**VIETNAM GENERAL CONFEDERATION OF LABOR**

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**REPORT**

**DESIGN AND ANALYSIS OF ALGORITHMS**

*Instructor*: **NGUYEN CHI THIEN**

*Student*: **BUI ANH PHU - 521H0508**

**BUI HAI DUONG - 521H0220**

**NGUYEN HOANG PHUC - 521H0511**

*Class*: **21H50302**

**HO CHI MINH CITY, 2023**

**VIETNAM GENERAL CONFEDERATION OF LABOR**

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**REPORT**

**DESIGN AND ANALYSIS OF ALGORITHMS**

*Instructor*: **NGUYEN CHI THIEN**

*Student*: **BUI ANH PHU - 521H0508**

**BUI HAI DUONG - 521H0220**

**NGUYEN HOANG PHUC - 521H0511**

*Class*: **21H50302**

**HO CHI MINH CITY, 2023**

ACKNOWLEDGEMENT

To complete this essay, besides our own efforts, we have received a lot of help in terms of knowledge, experience, and skills from the school and teachers. First and foremost, we would like to express my special gratitude to Nguyen Chi Thien - the lecturer of Design and Analysis of Algorithms course who has taught us valuable knowledge of the subject. That knowledge is the foundation for us to continue learning and effectively apply it to this essay. Additionally, We would like to thank the teacher for allowing us to complete this essay, which has helped us to further develop my understanding of the subject. Thank you for your guidance and support in helping us to complete this essay to the best of my ability. Moreover, we would also like to express our gratitude to the school and the teachers who have compiled the Design and Analysis of Algorithms materials, providing me with useful resources for research and essay writing. Thank you sincerely!

**COMPLETION OF THESIS**

**AT TON DUC THANG UNIVERSITY**

We here by certify that this thesis is my/our own work and was conducted under the guidance of Nguyen Chi Thien. The research and results presented in this thesis are truthful and have not been published previously in any form. The data presented in tables and figures used for analysis, comments, and evaluations were collected by the author from various sources and are clearly cited in the reference section.

Moreover, this thesis includes some comments, evaluations, and data from other authors and organizations, which are properly cited and referenced.

If any misconduct is detected, I fully take responsibility for the content of my thesis. Ton Duc Thang University is not liable for any copyright infringement that may occur during the thesis completion process.

*Ho Chi Minh City, October 22, 2023*

*Author*

*(signature and full name)*

ACKNOWLEDGEMENT AND EVALUATION SECTION BY INSTRUCTOR

**Instructor's Acknowledgement Section**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ho Chi Minh City, 2023

(signature and full name)

**Instructor's Evaluation Section**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ho Chi Minh City, 2023

(signature and full name)

SUMMARY

This document showing the theory based on the research document of Efficient weighted probabilistic frequent itemset mining in uncertain databases, explain the java code following the research then represent the result benchmark.

Table of Contents

[ACKNOWLEDGEMENT i](#_Toc154324995)

[ACKNOWLEDGEMENT AND EVALUATION SECTION BY INSTRUCTOR iii](#_Toc154324996)

[SUMMARY iv](#_Toc154324997)

[CHAPTER 1 – EFFICIENT WEIGHTED PROBABILISTIC FREQUENT ITEMSET MINING IN UNCERTAIN DATABASES THEORY BASED ON THE STUDY DOCUMENT 1](#_Toc154324998)

[1.1 Introduction 1](#_Toc154324999)

[1.2 Definitions 1](#_Toc154325000)

[1.2.1 Definition 1 (Uncertain dataset): 1](#_Toc154325001)

[1.2.2 Definition 2 (Probabilistic frequent itemset): 2](#_Toc154325002)

[1.2.3 Definition 3 (Weighted probabilistic frequent itemset): 3](#_Toc154325003)

[1.2.4 Definition 4 (Itemset weight): 4](#_Toc154325004)

[1.3 Theory 5](#_Toc154325005)

[1.3.1 Theorem 1 (Anti-monotonicity property for weighted PFI): 5](#_Toc154325006)

[1.3.2 Theorem 2: 6](#_Toc154325007)

[1.3.3 Theorem 3: 6](#_Toc154325008)

[1.3.4 Theorem 4: 6](#_Toc154325009)

[1.4 Algorithm 7](#_Toc154325010)

[1.4.1 Algorithm 1 7](#_Toc154325011)

[1.4.2 Algorithm 2 9](#_Toc154325012)

[1.4.3 Algorithm 3 10](#_Toc154325013)

[2.1 Java classes 12](#_Toc154325014)

