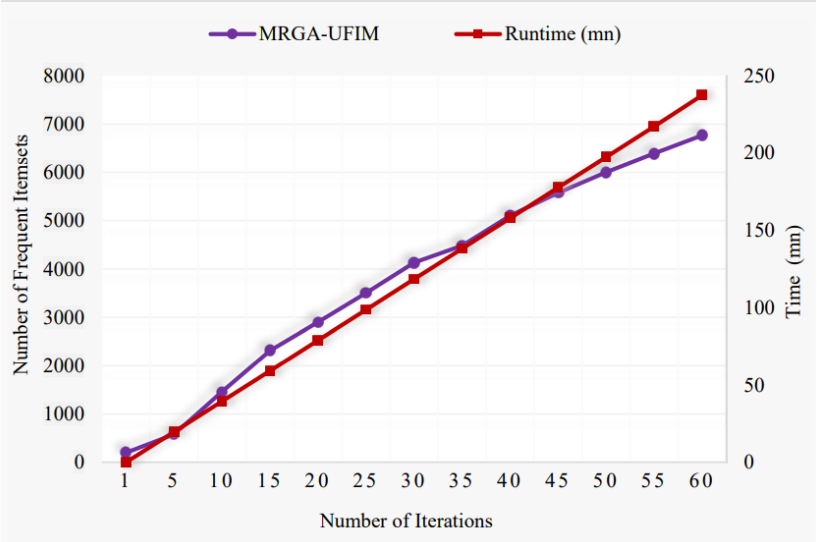


Figure.42: MRGA obtained frequent itemsets, runtime based on the #iterations(Examination#1, Test#2, Small).

Table.18: MRGA Examination#1, Test#2 coordination (Small).

Dataset	Mushroom
Algorithm	MRGA-UFIM
Threshold	0.1
Initial Population	200
Iterations	60
Frequency Average	112
Total Frequent Itemsets	6769
Average Time (mn)	3.95
Total Time (mn)	237.37

Table.19: MRBSO Examination#1, Test#2 coordination.

Dataset	Mushroom
Algorithm	MRBSO-UFIM
Threshold	0.1
Initial Population	Bee : 2 Chances : 3
Iterations	60
Frequency Average	203
Total Frequent Itemsets	12178
Average Time (mn)	1.73
Total Time (mn)	313.41

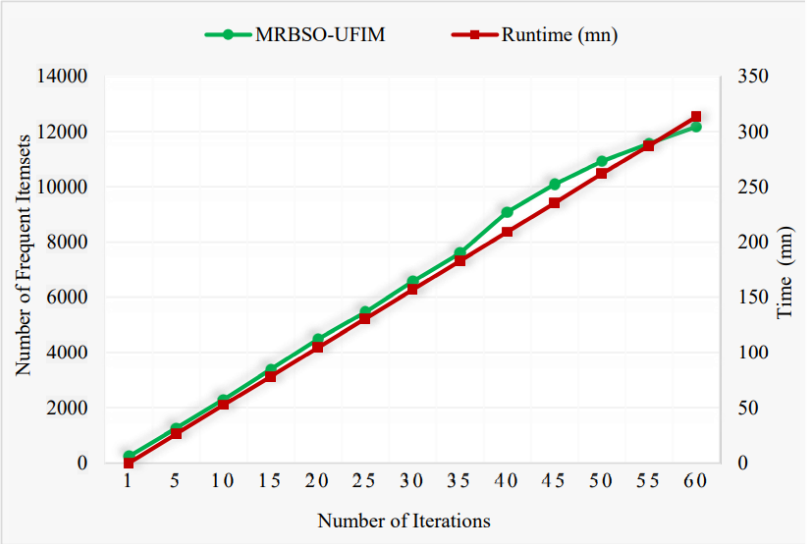


Figure.43: MRBSO obtained frequent itemsets, runtime based on the #iterations(Examination#1, Test#2 , Small).

From this Examination#1, we noticed that each time we increase the population the time of search and frequent itemsets increases in double linearly for all the algorithms. In the second Examination#2, the quality of the results is compared with the maximum obtained frequent itemsets collected from the UApriori(Chui et al., 2007) exact algorithm, which is 22.

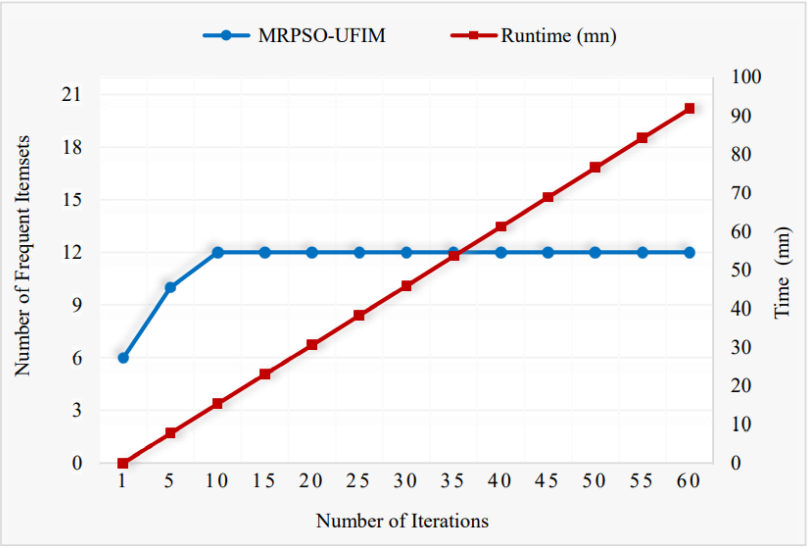


Figure.44: MRPSO obtained frequent itemsets, runtime based on the #iterations(Examination#2, Test#1, Small).

