2010

Mining of uncertain Web log sequences with access history probabilities

Olalekan Habeeb Kadri
University of Windsor

Mining of Uncertain Web Log Sequences with Access History Probabilities

By

Olalekan Habeeb Kadri

A Thesis
Submitted to the Faculty of Graduate Studies through the School of Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2010

Your file Votre référence
ISBN: 978-0-494-62717-4
Our file Notre référence
ISBN: 978-0-494-62717-4

AUTHOR'S DECLARATION OF ORIGINALITY

ABSTRACT

An uncertain data sequence is a sequence of data that exist with some level of doubt or probability. Each data item in the uncertain sequence is represented with a label and probability values, referred to as existential probability, ranging from 0 to 1.

Existing algorithms are either unsuitable or inefficient for discovering frequent sequences in uncertain data. This thesis presents mining of uncertain Web sequences with a method that combines access history probabilities from several Web log sessions with features of the PLWAP web sequential miner. The method is Uncertain Position Coded Pre-order Linked Web Access Pattern (U-PLWAP) algorithm for mining frequent sequential patterns in uncertain web logs. While PLWAP only considers a session of weblogs, U-PLWAP takes more sessions of weblogs from which existential probabilities are generated. Experiments show that U-PLWAP is at least 100% faster than U-apriori, and 33% faster than UF-growth. The UF-growth algorithm also fails to take into consideration the order of the items, thereby making U-PLWAP a richer algorithm in terms of the information its result contains.

KEYWORDS
Uncertain data mining, frequent sequential patterns, Web log mining, existential probability generation, dirty data mining, Tree-based mining, probabilistic data mining.

DEDICATION

To all those whom I had leaned on through my sojourn in life. This is another good reason for you all to be part of my story.

ACKNOWLEDGEMENT

My sincere appreciation goes to my parents, wife and siblings. Your perseverance and words of encouragement gave me the extra energy to see this work through. Special thanks go to Dr. A.O. Abdulkadri and family. Your financial and moral support is key to seeing me through those challenging moments. I will forever be grateful.

I will be an ingrate without recognising the invaluable tutoring and supervision from Dr. Christie Ezeife. Your constructive criticism and advice at all times gave me the needed drive to successfully complete this work. The research assistantship positions helped as well!!

Special thanks go to my external reader, Dr. Dilian Yang, my internal reader, Dr. Alioune Ngom, and my thesis committee chair, Dr. Akshaikumar Aggarwal for accepting to be in my thesis committee. Your decision, despite your tight schedules, to help in reading the thesis and providing valuable input is highly appreciated.

And lastly to friends and colleagues at Woddlab, I say a very big thank you for your advice and support through out the duration of this work.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Data mining or knowledge discovery is generally referred to as the process of analyzing data from different views and situations and presenting it into useful information that can be used to make critical decisions about day to day running of a business or process. This process involves analysing data from many different dimensions or angles, categorizing and summarizing the relationships identified with the aim of increasing revenue and/or cutting time and costs of the processes represented by the data. Agrawal and Srikant (1996) define data mining as a way of efficiently discovering interesting rules from large databases. This area of study is motivated by the need to continuously look for solutions to decision support problems faced by large retail organisations.

Data mining typically involves creating databases or warehouses where data are then read using various techniques to analyse and forecast on future events. Algorithms that employ techniques from statistics, machine learning and pattern recognition are used to search large databases automatically. The process goes beyond simple data analysis. Continuous innovations in computer processing power, disk storage, data capture technology, algorithms and methodologies have all helped in realizing this objective. One key factor that is important towards ensuring that information derived from data mining is accurate is the use of broad range of representative data so that conclusions drawn from data can be good enough to represent the general situation. Related to this is also the aspect of ensuring proper cleaning is done on these representative data when required so that accurate results are generated.