[2.1.1 UncertainDatabase class 12](#_Toc154325015)

[2.1.2 wPFIApriori class 16](#_Toc154325016)

[2.1.3 wPFIItem class 28](#_Toc154325017)

[2.2 Asymptotic Complexity Analysis (The worst-case complexity) 30](#_Toc154325018)

[2.3 Benchmark 31](#_Toc154325019)

[2.3.1 Runtime based on msup changing 31](#_Toc154325020)

[2.1.1 Runtime based on threshold t changing 33](#_Toc154325021)

[Conclusions 35](#_Toc154325022)

[**REFERENCES** 36](#_Toc154325023)

**LIST OF SYMBOLS AND ABBREVIATIONS**

**Symbols**

Uncertain dataset of size n

A set of transaction identifiers

An itemset of size m

An existential probability for item appearing in the transaction

A weight table for the itemset I. Each item has a real-valued weight

An integer between (0, n], the minimum support for expected support- based FI mining

**msup** An integer between (0, n], the minimum support for probabilistic FI mining

A real value between (0, 1], the probabilistic frequent threshold for probabilistic FI mining

A real value between [0, 1], the scale factor

**Abbreviations**

**PFI** probability frequent itemset

**w-PFI** weighted probability frequent itemset

CHAPTER 1 – EFFICIENT WEIGHTED PROBABILISTIC FREQUENT ITEMSET MINING IN UNCERTAIN DATABASES THEORY BASED ON THE STUDY DOCUMENT

* 1. Introduction

Uncertain data mining has attracted so much interest in many emerging applications over the past decade. Several probability models are presented to measure the frequency of an itemset in the dataset, and it is noted that the frequency itself cannot identify useful or meaningful patterns in some scenarios. Therefore, weighted (importance) frequent itemset mining in uncertain databases has been done in some studies but the result is still inefficient. To overcome this issue, we introduce the new algorithms and some pruning methods for narrowing space and improve the candidate generation for the result.

* 1. Definitions

In this section, there are all the fundamental definitions mentioned in the study document, which are needed before dealing with the problem of finding efficient weighted probabilistic frequent itemset mining in uncertain databases.

* + 1. Definition 1 (Uncertain dataset):

An uncertain dataset DB consists of a set of transactions DB = , Each transaction contains a subset of an itemset I = {}. There is an existential probability for each item in the transaction of DB.

For example:

DB = {}

= {(Milk, 0.4), (Fruit, 0.9), (Video, 0.6)}

= {(Milk, 0.3), (Fruit, 0.7)}

With:

DB is the uncertain database.

is the transaction identifier of DB.

In Itemset contains items Milk, Fruit, Video with the corresponding probability 0.4, 0.9, 0.6.

* + 1. Definition 2 (Probabilistic frequent itemset):

An itemset is a probabilistic frequent itemset if and only if:

The probability can be calculated by:

With:

is the probabilistic frequent threshold.

is the minimum support.

is the support (occurrence) of the itemset

is the possible world.

For example:

|  |  |  |
| --- | --- | --- |
| W | Tuples in W | Probability |
| 1 |  | 0.35 |
| 2 |  | 0.15 |
| 3 |  | 0.35 |
| 4 |  | 0.15 |

We set:

Therefore is the Probabilistic frequent itemset ()

* + 1. Definition 3 (Weighted probabilistic frequent itemset):

Weighted probabilistic frequent itemset is the probabilistic frequent itemset product with the weight of the itemset :

With:

is the weight of the itemset

is the minimum support.

is the support (occurrence) of the itemset

For example:

Weight table

|  |  |
| --- | --- |
| Item | Weight |
|  |  |
|  |  |

Possible world:

|  |  |  |
| --- | --- | --- |
| W | Tuples in W | Probability |
| 1 |  | 0.35 |
| 2 |  | 0.15 |
| 3 |  | 0.35 |
| 4 |  | 0.15 |

The itemset weight will be mentioned in definition 6 below.

Since 0.39 > , then is the Weighted probabilistic frequent itemset ()

* + 1. Definition 4 (Itemset weight):

The weight of X is the average weight of the items in the itemset X:

Itemset weight calculated by the formula:

For example:

Weight table

|  |  |
| --- | --- |
| Item | Weight |
|  |  |
|  |  |
|  |  |

We have the itemset = {Milk, Fruit, Video}

The weight of X:

* 1. Theory

In this section, here are the theorems mentioned in the document which have been used in the research. Some of these theorems are mainly used for the candidate generating and pruning algorithm, which will be present below soon.