Table.20: MRPSO Examination#2, Test#1 coordination(Small).

Dataset	Mushroom
Algorithm	MRPSO-UFIM
Threshold	50%
Initial Population	100
Iterations	60
Frequency Average	-
Total Frequent Itemsets	12
Average Time (mn)	1.5
Total Time (mn)	91.83

Table.21: MRGA Examination#2, Test#1 coordination(Small).

Dataset	Mushroom
Algorithm	MRGA-UFIM
Threshold	50%
Initial Population	100
Iterations	60
Frequency Average	-
Total Frequent Itemsets	9
Average Time (mn)	1.89
Total Time (mn)	113.80

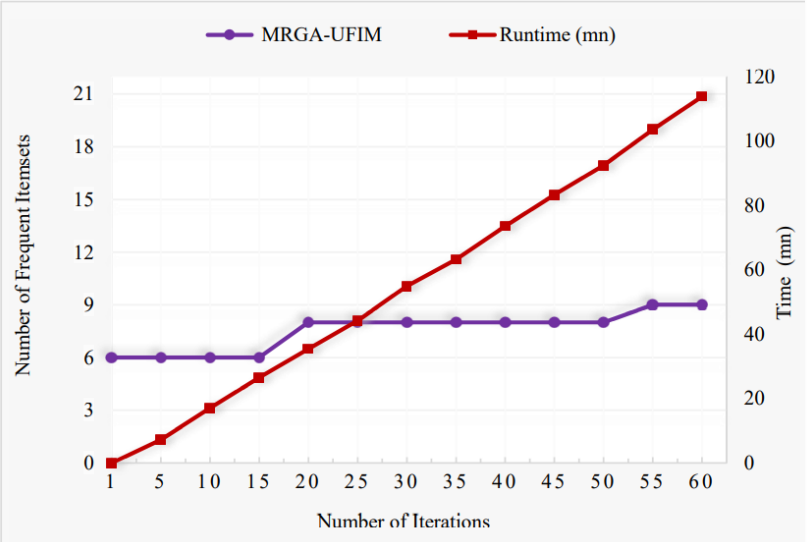


Figure.45: MRGA obtained frequent itemsets, runtime based on the #iterations(Examination#2, Test#1, Small).

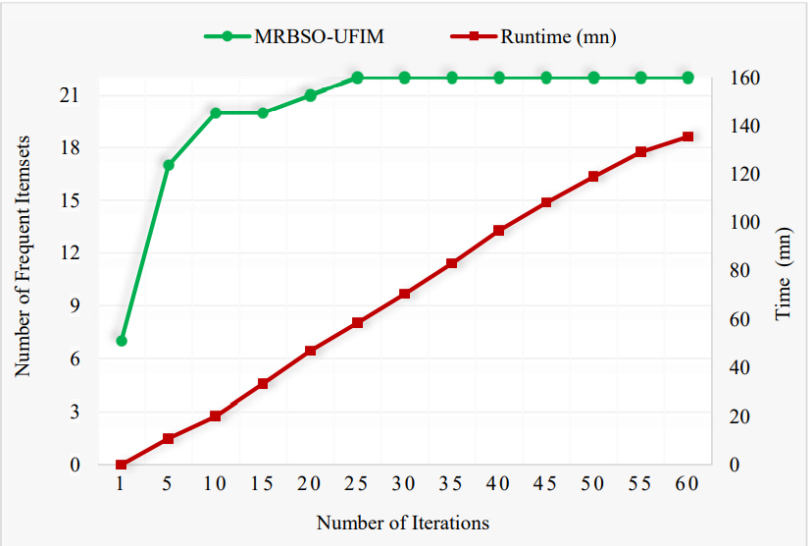


Figure.46: MRBSO obtained frequent itemsets, runtime based on the #iterations(Examination#2, Test#1, Small).

Table.22: MRBSO Examination#2, Test#1 coordination(Small).

Dataset	Mushroom
Algorithm	MRBSO-UFIM
Threshold	50%
Initial Population	Bee : 1 Chances : 2
Iterations	60
Frequency Average	-
Total Frequent Itemsets	22
Average Time (mn)	1.129
Total Time (mn)	135.48

Table.23: MRPSO Examination#2, Test#2 coordination(Small).

Dataset	Mushroom
Algorithm	MRPSO-UFIM
Threshold	50%
Initial Population	200
Iterations	60
Frequency Average	-
Total Frequent Itemsets	14
Average Time (mn)	2.72
Total Time (mn)	163.28

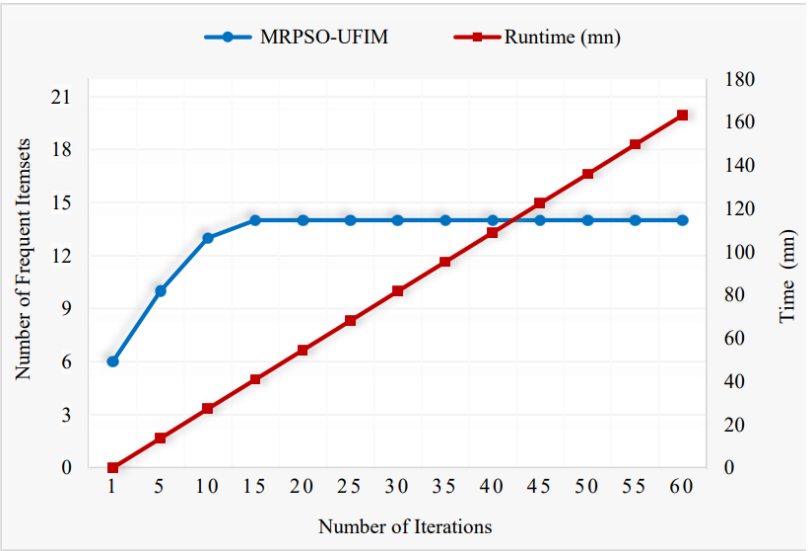


Figure.47: MRPSO obtained frequent itemsets, runtime based on the #iterations(Examination#2, Test#2, Small).