1.1 Data mining approaches

The process of discovering interesting information from huge set of data often employs different techniques and approaches. Some of the approaches include:

- Classification
- Clustering
- Regression
- Association rule mining

Classification involves arranging data into predefined groups. This is the filtering technique used in grouping spam and non-spam mails. Some of the algorithms used here are nearest neighbours, Naïve Bayes classifier and neural network.

Clustering, like classification, also groups data into groups but using similarities that exist between data rather than predefined grouping. K-means algorithm is an example that is used with this approach. Regression method finds functions that model the data with least error. Genetic programming is one of such techniques used for this purpose.

Association rule mining involves finding relationships that exist among data items in a database. It involves finding patterns in the data items which are dependent on support count and confidence values of the rules establishing relationships between data items. Rules like if item A (antecedent) is purchased then item B (consequent) is also purchased can be used to calculate the support count and confidence of "if A then B" as follows:

Support ($A \Rightarrow B$) = number of occurrence of (A and B)/ Total number of transaction.

Confidence ($A \Rightarrow B$) = number of occurrence of (A and B)/ number of occurrence of (A)

The approach proposed in this research is based on this technique.

1.2 Web mining types

Web Mining is often referred to as the extraction of interesting and potentially useful patterns and implicit information from the activity related to the World Wide Web. Three categories in which web mining is usually viewed are:

- Web Content Mining
- Web Structure Mining
- Web Usage Mining.

Web content mining is the process of extracting knowledge from the content of web pages, documents or their descriptions. Wrappers are used to map documents into some models in order to be able to explore known structures in documents. Documents can either be directly mined or the results of some content search of search engines can be improved upon.

Web structure mining can be defined as the process of discovering knowledge from the World Wide Web organization and links between references and referents on the Web. Web structure mining involves making intelligent deductions from the links directed into and out of contents on the web. This inward and outward links indicates the richness or importance to which the content is to a particular topic. Counters attached to these links can be used to retrace the structure of the activities on the web content.

Web Log Mining is the process of extracting interesting patterns in web access logs. Since web servers record and accumulate data about user interactions whenever requests for resources are received, such information can be collected, cleaned and analysed for knowledge discovery. Information such as user behaviour as regards page visiting patterns can be studied in order to aid the design and organisation of such websites. It can also serve as attractive means of choosing where advertisements are placed on websites. The web log can be taken a step further by dynamically customizing the order and structure in which pages are displayed for each user based on their respective browsing history.

1.3 Uncertain data sequence

An uncertain data sequence is a sequence of data that exists with some level of doubt or probability. Each data item in the uncertain sequence is represented with a label and probability values, referred to as existential probability, ranging from between 0 and 1. This shows the level of assurance that an item exist in the sequence. A zero (0) value indicates the item does not exist while one (1) shows that the item is 100% present in the sequence. Table 1 gives an example of uncertain sequences.

User_ID	Uncertain sequence
10	(a:1,b:0.5,c:0.75,d:0.5)
20	(a:1,d:0.5,e:0.5,d:0.5)
30	(a:1,b:0.5,d:0.5,b:0.5)

Table 1: Example of uncertainty in sequences

User_ID 10 for example, is represented by (a:1, b:0.5, c:0.75, d:0.5) indicating that item 'a' has 100% probability of existence. Items b, c and d however have 50%, 75%, and 50% existential probabilities respectively.

The problem of mining frequent sequence in uncertain data sequence is that of determining those sequences whose expected support count are greater than or equal to a specified threshold called minimum support. The expected support of an uncertain sequence is given by the sum of product of existential probability of the constituent items of the sequence (Chui et al., 2007). For example, given the database as shown in table 1, the expected support count of sequence 'ab' is calculated as (1 x 0.5) + (1 x 0.5) = 1. This is then compared with a specified minimum support. If the minimum support is 1, the sequence 'ab' is said to be frequent.