* + 1. Theorem 1 (Anti-monotonicity property for weighted PFI):

If an itemset is a , with the length of the set is k, there is at least one itemset with the length of the set is is a .

From which has been said in the paper, the anti-monotonicity property for the weighted PFI is different from the PFI. Therefore, they decided to design the new novel candidate generation items in the set without having the combination of the duplicated items.

Corollary 1:

The Corollary 1 based on the theorem 1:

We have the and where is the set of the wPFI with the size , is the item have the smallest weight in the itemset .

With , is not a if and ,

where and consist of all size 1 item of .

* + 1. Theorem 2:

We have is the random variable, following the Poisson Binomial Distribution.

With:

is the probability of the itemset , where is the subset of transaction

is the mean in the Poisson Binomial Distribution.

is the variance in the Poisson Binomial Distribution.

Frequentness probability of an itemset X calculated by this formular:

With:

F(.) is the cumulative distribution function of Poisson distribution.

* + 1. Theorem 3:

The frequentness probability increases monotonically with

Following the paper (Wang et al., 2010), an itemset X is more likely to be PFI if it has higher .

* + 1. Theorem 4:

We have two itemset and .

~PBD (), where the mean and variance

~PBD (), where the mean and variance

, where the mean and the variance

Based on the theorem:

Corollary 2:

The Corollary 2 based on the Theorem 6:

We have the itemset is not a if by applying from the Theorem 6

With:

The is thebased on the condition that if

Corollary 3:

We also have the Corollary 3

We have the itemset is not a if

With:

The is thebased on the condition that if.

is the number of transactions.

* 1. Algorithm
     1. Algorithm 1

Pseudocode given by the paper:

The algorithm 1 is implemented by applying the corollary 1.

A computer screen shot of a computer code

Description automatically generated

Understanding the algorithm:

The algorithm starts by finding all candidates. This is done by simply scanning the database and keeping the items that have a support of at least the minimum support threshold.

Once the set of candidates has been found, the algorithm proceeds to generate all possible candidates by joining the size-k candidates with each other. The algorithm then prunes the candidate set by removing any candidates that are not downwardly closed. A candidate is downwardly closed if it contains all its subsets.

The remaining candidates are then scanned in the database to count their support. Any candidates with a support of at least the minimum support threshold are added to the set of . The algorithm then repeats until no more WPFI candidates can be generated.

* + 1. Algorithm 2

Pseudocode given by the paper:

The algorithm 2 is implemented by applying the corollary 2.

A screenshot of a computer program

Description automatically generated

Understanding the algorithm:

Using the iteration to take each itemset obtained by the , then combine these itemset with each item which is not contained by the itemset . If the weight of the combination reaches the condition that is smaller or equal to the threshold t, then that combination would be added into a set of the result.

In the iteration, continuing take each item in the list contains the itemset except the itemset of the combination mentioned above. Then combine that item with the itemset . If the new combination reaches the condition that is smaller or equal to threshold t and the weight of combined item greater than the item has the minimum the weight of the itemset then adding the combination to the list.

The algorithm then repeats until no more itemset remaining.

* + 1. Algorithm 3

Pseudocode:

The algorithm 3 is based on the corollary 3.

A screenshot of a computer program

Description automatically generated

Understanding the algorithm:

Using the iteration to take each itemset obtained by the , then combine these itemset with each item which is not contained by the itemset . If the weight of the combination reaches the condition that is smaller or equal to the threshold t, then it go to the next condition. If the min between the mu of itemset with the greater than or equal to the and the product of mu and is greater than or equal to the product of scale factor, number of transaction and then that combination would be added into a set of the result.

In the iteration, continuing take each item in the list contains the itemset except the itemset of the combination mentioned above. Then combine that item with the itemset . If the new combination reaches the condition that is smaller or equal to threshold t and the weight of combined item greater than the item has the minimum the weight of the itemset , then we go to the next condition. If the min between the mu of itemset with the greater than or equal to the and the product of mu and is greater than or equal to the product of scale factor, number of transaction and then adding the combination to the list.

The algorithm then repeats until no more itemset remaining.

**CHAPTER 2 – JAVA CODE IMPLIMENTATION FOR EFFICIENT WEIGHTED PROBABILISTIC FREQUENT ITEMSET MINING IN UNCERTAIN DATABASES**

* 1. **Java classes** 
     1. **UncertainDatabase class**

**Introduction:** The UncertainDatabase class represent for the uncertain database with existential probabilities. This class is implemented based on the **Definition 1**.