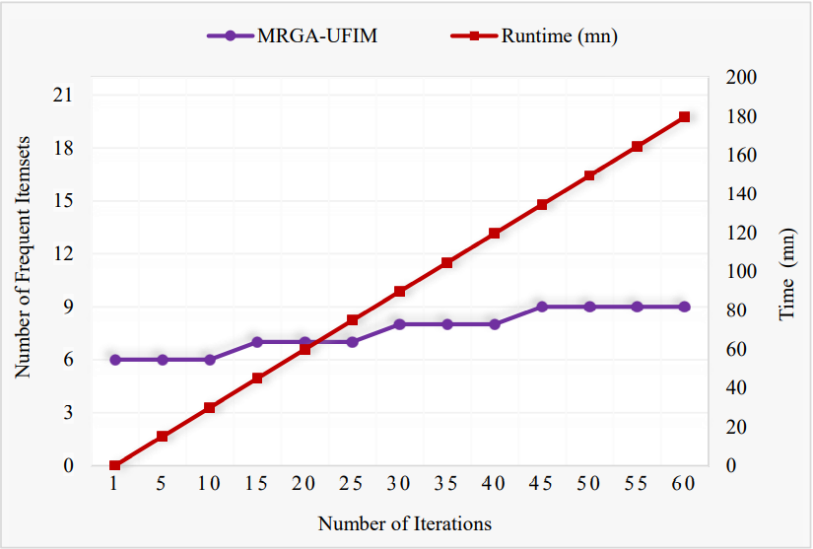


Figure.48: MRGA obtained frequent itemsets, runtime based on the #iterations(Examination#2, Test#2 , Small).

Table.24: MRGA Examination#2, Test#2 coordination(Small).

Dataset	Mushroom
Algorithm	MRGA-UFIM
Threshold	50%
Initial Population	200
Iterations	60
Frequency Average	-
Total Frequent Itemsets	9
Average Time (mn)	2.99
Total Time (mn)	179.49

Table.25: MRBSO Examination#2, Test#2 coordination(Small).

Dataset	Mushroom
Algorithm	MRBSO-UFIM
Threshold	50%
Initial Population	Bee : 2 Chances : 3
Iterations	60
Frequency Average	-
Total Frequent Itemsets	22
Average Time (mn)	1.75
Total Time (mn)	315.02

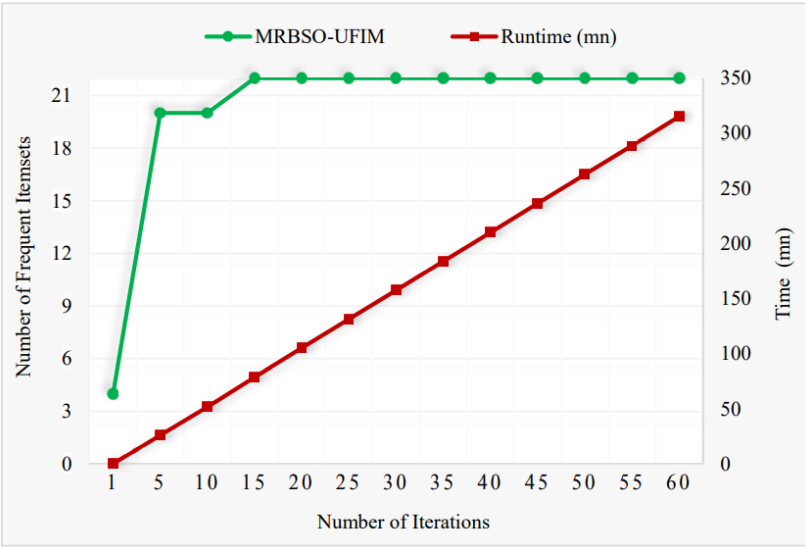


Figure.49: MRBSO obtained frequent itemsets, runtime based on the #iterations(Examination#2, Test#2, Small).

We can notice that the MRBSO in both tests, has successfully reached the total number of frequent itemsets with a threshold of 50%, compared to the others.

C. Large dataset, Few number of items

In this section, we will talk about the UBRecordLink database which is classified as large data but contains a few items, where we have done a set of tests on our three frameworks by stabilizing the initial population size and reducing the number of iterations due to the tightness of the time and performing only the examinations. As we spoke earlier, each evaluation has its information that is summarized in the tables presented beside the plot's results.