1.4 Problems and applications of mining with dirty and uncertainty data

Cong et al. (2007) show that real world data often contain inconsistencies and errors. The challenge is therefore in removing these errors (dirt) while ensuring the integrity of the data is maintained. It is of high importance that the original meaning of the data is not, in any way, changed. They demonstrate that the traditional functional dependencies in database (Normal forms) may not fully guarantee the correct content of the database. Given an order database shown as follows:

Order (id, name, AC, PR, PN, STR, CT, ST, zip). Each order contains name, unique item id, and price (PR) of the item. It also contains the phone number, which contains area code (AC) and phone number (PN), and the address that indicates street (STR), city (CT), state (ST) and Zip code (Zip) of the customer buying the item. The functional dependencies of the Order database are as follows:

f_1[AC, PN] $\longrightarrow$ [STR, CT, ST] f_2 [Id] $\longrightarrow$ [name, PR]

f_3[Zip] $\longrightarrow$ [CT, ST] f_4 [CT, STR] $\longrightarrow$ [Zip]

Where AC, PN, STR, CT, ST are Area Code, Phone Number, Street, City and State respectively. Id represents the item identification number and PR is its price. This design for example only specifies that AC and PN identify [STR, CT, ST] but fails to restrict values that can be entered to represent real life situation. Area code 212 for example, can be used to represent any city in the US. However in real life situation, 212 can only represent New York City (NYC) as a city and New York (NY) as a state. A conditional functional dependency (CFD) φ is then defined as ([AC, PN] $\longrightarrow$[STR, CT, ST]), with an additional constraint T defined on attributes of the table that stipulates the kind of values that may exist together in the tables while still ensuring that each AC and PN uniquely identify each STR, CT, ST combinations. An instance of the table showing the constraint T is shown in Table 2:

AC	PN	STR	CT	ST
-	-	-	-	-
212	-	-	NYC	NY
610	-	-	PHI	PA
215	-	-	PHI	PA

Table 2: Conditional functional dependencies

With these additional constraint, all entries having 212 as AC must be from NYC and NY while those from PHI and PA must either have 610 or 215 AC. Non conforming tuples are cleaned accordingly.

Cong et al. (2007) stress that it is in order to guarantee the accuracy of cleaned data after the application of data cleaning methodology, inspection of a manageable sample of the cleaned data is needed. This ensures the accuracy of the repairs found is above a predefined bound with high confidence. This is done by drawing a sample of K tuples from the repaired database. A predefined inaccuracy threshold ξ is specified. The sample is partitioned into M strata where M< K. Each stratum is then assigned its inaccuracy threshold ξi such that $\xi_1 + \xi_2 + \ldots + \xi_M = 1$. Each ξi can be adjusted depending on the likelihood that inaccurate tuples may exist in a particular stratum. The number of tuples drawn from each stratum is given by the product of ξi and K (total number of tuples). The number of inaccurate tuples e_i in each stratum is noted. The test statistic (z) is computed as follows:

The ratio of number of tuples in stratum i to the number of tuples drawn from the stratum to check for inaccuracy is defined by $s_i = P_i / (\xi_i . K)$.

The inaccuracy rate, $p = (\sum e_i . s_i) / (\sum P_i . s_i)$, where P_i is the number of tuples in each stratum i.

The test statistic $z = (p - \xi) / (\sqrt{(\xi(1 - \xi)/K)})$ where p is the inaccuracy rate of the sample

The test statistic z is tested against the critical value z_∞ at confidence level δ. If $z \leq - z_\infty$, inaccuracy level of the cleaned data is said to be below the threshold ξ making the cleaned data acceptable for use.