**Detail:**

Attribute:

private final HashSet<wPFIItem> allItems = new HashSet<wPFIItem>();

private final ArrayList<HashSet<wPFIItem>> transactions = new ArrayList<>();

Method:

These function below based on the Definition 1:

public int size()

**Parameter:** None

**Output:** the integer value is the size of the database.

* Get the database size.
* public int size() {
* return transactions.size();
* }

public ArrayList<HashSet<wPFIItem>> getTransactions()

**Parameter:** None

**Output:** the list of Transactions.

* Get the list of transactions.
* public ArrayList<HashSet<wPFIItem>> getTransactions() {
* return transactions;
* }

public HashSet<wPFIItem> getAllItems()

**Parameter:** None

**Output:** a Set of Items

* Get the set of items in this database.
* public HashSet<wPFIItem> getAllItems() {
* return allItems;
* }

private void processTransactions(String itemsString[])

**Parameter:** None

**Output:** itemsString the list of items

* Process a transaction from a list of items, then add it to the transaction list.
* private void processTransactions(String itemsString[]) {
* HashSet<wPFIItem> transaction = new HashSet<>();
* for (String itemString : itemsString) {
* int itemID = Integer.parseInt(itemString);
* double value = gaussianDistribution();
* wPFIItem item = new wPFIItem(itemID, value);
* transaction.add(item);
* allItems.add(item);
* }
* transactionSize += transaction.size();
* transactions.add(transaction);
* }

Here are some support function to print out the information of the database.

private void processTransactionsWithProbability(String itemsString[])

**Parameter:** None

**Output:** None

* private void processTransactionsWithProbability(String itemsString[]) {
* HashSet<wPFIItem> transaction = new HashSet<>();
* String pattern = "\\((\\d+),(\\d+\\.\\d+)\\)";
* Pattern p = Pattern.compile(pattern);
* for (String itemString : itemsString) {
* try {
* Matcher matcher = p.matcher(itemString);
* if (!matcher.find())
* continue;
* int itemID = Integer.parseInt(matcher.group(1));
* double value = Double.parseDouble(matcher.group(2));
* if (value == 0)
* continue;
* wPFIItem item = new wPFIItem(itemID, value);
* transaction.add(item);
* allItems.add(item);
* } catch (Exception e) {
* e.printStackTrace();
* return;
* }
* }
* transactionSize += transaction.size();
* transactions.add(transaction);
* }

public void printDatabase(boolean hasProbability)

**Parameter:** None

**Output:** None

* Printing out the database.
* public void printDatabase(boolean hasProbability) {
* System.out.println("===================  UNCERTAIN DATABASE ===================");
* int count = 0;
* for (HashSet<wPFIItem> itemset : transactions) {
* System.out.print("0" + count + ":\t");
* if (hasProbability) {
* String str = "";
* for (wPFIItem item : itemset)
* str += item.toStringWithProbability();
* System.out.print(str);
* } else
* System.out.print(itemset.toString());
* System.out.println();
* count++;
* }
* }

public void printDatabaseProperties(String path)

**Parameter:** None

**Output:** None

* Printing out database properties.
* public void printDatabaseProperties(String path) {
* System.out.println("=================== DATABASE PROPERTIES ===================");
* System.out.println("File path: " + path);
* System.out.println("Database size: " + transactions.size());
* System.out.println("Distinct items: " + allItems.size());
* }

private static double gaussianDistribution()

**Parameter:** None

**Output:** a double probability based on gaussianDistribution

* A random probability based on Gaussian Distribution for an item.
* private static double gaussianDistribution() {
* Random random = new Random();
* double prob = random.nextGaussian() \* Math.sqrt(0.125) + 0.5;
* prob = Math.round(prob \* 10) / (Double) 10.0;
* if (prob > 1)
* return 1;
* if (prob <= 0)
* return 0.1;
* return prob;
* }
* }
  + 1. **wPFIApriori class**

**Introduction:**

The wPFIApriori class is implemented for the representation of 3 algorithms mentioned in the theory in chapter 1.

**Detail:**

Attributes:

protected UncertainDatabase db;

protected HashSet<wPFIItem> allItems;

protected HashMap<Integer, Double> weightTable;

protected int k;

protected int minsup;

protected int databaseSize;

protected float t;

Method:

public void runAlgorithm(float msup\_ratio, float threshold, float scale\_factor)

**Parameter:**

msup\_ratio: a float representing the minimum support ratio.

Threshold: a float representing the minimum confidence threshold.

scale\_factor: a float representing the scaling factor for the probability model.