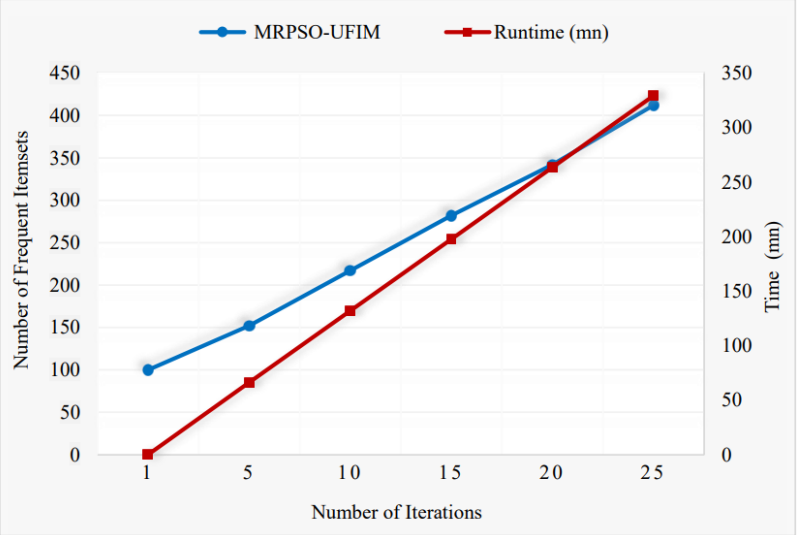


Figure.50: MRPSO obtained frequent itemsets, runtime based on the #iterations(Examination#1, Large).

Table.26: MRPSO Examination#1 coordination(Large).

Dataset	UBRecordLink
Algorithm	MRPSO-UFIM
Threshold	0.1
Initial Population	100
Iterations	25
Frequency Average	16
Total Frequent Itemsets	412
Average Time (mn)	13.17
Total Time (mn)	329.3

Table.27: MRGA Examination#1 coordination(Large).

Dataset	UBRecordLink
Algorithm	MRGA-UFIM
Threshold	0.1
Initial Population	100
Iterations	25
Frequency Average	34
Total Frequent Itemsets	868
Average Time (mn)	16.81
Total Time (mn)	420.4

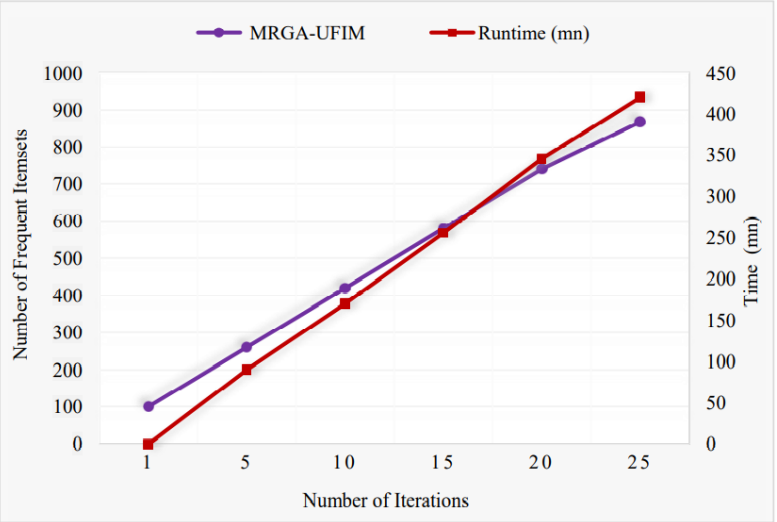


Figure.51: MRGA obtained frequent itemsets, runtime based on the #iterations(Examination#1, Large).

Table.28: MRBSO Examination#1 coordination(Large).

Dataset	UBRecordLink
Algorithm	MRBSO-UFIM
Threshold	50%
Initial Population	Bee : 2 Chances : 3
Iterations	25
Frequency Average	36
Total Frequent Itemsets	910
Average Time (mn)	40.83
Total Time (mn)	1020.76

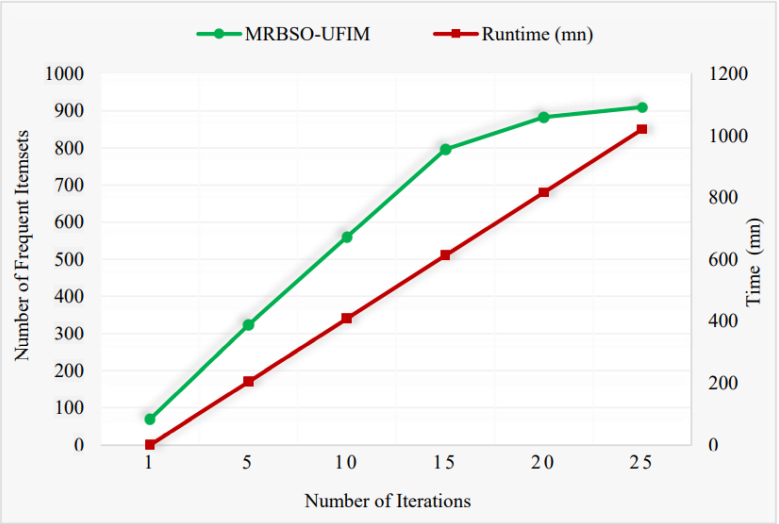


Figure.52: MRBSO obtained frequent itemsets, runtime based on the #iterations(Examination#1, Large).

After Examination#1, the threshold value is set to 50%, to raise the difficulty of the quality test.

Table.29: MRPSO Examination#2 coordination(Large).