Chiang and Miller (2008) go a step further by automatically generating conditional functional dependencies (CFDs) for a given relation with sample data. These CFDs are then used to remove the dirty data that fail to comply with the CFDs without any need of human inspection. Given the relation showing US Census Data:

Tuple	CLS	ED	MR	OCC	REL	GEN	SAL
t_1	Private	Bachelors	Married	Exec-mgr	Husband	Male	>50K
t_2	Private	Bachelors	Married	Prof-specialty	Husband	Male	>50K
t_3	Self-emp	Masters	Married	Exec-mgr	Wife	Male	>50K
t_4	?	HS-grad	Divorced	?	Not-in-family	Male	>50K
t_5	Self-emp	Masters	Married	Admin	Wife	Female	>50K
t_6	Never-worked	7th-8th	Divorced	?	Not-in-family	Male	≤50K
t_7	Self-emp	HS-grad	Never-married	Farming	Own-child	Male	≤50K
t_8	Local-gov	Some college	Never-married	Admin	Own-child	Female	≤50K
t_9	State-gov	Masters	Divorced	Prof-specialty	Own-child	Male	>50K
t_{10}	?	Bachelors	Divorced	?	Not-in-family	Female	≤50K
t_{11}	Self-emp	Some college	Never-married	Machine-up	Not-in-family	Female	>50K

Table 3: The US-Census sample data

The CFDs are generated by partitioning a relation into attributes of equivalence classes. CFDs are then defined between classes $X \rightarrow Y$ when there exist some classes in X that are fully contained in some classes in Y. The CFDs are then defined on the conditions in which the classes are contained in each other. If for example, the following classes, denoted by Π, are defined from the census sample data based on the similarities in their values:

$\Pi_{MR} = \{\{1,2,3,5\}, \{4,6,9,10\}, \{7,8,11\}\}$

$\Pi_{REL} = \{\{1,2\}, \{3,5\}, \{4,6,10,11\}, \{7,8,9\}\}$

$\Pi_{(R,M)} = \{\{1, 2\}, \{3, 5\}, \{4, 6, 10\}, \{7, 8\}\{9\}, \{11\}\}$

$\Pi_{ED} = \{\{1, 2, 10\}, \{3, 5, 9\}, \{4, 7\}, \{6\}, \{8, 11\}\}$

$\Pi_{CLS} = \{\{1, 2\}, \{3, 5, 7, 11\}, \{4, 10\}, \{6\}, \{8\}, \{9\}\}$

$\Pi_{(C,E)} = \{\{1, 2\}, \{10\}, \{3, 5\}, \{4\}, \{6\}, \{7\}, \{8\}, \{9\}, \{11\}\}$

$\Pi_{(M,E)} = \{\{1, 2\}, \{10\}, \{3, 5\}, \{4\}, \{6\}, \{7\}, \{9\}, \{8, 11\}\}$

$\Pi_{(M,E,C)} = \{\{1, 2\}, \{10\}, \{3, 5\}, \{4\}, \{6\}, \{7\}, \{9\}, \{8\}, \{11\}\}$

The class Π_{MR} is formed by grouping tuple numbers with same value in column MR. That of $\Pi_{(C,E)}$ is formed by grouping tuples with same value in both CLS and ED and so on. Rules are then formed by establishing edges between classes. For edge $X \rightarrow Y$ = (REL $\rightarrow$(REL, MR)). The classes {1, 2}, {3,5} in REL are also contained in (REL, MR). REL is then conditioned on common values in {1, 2} and {3, 5} to give the CFDs required. The CFDs are: ({REL = 'Husband'} $\rightarrow$ MR) and
({REL = 'Wife'} $\rightarrow$ MR) or more explicitly as ({REL = 'Husband'} $\rightarrow$ Married) and
({REL = 'Wife'} $\rightarrow$ Married).

By going to the next level, when the 'X' part of the edge has more than an attribute, one of the attributes is made the conditional attribute if a match is found.

If $X \rightarrow Y$ = (ED, MR)$\rightarrow$(ED, MR, CLS)). The classes {1, 2} and {3, 5} in (ED, MR) is contained in (ED, MR, CLS). If MR is made the conditional attribute from (ED, MR), a class in Π_{MR} that only contains tuples from {1, 2} and {3, 5} must be found in order to define a rule. Since Π_{MR} has {1,2,3,5}, a conditional rule can be formed on the common value in {1,2,3,5} = Married. Therefore, the rule: ({MR = 'Married', ED} $\longrightarrow$ {CLS}). This means that MR must be married for MR and ED to functionally determine CLS.