**Output:** None

* The implementation of Algorithm 1 from the research paper
* public void runAlgorithm(float msup\_ratio, float threshold, float scale\_factor, boolean useProbabilityModel) {
* this.startTime = System.currentTimeMillis();
* this.k = 1;
* this.t = threshold;
* this.minsup = (int) Math.round(msup\_ratio \* databaseSize);
* this.alpha = scale\_factor;
* System.out.println("===========================================================");
* System.out.println("Minimum support ratio: " + msup\_ratio);
* System.out.println("Confidence threshold: " + threshold);
* ArrayList<HashSet<HashSet<wPFIItem>>> wPFI = new ArrayList<>();
* System.out.println("===========================================================");
* HashSet<HashSet<wPFIItem>> wPFI\_k = scanFindSize1();
* wPFI.add(wPFI\_k);
* while (wPFI\_k.size() != 0) {
* HashSet<HashSet<wPFIItem>> candidateK = wPFIAprioriGenerate(wPFI\_k, useProbabilityModel);
* System.out.printf("There are %d\t size-%d candidates.\n", candidateK.size(), k);
* wPFI\_k = scanFindSizeK(candidateK);
* wPFI.add(wPFI\_k);
* k++;
* }
* endTime = System.currentTimeMillis();
* System.out.println("===========================================================");
* System.out.printf("Total runtime: %ds", (int) (endTime - startTime) / 1000);
* }

protected HashMap<Integer, Double> generateWeightTable()

**Parameter:** None

**Output:** a HashMap of integer keys and double values representing the weight of each item.

* Generate a weight table that assigns a random weight between 0 and 1 to each item based on the Definition 1.
* public HashMap<Integer, Double> generateWeightTable() {
* HashMap<Integer, Double> weightTable = new HashMap<Integer, Double>();
* Random random = new Random();
* for (wPFIItem item : allItems) {
* weightTable.put(item.getId(), random.nextDouble());
* }
* return weightTable;
* }

protected double itemsetWeight(HashSet<wPFIItem> itemset)

**Parameter:**

Itemset: a HashSet of wPFIItem objects representing an itemset.

**Output:** a double value representing the average weight of the items in the itemset.

* Calculate the average weight of items within a given itemset based on the Definition 4.
* public double itemsetWeight(HashSet<wPFIItem> itemset) {
* double sumWeight = 0;
* for (wPFIItem item : itemset) {
* sumWeight += weightTable.get(item.getId());
* }
* return sumWeight / itemset.size();
* }

protected ArrayList<HashSet<wPFIItem>> scanFindSize1()

**Parameter:** None

**Output:** a HashSet of HashSet of wPFIItem objects representing FPIs of size 1.

* Finds wPFIs of size 1 based on the Algorithm 1.
* public HashSet<HashSet<wPFIItem>> scanFindSize1() {
* HashSet<HashSet<wPFIItem>> new\_candidates = new HashSet<HashSet<wPFIItem>>();
* HashSet<wPFIItem> candidate = new HashSet<wPFIItem>();
* for (wPFIItem item : allItems) {
* candidate.add(item);
* double candidate\_weight = weightTable.get(item.getId());
* double candidate\_confidence = Pr(candidate);
* if (candidate\_confidence \* candidate\_weight >= t)
* new\_candidates.add(candidate);
* candidate.clear();
* }
* return new\_candidates;
* }

protected ArrayList<HashSet<wPFIItem>> scanFindSizeK(ArrayList<HashSet<wPFIItem>> wPFI\_k)

**Parameter:**

wPFI\_k: a HashSet of HashSet of wPFIItem objects representing candidate PFIs of size k.

**Output:** a HashSet of HashSet of wPFIItem objects representing FPIs of size k.

* Identify PFIs of size k from a set of candidates wPFI based on the Algorithm 1.
* public HashSet<HashSet<wPFIItem>> scanFindSizeK(HashSet<HashSet<wPFIItem>> wPFI\_k) {
* HashSet<HashSet<wPFIItem>> new\_candidates = new HashSet<HashSet<wPFIItem>>();
* for (HashSet<wPFIItem> candidate : wPFI\_k) {
* double candidate\_weight = itemsetWeight(candidate);
* double candidate\_confidence = Pr(candidate);
* if (candidate\_confidence \* candidate\_weight >= t)
* new\_candidates.add(candidate);
* }
* return new\_candidates;
* }

protected double itemsetSupportInTransaction(int j, HashSet<wPFIItem> itemset)

**Parameter:**

j: an integer value representing the index of the transaction to be analyzed.

Itemset: a HashSet of wPFIItem objects representing the itemset for which support is calculated.