Dataset	UBRecordLink
Algorithm	MRPSO-UFIM
Threshold	50%
Initial Population	100
Iterations	25
Frequency Average	-
Total Frequent Itemsets	17
Average Time (mn)	9.73
Total Time (mn)	243.31

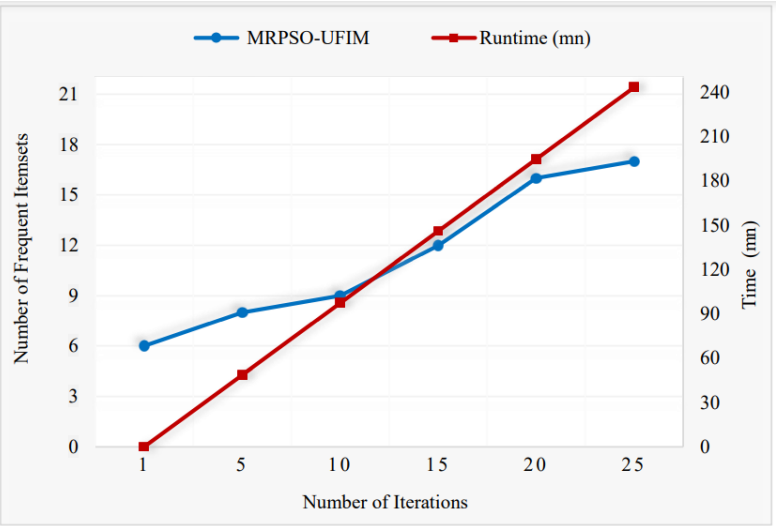


Figure.53: MRPSO obtained frequent itemsets, runtime based on the #iterations(Examination#2, Large).

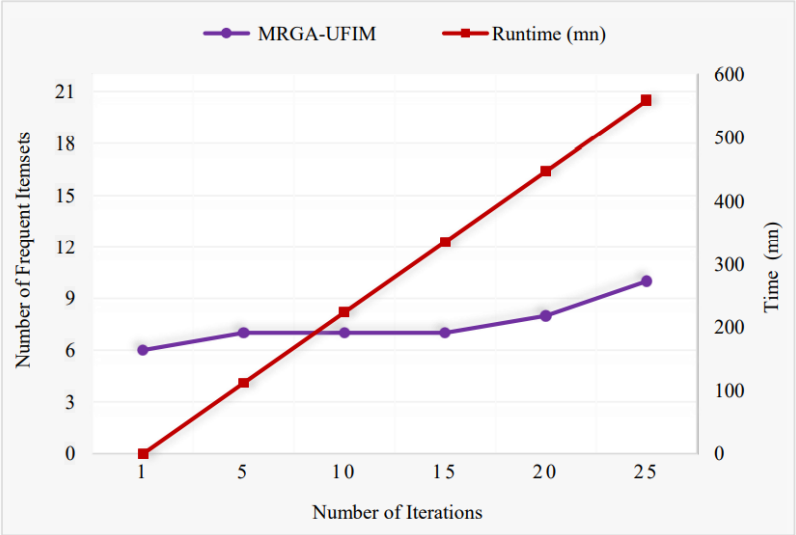


Figure.54: MRGA obtained frequent itemsets, runtime based on the #iterations(Examination#2, Large).

Table.30: MRGA Examination#2 coordination(Large).

Dataset	UBRecordLink
Algorithm	MRGA-UFIM
Threshold	50%
Initial Population	100
Iterations	25
Frequency Average	-
Total Frequent Itemsets	10
Average Time (mn)	22.36
Total Time (mn)	559.14

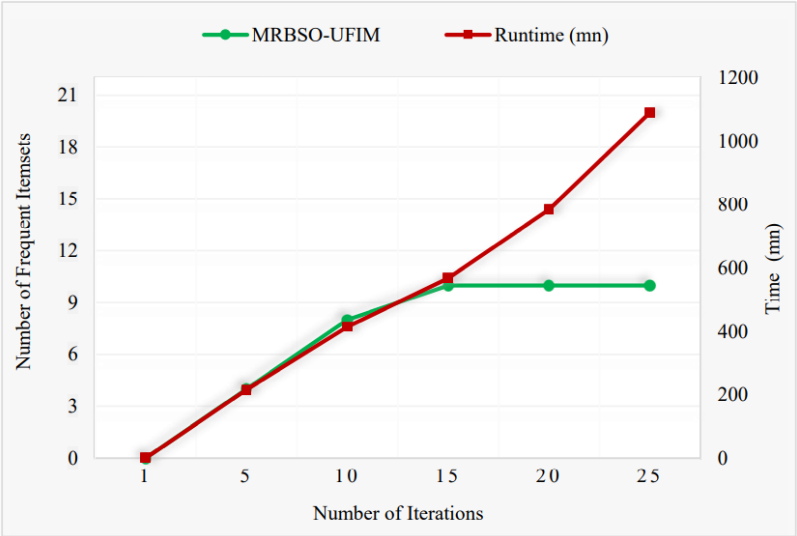


Figure.55: MRBSO obtained frequent itemsets, runtime based on the #iterations(Examination#2, Large).

Table.31: MRBSO Examination#2 coordination(Large).

Dataset	UBRecordLink
Algorithm	MRBSO-UFIM
Threshold	50%
Initial Population	Bee : 2 Chances : 3
Iterations	25
Frequency Average	-
Total Frequent Itemsets	10
Average Time (mn)	43.60
Total Time (mn)	1090.05

This Examination#2 shows that during this test the MRPSO outperforms other algorithms in both comparative terms, time and frequency.

V. Conclusion

In this examination we tested the three approaches proposed in the previous chapter, tested on three different datasets with a variation of the global parameters to better explore the designed algorithms performance, the given test

Since our proposed algorithms are metaheuristic based, we cannot give the final best achieved performances or the most efficient algorithm, we need more realistic tests that necessitate more time to compare them, we can notice also that every test takes more additional time to complete, the main reason is that we employed a single environment, due to the lack of materials. But we can, based on these meaningful tests made, perceive that the variation in the percentage of the minimum threshold didn't impact the response time, where the variation in the initial population size affected both time and obtained frequent itemsets.