The main problem with dirty data is being able to strike a balance between making sure dirty data are not used for mining and at the same time ensuring the cleaning process does not introduce another set of unwanted data. While the Cong et al. (2007) approach needs sampling and physical inspection to guarantee its correctness, the automatic approach of Chiang and Miller (2008) depends on the cleanliness of the data that is used to generate the rules.

In recent times, there has been interest in mining data generated from non-traditional domains such as sensor networks and location based services. Such data are generated with some level of uncertainty due to measurement inaccuracies, sampling error and network latencies. Location based systems as found in GSM (Global System for Mobile communication) are used in delivering several value added services in the mobile telephoning industry. It can be used in determining location of a subscriber or the nearest banking cash machine. However, due to the unreliability of radio signal that is used to relay information from the nearest mobile base station (tower), information delivered might sometimes be prone to errors. Fluctuations in signals and network can cause some farther base stations to serve a particular mobile subscriber and as a result deliver information on a farther location. Therefore mining data generated by such systems demands attaching some uncertainties unto them.

Sensor networks that combine computational functionalities with physical sensing capabilities could also transmit errors of the physical entity such as temperature that they are designed to measure. Sensors designed to capture temperature values might be corrupted or biased if placed near a power generating plant or an air conditioner. Analysis based on these types of data may therefore be inaccurate if the uncertainties that may be present are not equally accounted for.

One main problem associated with mining uncertain data is how such data are objectively generated. Chui et al. (2007) and Leung et al. (2008) also suggested that historical behaviour of entities can be used for presenting uncertainties that might be attached to their current behaviour. Their work does not show exactly how this can be done objectively. The use of historical data can be used objectively in arriving at a more unbiased result during mining based on sample data. Uncertainties can also be useful in analysing how unique users may find a particular webpage useful. Such analysis can then be used in improving the design of such websites. Web logs, though are doubtless facts, can be represented with uncertainty when history of log sessions for each user is used to compute the consistency of the user's behaviour based on the most current web log session.

Given four different web log sessions as shown in Tables 4a, 4b, 4c and 4d. The tables represent access histories for users 10, 20 and 30 at different times 1, 2, 3 and 4. User IDs 10 and 30 participates in all the 4 sessions while user ID 20 only take part in twice. The most current session is time 4. The four web log sessions are then merged into one database and ordered by user ID and time so that the most current web log for each user comes first as shown in Table 5.

User_ID	Time	Web log
10	1	(a,c,a,c)
20	1	(a,c,b)
30	1	(a,e)

a

User_ID	Time	Web log
10	2	(a,d,c,a)
20	2	(a,d,e,d)
30	2	(a,e,d,e)

b

User_ID	Time	Web log
10	3	(a,d,a,b)
30	3	(a,c,e,b)

c

User_ID	Time	Web log
10	4	(a,b,c,d)
30	4	(a,b,d,b)

d

Table 4: The 4 different web log sessions

User_ID	Time	Web logs
10	4	(a,b,c,d)
	3	(a,d,a,b)
	2	(a,d,c,a)
	1	(a,c,a,c)
20	2	(a,d,e,d)
	1	(a,c,b)
30	4	(a,b,d,b)
	3	(a,c,e,b)
	2	(a,e,d,e)
	1	(a,e)

Table 5: The merged and sorted logs for the different sessions

The existential probability, prob (u(e)), of each event e, for each user u, in the most current log is given as:

$$\text{prob}\ (u(e)) = \frac{\text{Number of existence of e in any record with u}}{\text{Total number of records with u}}$$

The likelihood of having the latest log to be a true reflection of how each user browses the site based on the historical behaviour is calculated as follows:

User ID 10: Since 'a' appears at least once in all the 4 logs, existential probability of 'a' is given as 4/4 = 1. Item 'b' appears only in the first two logs, its existential probability is 2/4 = 0.5. Item 'c' appears 3 times, its existential probability is 3/4 = 0.75 and that of d is 2/4 = 0.5. Using the same method, user ID 20's existential probability values for the latest log is:

'a' = 2/2 = 1

'd' = 1/2 = 0.5

'e' = 1/2 = 0.5

User ID 30:

'a' = 4/4 = 1

'b' = 2/4 = 0.5

'd' = 2/4 = 0.5

The uncertain sequence to be mined is therefore given as in Table 6 :

User_ID	Web logs
10	(a:1,b:0.5,c:0.75,d:0.5)
20	(a:1,d:0.5,e:0.5,d:0.5)
30	(a:1,b:0.5,d:0.5,b:0.5)

Table 6: The generated uncertainty in the logs

User_ID 10 for example, is represented by (a:1, b:0.5, c:0.75, d:0.5) indicating the consistency of the current behaviour based on the historical behaviour. There is a 100% certainty that User_ID 10 always visit page 'a'. In the same manner, the probability of pages b, c, d are 50%, 75%, and 50% respectively.

1.5 Thesis problem and contribution

This thesis is set out to propose a sequential web log mining solution for mining uncertain weblog sequences with access history probabilities using the PLWAP tree mining approach. The works of Agrawal and Srikant (1995), Pei et al. (2000) and Ezeife and Lu (2005) are all based on precise data sequences with 100% certainty. Chui et al. (2007) proposed U-apriori, an uncertain data implementation of apriori algorithm. Leung et al. (2008) proposed UF-growth that is a non-sequential mining approach of frequent patterns. The contributions of this thesis are therefore as follows:

- The approach (U-PLWAP) is the first tree based sequential mining approach for data with uncertainty.
- U-PLWAP makes use of historical web log session in calculating existential probability of web log sessions.
- The speed of the algorithm is improved by collapsing events with the same label but different existential probabilities into one single node during the construction of the U-PLWAP tree. This makes the U-PLWAP tree more scalable in terms of number of nodes.
- In order to effectively mine frequent patterns in U-PLWAP, sequence of cumulative products of existential probabilities of events found along each temporary root nodes (from the root node) are dynamically generated for each new sets of temporary root nodes.
- The general performance of U-PLWAP is at least 2 times faster than U-apriori. The U-PLWAP algorithm is also at least 33% faster than UF-growth. This is addition to the richer result generated by U-PLWAP (frequent sequential patterns) as against UF-growth that is a frequent non sequential pattern algorithm.

2. RELATED WORKS

There have been several research findings on the general area of data mining. Specifically, the problem of mining web logs can be traced back to the association rule mining technique originally introduced by Agrawal and Srikant (1995). Etzioni (1996) state that web mining can be useful for server performance enhancements, restructuring of website and direct marketing in e-commerce. This has made research in this area an important one, given the role internet and websites play in our day to day activities. Web log mining, synonymous with Web usage mining, is one of the three areas of web mining identified by Madria et al. (1999) and Borges and Leven (1999). Technically, the problem of sequential mining of web logs is that of finding all ordered sequence of web pages accessed whose support count is greater than a specified minimum threshold (Ayres et al., 2002). Cooley (2003) defined web usage mining as "the application of data mining techniques to web click stream data in order to extract usage patterns". Han and Kamber (2006) also gave an insight to the area of data mining including the sequential mining of web logs.

The discussion in this chapter is grouped into those methods that are apriori and pattern-based. The algorithms with uncertainty are also discussed in a separate section.

2.1 Apriori based methods

This section gives an introduction into the problem of sequential pattern mining and the early methods used to solve this problem. It gives insight into the general problem of sequential mining which can be adapted to the web usage mining environment.