**Output:** a double value representing the probability of the given itemset occurring in the specified transaction.

* Calculate the support of a given itemset within a specific transaction. This is the support function for Pr()
* public double itemsetSupportInTransaction(int j, HashSet<wPFIItem> itemset) {
* HashSet<wPFIItem> transaction = db.getTransactions().get(j);
* if (itemset.size() > transaction.size())
* return 0;
* double probability = 1;
* for (wPFIItem item : itemset) {
* boolean found = false;
* for (wPFIItem itemTransaction : transaction) {
* if (itemTransaction.equals(item)) {
* found = true;
* probability \*= itemTransaction.getProbability();
* break;
* }
* }
* if (!found)
* return 0;
* }
* return probability;
* }

protected double Pr(HashSet<wPFIItem> itemset)

**Parameter:**

Itemset: a HashSet of wPFIItem objects representing an itemset.

**Output:** a double value representing the probability of the given itemset occurring in a transaction.

* Calculate the probability of a given itemset occurring in a transaction based on Theorem 2.
* public double Pr(HashSet<wPFIItem> itemset) {
* double[][] P = new double[minsup + 1][databaseSize + 1];
* double mu\_itemset = 0;
* double[] probabilities = new double[databaseSize];
* for (int i = 0; i < databaseSize; i++) {
* probabilities[i] = itemsetSupportInTransaction(i, itemset);
* mu\_itemset += probabilities[i];
* }
* supportDict.put(itemset, mu\_itemset);
* for (int j = 0; j <= databaseSize; j++) {
* P[0][j] = 1.0;
* }
* for (int i = 1; i <= minsup; i++) {
* if (P[i - 1][databaseSize - minsup + i] < t) {
* return 0.0;
* }
* for (int j = i; j <= databaseSize; j++) {
* P[i][j] = P[i - 1][j - 1] \* probabilities[j - 1] + P[i][j - 1] \* (1 - probabilities[j - 1]);
* }
* }
* return P[minsup][databaseSize];
* }

protected double minWeightItemset(HashSet<wPFIItem> itemset)

**Parameter:**

Itemset: a HashSet of wPFIItem objects representing an itemset.

**Output:** a double value representing the minimum weight of any item in the given itemset.

* Find the minimum weight of the items within the given itemset based on the Algorithm 2 and Algorithm 3.
* public double minWeightItemset(HashSet<wPFIItem> itemset) {
* double minWeight = 1.1;
* double itemWeight;
* for (wPFIItem item : itemset) {
* itemWeight = weightTable.get(item.getId());
* if (itemWeight < minWeight) {
* minWeight = itemWeight;
* }
* }
* return minWeight;
* }

protected ArrayList<HashSet<wPFIItem>> wPFIAprioriGenerate(ArrayList<HashSet<wPFIItem>> wPFI\_K\_1)

**Parameter:**

wPFI\_k: a HashSet of HashSet of wPFIItem objects representing candidate PFIs of size k.

**Output:** a HashSet of HashSet of wPFIItem objects representing FPIs of size k.

* The implementation of Algorithm 2 in the research paper for generating candidates wPFI of size k from wPFI of size k-1.
* public HashSet<HashSet<wPFIItem>> wPFIAprioriGenerate(HashSet<HashSet<wPFIItem>> wPFI\_K\_1,
* boolean useProbabilityModel) {
* HashSet<HashSet<wPFIItem>> candidateK = new HashSet<HashSet<wPFIItem>>();
* HashSet<wPFIItem> I\_ = new HashSet<wPFIItem>();
* HashSet<wPFIItem> differentSet = new HashSet<>();
* HashSet<wPFIItem> tempCandidate = new HashSet<>();
* for (HashSet<wPFIItem> candidate : wPFI\_K\_1) {
* I\_.addAll(candidate);
* }
* double maxWeight = Collections.max(weightTable.values());
* double mu\_ = calculateMu\_(maxWeight, 0, databaseSize);
* for (HashSet<wPFIItem> candidate : wPFI\_K\_1) {
* differentSet.addAll(I\_);
* differentSet.removeAll(candidate);
* for (wPFIItem item : differentSet) {
* tempCandidate.addAll(candidate);
* tempCandidate.add(item);
* if (itemsetWeight(tempCandidate) < t) {
* tempCandidate.clear();
* continue;
* }
* if (useProbabilityModel) {
* if (!conditionAlgorithm3(candidate, item, mu\_)) {
* tempCandidate.clear();
* continue;
* }
* }
* candidateK.add(new HashSet<>(tempCandidate));
* tempCandidate.clear();
* }
* double argmin = minWeightItemset(candidate);
* tempCandidate.clear();
* differentSet.addAll(allItems);
* differentSet.removeAll(I\_);
* // differentSet.removeAll(candidate);
* for (wPFIItem item : differentSet) {
* tempCandidate.addAll(candidate);
* tempCandidate.add(item);
* if (itemsetWeight(tempCandidate) < t) {
* tempCandidate.clear();
* continue;
* }
* if (weightTable.get(item.getId()) >= argmin) {
* tempCandidate.clear();
* continue;
* }
* if (useProbabilityModel) {
* if (!conditionAlgorithm3(candidate, item, mu\_)) {
* tempCandidate.clear();
* continue;
* }
* }
* candidateK.add(new HashSet<>(tempCandidate));
* tempCandidate.clear();
* }
* differentSet.clear();
* }
* return candidateK;
* }