General Conclusion

The goal of this thesis was to extract frequent patterns from uncertain Big Datasets in a reasonable time and with high quality results using metaheuristic based algorithms in a distributed environment. In other words, the main objective was to perform the parallel processing distributed via the Hadoop software, to conduct master and slaves network architecture, where we have proposed three approaches for the Big Data UFPM task using the MapReduce framework.

In order to address the challenge faced by our problematic optimization problem, this work has provided a literature review of the exact uncertain frequent pattern mining algorithms, next it highlights the uncertainty in Big Data including its important paradigms and comparative study of its state-of-the-art algorithms. The third chapter defines three of the well-known bio-inspired metaheuristic based algorithms and reviews their latest enhancements. Where in the last two chapters we introduced our proposed approaches and the employed tools, tests and validations of our frameworks.

I. Contributions and summary of experimental findings

- Our MRPSO-UFIM has successfully controlled its inherent problems of the early convergence occurring during the mining process, based on a lot of local tests to stabilize its parameters, the enhancements made allowed the algorithm to perform the exploration and exploitation properly according to the data coordination dynamically.

- The traditional MapReduce architecture may not help in the Big Data multiple scans, and they are specified for certain datasets. Thus, our fitness function is observed in an improved MapReduce model. Thus, all of our proposed approaches used inside the MapReduce parallel model an extra parallelization process to fastly accomplish the expected support count.

- The aim of the designed framework is to perform the maximization of the obtained frequent itemsets.

From the experiments achieved in this study work, we can acquire that in terms of efficacy, all the algorithms showed a remarkable and acceptable results, where the quality of the results has been almost reached with the MRGA and the MRBSO compared to the MRPSO, but there were some limitations that prevented us from achieving the desired execution time completely, as listed in the next part.

II. Limits and Future works

The limitations of this work are dependent to physical and material occurred problems, such as :

- One of the obstacles or problems we have faced in this work is the unavailability of the desired materials that need to be provided in the first case. This changed our main objective to apply our proposed approaches in a distribution environment of machines.
- As a consequence of the listed above failure, we had to test using our machines, where because of the massive data, one machine was disrupted and failed to restart, so we had to make all the tests on one machine with the single node clustering. That's why we didn't compare our algorithms with the rest of the existing state-of-the-art algorithms.

This research work has yet multiple interesting areas that can be further explored such as:

- As a replacement to the dynamic parameterization in our MRPSO-UFIM, we further want to use a different well-known branch, which is the reinforcement learning to better learn avoiding early convergence.
- We can notice from the first chapter that we introduced the High Utility Pattern Mining, as a perspective we want to extend from these proposed approaches a bio-inspired metaheuristic solutions for the HUPM.

References

1. Abd-Elmegid, L., E. El-Sharkawi, M., M. El-Fangary, L., & K. Helmy, Y. (2010). Vertical Mining of Frequent Patterns from Uncertain Data. *Computer And Information Science*, *3*(2). https://doi.org/10.5539/cis.v3n2p171.

2. Aggarwal, C., & Han, J. (2014). Frequent Pattern Mining. https://doi.org/10.1007/978-3-319-07821-2.

3. Agrawal, R., & Srikant, R. (1994). Fast Algorithms for Mining Association Rules in Large Databases. VLDB.

4. Balabanov, T., Ivanov, S., & Ketipov, R. (2020). Solving Combinatorial Puzzles with Parallel Evolutionary Algorithms. *Large-Scale Scientific Computing*, 493-500. https://doi.org/10.1007/978-3-030-41032-2_56.

5. Beni, G., & Wang, J. (1989). Swarm intelligence in cellular robotic systems. Proceedings of NATO advanced workshop on robots and biological systems.

6. Braun, P., Cuzzocrea, A., Leung, C., Pazdor, A., & Souza, J. (2018). Item-centric mining of frequent patterns from big uncertain data. Procedia Computer Science, 126, 1875-1884. https://doi.org/10.1016/j.procs.2018.08.075.

7. Brezočnik, L., Fister, I., & Podgorelec, V. (2018). Swarm Intelligence Algorithms for Feature Selection: A Review. Applied Sciences, 8(9), 1521. https://doi.org/10.3390/app8091521.

8. Calders, T., Garboni, C., & Goethals, B. (2010). Approximation of Frequentness Probability of Itemsets in Uncertain Data. *2010 IEEE International Conference On Data Mining*. https://doi.org/10.1109/icdm.2010.42.

9. Calders, T., Garboni, C., & Goethals, B. (2010). Efficient Pattern Mining of Uncertain Data with Sampling. *Advances In Knowledge Discovery And Data Mining*, 480-487. https://doi.org/10.1007/978-3-642-13657-3_51.

10. Cam, L.L. (1960). An approximation theorem for the Poisson binomial distribution. *Pacific Journal of Mathematics, 10*, 1181-1197.

11. Chauhan, N., (2019). *Market Basket Analysis: A Tutorial - KDnuggets*. [online] KDnuggets. Available at: <https://www.kdnuggets.com/2019/12/market-basket-analysis.html>.

12. Chebbi, I., Boulila, W., & Farah, I. (2015). Big Data: Concepts, Challenges and Applications. *Computational Collective Intelligence*, 638-647. https://doi.org/10.1007/978-3-319-24306-1_62.

13. Chui, C., Kao, B., & Hung, E. (2007). Mining Frequent Itemsets from Uncertain Data. Advances In Knowledge Discovery And Data Mining, 47-58. https://doi.org/10.1007/978-3-540-71701-0_8.

14. Darwin, C. & Kebler, L. (1859) On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life. London: J. Murray. [Pdf] Retrieved from the Library of Congress, https://www.loc.gov/item/06017473/.

15. Dean, J., & Ghemawat, S. (2004). *MapReduce: Simplified Data Processing on Large Clusters*. Google Research, from https://research.google/pubs/pub62/.