Agrawal and Srikant (1995) introduce the problem of mining frequent sequential patterns. They state that early works by Agrawal et al. (1993) only concentrate on discovering patterns within transactions. Agrawal and Srikant (1995) introduce and addressed the problem of discovering sequential patterns in transactional databases where transactions are taken as sequences with each sequence containing one or more items. The authors set out to identify inter-transaction patterns, treating each transaction (itemsets) as a unit rather than treating items as a unit. For example, itemsets ⟨(1,5) (2) (3) (4,7)⟩ can be considered for a single user or customer in the approach introduced by Agrawal and Srikant (1995). That is, more than one itemset per record/user is considered for mining as against single itemset per user in Agrawal et al. (1993).

2.1.1 Apriori algorithm (AprioriAll)

Agrawal and Srikant (1995) propose AprioriAll algorithm to discover sequential patterns in a transaction database. The algorithm finds sequential patterns in transaction database where several transactions can be done by a given customer as shown in Table 8. Such transactions are grouped to form a sequence for each customer as given in Table 7.

Cust_Id	Time	Transaction
1	1	1,5
1	2	2
1	3	3
1	4	4,7
2	1	1,8
2	2	3
2	3	4
2	4	3,5
3	1	1
3	2	2
3	3	3
3	4	4
4	2	1
4	3	3
4	4	5
5	1	4
5	2	5

Table 8: The sample transaction database

Cust_id	Sequence
1	⟨(1,5) (2) (3) (4,7)⟩
2	⟨(1,8) (3) (4) (3,5)⟩
3	⟨(1) (2) (3) (4)⟩
4	⟨(1) (3) (5)⟩
5	⟨ (4) (5)⟩

Table 7: The sequence formed

The sample transaction in Table 8 shows transactions done by each customer at different time. The transactions are then grouped by customer ID so that customer ID 1 for example has a sequence of transactions ⟨(1,5) (2) (3) (4,7)⟩ as shown in table 6. Each transaction grouping in a sequence is known as itemset. For example, ⟨(1,5) and (2) are itemsets. In order to mine frequent sequence pattern, a minimum threshold referred to as the minimum support must be specified. The database is then scanned to record the occurrence of each itemset in order to discover if such itemset is frequent. Itemsets with count that is greater or equal to the minimum support are said to be frequent. The set of frequent itemsets are referred to as the Large itemsets. Given the sequence shown in Table 8, the algorithm operates in 5 phases; Sort phase, Litemset (Large itemset) phase, Transformation phase, Sequence phase and Maximal phase.

The sort phase prepares the transaction database by grouping transactions according to the customer ID thereby forming a sequence for each customer ID. Using the sequence in Table 8, AprioriAll works by first ordering the transaction database by customer ID and time. It is then implicitly changed to the form shown in Table 7. Given that the minimum support is 2, the Large itemset phase then finds all frequent (Large) 1-sequences by scanning the database for itemsets that occur more than once provided that the minimum support specified is 2. Out of the possible itemsets (candidate itemsets) {(1), (2),(3), (4), (5), (1,5), (1,8), (3,5) and (4,7)}, the frequent 1-sequences are as shown in Table 9.

1-Sequences	Support
(1)	4
(2)	2
(3)	4
(4)	4
(5)	4

Table 9: The Large 1-sequence

Transformation phase is done by removing the non-frequent items/itemsets. It also replaces all transactions with the set of all Large itemsets contained in that transaction. A mapping of the transformed database is also shown in Table 10 given that large itemsets (1). (2), (3), (4) and (5) are mapped to 1, 2, 3, 4, 5 respectively. Items 7 and 8 are not contained in Large 1-sequences and have been removed in the transformed sequence.