protected double factorial(int n)

**Parameter:**

n: an integer representing the non-negative number for which the factorial is to be calculated.

**Output:** a double value representing the factorial of n.

* Calculate the factorial of a given non-negative integer n. This is the support function for calculating the CDF of Poisson Distribution.
* public double factorial(int n) {
* if (n == 0 || n == 1)
* return 1.0;
* double result = 1;
* for (int i = 2; i <= n; i++)
* result \*= i;
* return result;
* }

protected double CDF(int k, double lambda)

**Parameter:**

k: an integer representing the number of occurences.

Lambda: a double value representing the average rate of occurences.

**Output:** a double value representing the CDF at step k.

* Calculate the CDF of Poisson Distribution at a given k value based on the Theorem 2.
* public double CDF(int k, double lambda) {
* double result = 0;
* for (int i = 0; i <= k; i++) {
* result += Math.pow(lambda, i) / factorial(i);
* }
* result \*= Math.pow(Math.E, -lambda);
* return result;
* }

protected double calculateMu\_(double maxWeight, int lower, int upper)

**Parameter:**

maxWeight: a double value representing the maximum weight in the weight table.

Lower: an integer representing the lower bound for the binary search.

Upper: an integer representing the upper bound for the binary search.

**Output:** a double value representing the mu\_ threshold.

* This method approximates the mu\_ threshold using a binary search algorithm based on the Theorem 4.
* public double calculateMu\_(double maxWeight, int lower, int upper) {
* double epsilon = 0.000001;
* double lowerDouble = (double) lower;
* double upperDouble = (double) upper;
* while (upperDouble - lowerDouble > epsilon) {
* double value = 1 - CDF(minsup - 1, (upperDouble + lowerDouble) / 2.0) - t / maxWeight;
* if (value > 0)
* upperDouble -= (upperDouble + lowerDouble) / 2.0;
* else if (value < 0)
* lowerDouble += (upperDouble + lowerDouble) / 2.0;
* else
* break;
* }
* return (upperDouble + lowerDouble) / 2.0;
* }

protected boolean conditionAlgorithm3(HashSet<wPFIItem> itemset, wPFIItem item, double mu\_)

**Parameter:**

Itemset: a HashSet of wPFIItem objects.

Item: a wPFIItem object.

mu\_: a double value representing the minimum support threshold.

**Output:** a boolean flag indicating whether the given itemset and item satisfy the conditions of the algorithm.

* The implementation of additional conditions from Algorithm 3 in the research paper. This condition will be added in the algorithm 2 to early prune the candidates.
* public boolean conditionAlgorithm3(HashSet<wPFIItem> itemset, wPFIItem item, double mu\_) {
* if (itemset == null || item == null)
* return false;
* HashSet<wPFIItem> itemWrapper = new HashSet<>();
* itemWrapper.add(item);
* if (supportDict.get(itemset) == null)
* Pr(itemset);
* if (supportDict.get(itemWrapper) == null)
* Pr(itemWrapper);
* double mu\_X = supportDict.get(itemset);
* double mu\_I = supportDict.get(itemWrapper);
* if (mu\_X < mu\_ || mu\_I < mu\_)
* return false;
* if (mu\_X \* mu\_I < alpha \* transactionSize \* mu\_)
* return false;
* return true;
* }
  + 1. **wPFIItem class**

**Introduction:**

The wPFIItem class is implement for the representation of the item in the uncertain database. The class is created based on the Definition 1 of the definition mentioned in the chapter 1.