16. Dessouky, M., & Elrashidy, A. (2016). Feature Extraction of the Alzheimer's Disease Images Using Different Optimization Algorithms. Journal of Alzheimer's Disease & Parkinsonism, 06(02).

17. Ding, J., Li, H., Wang, Y., Jia, L., You, J., & Yang, Y. (2019). A Parallel Uncertain Frequent Itemset Mining Algorithm with Spark. 2019 20Th International Conference On Parallel And Distributed Computing, Applications And Technologies (PDCAT). doi: 10.1109/pdcat46702.2019.00092.

18. Djenouri, Y., Drias, H., & Habbas, Z. (2014). Bees swarm optimisation using multiple strategies for association rule mining. International Journal Of Bio-Inspired Computation, 6(4), 239. https://doi.org/10.1504/ijbic.2014.064990.

19. Drias, H., Sadeg, S., & Yahi, S. (2005). Cooperative Bees Swarm for Solving the Maximum Weighted Satisfiability Problem. Computational Intelligence And Bioinspired Systems, 318-325. https://doi.org/10.1007/11494669_39.

20. Eiben, A.E., & Smith, J.E. (2003). What is an Evolutionary Algorithm?. In: Introduction to Evolutionary Computing. Natural Computing Series. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-05094-1_2.

21. Fogel, L.J. (1962). Autonomous automata. Industrial Research, Vol. 4, pp. 14-19.

22. Foscari, C., Venezia, D., Palmerini, P., & Orlando, S. (2004). On performance of data mining: from algorithms to management systems for data exploration. 2004–2002.

23. Fournier-Viger, P., Chun-Wei Lin, J., Truong-Chi, T., & Nkambou, R. (2019). A Survey of High Utility Itemset Mining. Studies In Big Data, 1-45. https://doi.org/10.1007/978-3-030-04921-8_1.

24. Fournier-Viger, P., Lin, C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H. T. (2016). The SPMF Open-Source Data Mining Library Version 2. Proc. 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III, Springer LNCS 9853, pp. 36-40.

25. Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. Proceedings Of The 2000 ACM SIGMOD International Conference On Management Of Data - SIGMOD '00. https://doi.org/10.1145/342009.335372.

26. Hariri, R., Fredericks, E., & Bowers, K. (2019). Uncertainty in big data analytics: survey, opportunities, and challenges. *Journal Of Big Data*, *6*(1). https://doi.org/10.1186/s40537-019-0206-3.

27. Holland, J. (1984). Genetic Algorithms and Adaptation. Adaptive Control Of Ill-Defined Systems, 317-333. https://doi.org/10.1007/978-1-4684-8941-5_21.

28. Jain, R., 2017. *A beginner's tutorial on the apriori algorithm in data mining with R implementation*. [online] HackerEarth Blog. Available at: <https://www.hackerearth.com/blog/developers/beginners-tutorial-apriori-algorithm-data-mining-r-implementation/>.

29. Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. Proceedings Of ICNN'95 - International Conference On Neural Networks. https://doi.org/10.1109/icnn.1995.488968.

30. Kennedy, J., & Eberhart, R.C. (1997). A discrete binary version of the particle swarm algorithm. 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, 5, 4104-4108 vol.5.

31. Khan, S., Kamran, M., Rehman, O., Liu, L., & Yang, S. (2017). A modified PSO algorithm with dynamic parameters for solving complex engineering design problem.

32. Lee, G., Yun, U., & Ryang, H. (2015). An uncertainty-based approach: Frequent itemset mining from uncertain data with different item importance. doi: 10.1016/j.knosys.2015.08.018.

33. Leung, C.KS., & Hayduk, Y. (2013). Mining Frequent Patterns from Uncertain Data with MapReduce for Big Data Analytics. Database Systems For Advanced Applications, 440-455. https://doi.org/10.1007/978-3-642-37487-6_33.

34. Leung, C.KS., & Jiang, F. (2014). A Data Science Solution for Mining Interesting Patterns from Uncertain Big Data. 2014 IEEE Fourth International Conference On Big Data And Cloud Computing. doi: 10.1109/bdcloud.2014.136.

35. Leung, C.KS., Carmichael, C., & Hao, B. (2007). Efficient Mining of Frequent Patterns from Uncertain Data. Seventh IEEE International Conference On Data Mining Workshops (ICDMW 2007). https://doi.org/10.1109/icdmw.2007.84.

36. Leung, C.KS., Mateo, M.A.F., & Brajczuk, D.A. (2008). A Tree-Based Approach for Frequent Pattern Mining from Uncertain Data. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2008. Lecture Notes in Computer Science(), vol 5012. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-68125-0_61.

37. Leung, C.KS., & Sun, L., (2011). Equivalence class transformation based mining of frequent itemsets from uncertain data. *Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11*,.

38. Leung, C., & Tanbeer, S. (2013). PUF-Tree: A Compact Tree Structure for Frequent Pattern Mining of Uncertain Data. *Advances In Knowledge Discovery And Data Mining*, 13-25. https://doi.org/10.1007/978-3-642-37453-1_2.

39. Li, H., Huang, H., Chen, Y., Liu, Y., & Lee, S. (2008). Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams. 2008 Eighth IEEE International Conference On Data Mining. https://doi.org/10.1109/icdm.2008.107.

40. Lin, J., Gan, W., Fournier-Viger, P., Hong, T., & Tseng, V. (2015). Weighted frequent itemset mining over uncertain databases. *Applied Intelligence*, *44*(1), 232-250. https://doi.org/10.1007/s10489-015-0703-9.