**Detail:**

Attribute:

private final int id;

private final double probability;

Method:

public int getId()

**Parameter:** None

**Output:** an integer number represent for item

* Get the item id.
* public int getId() {
* return id;
* }

public double getProbability()

**Parameter:** None

**Output:** a double number represent for probability of item

* Get the existential probability of the item.
* public double getProbability() {
* return probability;
* }

public boolean equals(Object object)

**Parameter:** object another item

**Output:** true if equal, otherwise false.

* Check if this item is equal to another.
* public boolean equals(Object object) {
* wPFIItem anotherItem = (wPFIItem) object;
* if ((anotherItem.getId() == this.getId())) {
* return true;
* }
* return false;
* }

public int hashCode()

**Parameter:** None

**Output:** a hash code as a int.

* Generate a hash code for the item.
* public int hashCode() {
* String string = "" + getId();
* return string.hashCode();
* }

Here are the support function to print out the item.

public String toString()

**Parameter:** None

**Output:** a string

* Get a string representation of this item.
* public String toString() {
* return getId() + " ";
* }

public String toStringWithProbability()

**Parameter:** None

**Output:** a string

* Get a sting representation of this item and its probability.
* public String toStringWithProbability() {
* return "(" + getId() + "," + probability + ") ";
* }
  1. **Asymptotic Complexity Analysis (The worst-case complexity)**

**Generate Weight Table:**

Generating the weight table involves iterating over all items, resulting in a time complexity of where is the number of items.

**Scan Find Size 1:**

This operation scans for frequent itemsets of size 1. In the worst case, it iterates over all items, resulting in a time complexity of .

**Scan Find Size K:**

Scanning for frequent itemsets of size involves iterating over candidate itemsets. In the worst case, this operation may have a time complexity of where is the number of candidate itemsets.

**wPFIApriori Generate:**

Generating candidate itemsets for the next size involves nested iterations. The worst-case time complexity may be influenced by the number of candidate itemsets and the size of the dataset. Let's denote these as and , respectively. The complexity may be

**Pr (Probability Calculation):**

The probability calculation involves nested loops and may depend on the dataset size The worst-case complexity might be .

**calculateMu\_:**

The binary search operation may have a time complexity of where is the size of the dataset.

**conditionAlgorithm3:**

This involves conditional checks and operations on sets. In the worst case, it may have a time complexity of .

* 1. **Benchmark**
     1. **Runtime based on msup changing**

Each dataset has 10000 lines.

**Dataset:** accidents.dat

**Setting:**

**A graph with a line and a line

Description automatically generated**

**Dataset:** accidents.dat

**Setting:**

**A graph with blue and orange lines

Description automatically generated**

**Dataset:** T40I10D100K.dat

**Setting:**

**A graph with a line and a blue line

Description automatically generated**

* + 1. **Runtime based on threshold t changing**

**Dataset:** connect.dat

**Setting:**

**A graph with blue and orange lines

Description automatically generated**

**Dataset:** accidents.dat

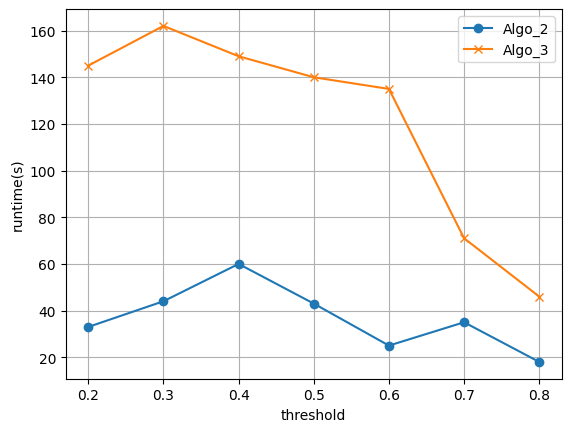
**Setting:**

**A graph with a line and a line

Description automatically generated**

**Dataset:** T40I10D100K.dat

**Setting:**



# Conclusions

To discover w-PFIs from uncertain databases, we experiment based on the paper named “Efficient weighted probabilistic frequent itemset mining in uncertain databases” and implement the java code to build the model for the support of w-PFI candidate using three pruning techniques mentioned in the paper to remove unpromising candidates immediately. With the result visualized at the benchmark part, the model is quite good on the **connect** and **accidents** dataset and bad on the **T40I10D100K** dataset.

Group GitHub link: <https://github.com/scuph-ng/weighted-frequent-itemset.git>

**REFERENCES**

1. Zhiyang Li | Fengjuan Chen | Junfeng Wu | Zhaobin Liu | Weijiang Liu (2020), Efficient weighted probabilistic frequent itemset mining in uncertain databases.