41. Lin, J., Gan, W., Fournier-Viger, P., Hong, T., & Tseng, V. (2016). Efficiently mining uncertain high-utility itemsets. Soft Computing, 21(11), 2801-2820. https://doi.org/10.1007/s00500-016-2159-1.

42. Liu, M., & Qu, J. (2012). Mining high utility itemsets without candidate generation. Proceedings Of The 21St ACM International Conference On Information And Knowledge Management - CIKM '12. https://doi.org/10.1145/2396761.2396773.

43. MacKinnon, R., (2015). *Seeing the forest for the trees: tree-based uncertain frequent pattern mining*. [online] Hdl.handle.net. Available at: <http://hdl.handle.net/1993/31059>.

44. Massone, M. (2018). Cross-Sections for Transient Analyses: Development of a Genetic Algorithm for the Energy Meshing. Karlsruher Institut für Technologie (KIT).

45. Musdholofah, A., Kom, S., Kom, M., & D, P. (2020). *Evolutionary Computation and Its Application*. Docplayer.net, from https://docplayer.net/229200784-Aina-musdholofah-s-kom-m-kom-ph-d-webinar-series-lab-sc-thursday-30-july-2020.html.

46. Rathan, B., & Rani, K. (2017). A novel approach for mining patterns from large uncertain data using MapReduce model. 2017 International Conference On Computer Communication And Informatics (ICCCI). https://doi.org/10.1109/iccci.2017.8117705.

47. Rechenberg, I. (1973). Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen evolution [Evolution Strategy: Optimization of Technical Systems According to the Principles of Biological Evolution]. Frommann-Holzboog Verlag, Stuttgart.

48. Repoussis, P. *Metaheuristic Algorithms A brief introduction on the basics*. Slidetodoc.com, from https://slidetodoc.com/metaheuristic-algorithms-a-brief-introduction-on-the-basics/.

49. Reynaldo, J., & Boy Tonara, D. (2018). Data Mining Application using Association Rule Mining ECLAT Algorithm Based on SPMF. MATEC Web Of Conferences, 164, 01019. https://doi.org/10.1051/matecconf/201816401019.

50. Scrucca, L. (2013). GA: A Package for Genetic Algorithms inR. Journal Of Statistical Software, 53(4). https://doi.org/10.18637/jss.v053.i04.

51. Solanki, S.K., & Soni, N. (2015). A Survey on Frequent Pattern Mining Methods Apriori , Eclat , FP growth.

52. Sörensen, K., & Glover, F. (2013). Metaheuristics. Encyclopedia Of Operations Research And Management Science, 960-970. https://doi.org/10.1007/978-1-4419-1153-7_1167.

53. Talbi, E. (2009). Metaheuristics: From Design to Implementation. https://doi.org/10.1002/9780470496916.

54. Tezel, B., & Mert, A. (2011). A cooperative system for metaheuristic algorithms. *Expert Systems With Applications*, *165*, 113976. https://doi.org/10.1016/j.eswa.2020.113976.

55. Tong, Y., Chen, L., Cheng, Y., & Yu, P. (2012). Mining frequent itemsets over uncertain databases. *Proceedings Of The VLDB Endowment*, *5*(11), 1650-1661. https://doi.org/10.14778/2350229.2350277.

56. Tseng, V., Wu, C., Shie, B., & Yu, P. (2010). UP-Growth. Proceedings Of The 16Th ACM SIGKDD International Conference On Knowledge Discovery And Data Mining - KDD '10. https://doi.org/10.1145/1835804.1835839.

57. Wang, L., Cheng, R., Lee, S., & Cheung, D. (2010). Accelerating probabilistic frequent itemset mining. *Proceedings Of The 19Th ACM International Conference On Information And Knowledge Management - CIKM '10*. https://doi.org/10.1145/1871437.1871494.

58. Xu, J., Li, N., Mao, X., & Yang, Y. (2014). Efficient Probabilistic Frequent Itemset Mining in Big Sparse Uncertain Data. Lecture Notes In Computer Science, 235-247. https://doi.org/10.1007/978-3-319-13560-1_19.

59. Xu, J., Mao, X., Lu, W., Zhu, Q., Li, N., & Yang, Y. (2015). MapReduce-based Parallelized Approximation of Frequent Itemsets Mining in Uncertain Data. Neural Information Processing, 136-144. doi: 10.1007/978-3-319-26561-2_17.

60. Yang, X. (2011). Metaheuristic Optimization. *Scholarpedia*, *6*(8), 11472. https://doi.org/10.4249/scholarpedia.114720.

61. Zaharia, M., Chowdhury, M., Franklin, M., Shenker, S., & Stoica, I. (2010). *Apache Spark - Wikipedia*. En.wikipedia.org., from https://en.wikipedia.org/wiki/Apache_Spark.

62. Zaki, M. (2000). Scalable algorithms for association mining. IEEE Transactions On Knowledge And Data Engineering, 12(3), 372-390. https://doi.org/10.1109/69.846291.

63. Zhang, B. (2021). Optimization of FP-Growth algorithm based on cloud computing and computer big data. International Journal Of System Assurance Engineering And Management, 12(4), 853-863. https://doi.org/10.1007/s13198-021-01139-2.

64. Zida, S., Fournier-Viger, P., Lin, J.CW., Wu, CW., & Tseng, V.S. (2015). EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining. In: Sidorov, G., Galicia-Haro, S. (eds) Advances in Artificial Intelligence and Soft Computing. MICAI 2015. Lecture Notes in Computer Science(), vol 9413. Springer, Cham. https://doi.org/10.1007/978-3-319-27060-9_44..