Cust_id	Sequence Before Transformation	Sequence After Transformation (deleting small items)
1	⟨(1,5) (2) (3) (4,7)⟩	⟨{(1),(5)} {(2)} {(3)} {(4)}⟩
2	⟨(1,8) (3) (4) (3,5)⟩	⟨{(1)} {(3)} {(4)} {(3),(5)}⟩
3	⟨(1) (2) (3) (4)⟩	⟨{(1)} {(2)} {(3)} {(4)}⟩
4	⟨(1) (3) (5)⟩	⟨{(1)} {(3)} {(5)}⟩
5	⟨(4) (5)⟩	⟨{(4)} {(5)}⟩

Table 10: The transformed sequence

The Sequence phase performs an apriori-gen join on the frequent 1-sequence in Table 9 to form candidate sequence C_2 as shown in Table 14. Large 2-sequences (L_2) are then derived from C_2 by selecting those with support greater than or equal to 2 since the minimum support is 2. L_2 is then used in an apriori-gen join to generate C_3. Apriori-gen join works by performing a self join on L_{k-1} in order to arrive at C_k only when the first k-2 items are equal. The generated C_k is then pruned by removing any sequence in C_k that has any of its k-1 subsequence not in L_{k-1}. For example, C_3 is derived by an apriori-gen join on L_2 when the first items in each L_2 are the same. That is, (1 2) joins with (1 3) to form (1 2 3) because they both have '1' as the first item. C_3 is then pruned by removing any sequence that has any of its 2-subsequences not in L_2. For example, sequence (1 3 2) is pruned from C_3 since its sub-sequence (3 2) is not in L_2. This process is recursively done to generate new C_k and L_k until no more C_k is formed or no frequent sequence found. The Tables 11 to 16 show the process in details.

Seq	Count
(1 2)	2
(1 3)	4
(1 4)	3
(1 5)	3
(2 3)	2
(2 4)	2
(2 5)	0
(3 4)	3
(3 5)	2
(4 5)	2

Table 14: C2

Seq	Count
(1 2)	2
(1 3)	4
(1 4)	3
(1 5)	3
(2 3)	2
(2 4)	2
(3 4)	3
(3 5)	2
(4 5)	2

Table 13: L2

Seq	Count
(1 2 3)	2
(1 2 4)	2
(1 2 5)	0
(1 3 4)	3
(1 3 5)	2
(1 4 5)	1
(2 3 4)	2
(3 4 5)	1

Table 12: C3

Seq	Supp
(1 2 3)	2
(1 2 4)	2
(1 3 4)	3
(1 3 5)	2
(2 3 4)	2

Table 11: L3

Seq	Count
(1 2 3 4)	2
(1 3 4 5)	0

Table 16: C4

Seq	Count
(1 2 3 4)	2

Table 15: L4

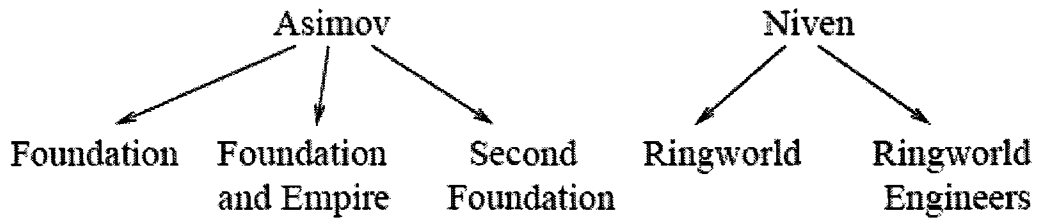
Maximal phase is then executed by finding only frequent itemsets that are not contained in any other large itemset. Itemsets {(4 5), (1 3 5), (1 2 3 4)} are not contained in any of the large sequences. The maximal sequences are therefore {(4 5), (1 3 5), (1 2 3 4)}.

2.1.2 The GSP algorithm

Srikant and Agrawal (1996) introduced GSP, a more flexible approach into their earlier work stated above. They show that the problem of discovering sequential patterns with earlier approaches fail to take into account time constraints in the sequence to be considered. They also stress that existing approaches have rigid definitions of transactions. Grouping of items into hierarchies (taxonomies) can also be found to be missing in the earlier algorithms. The authors proposed GSP (Generalized Sequential Patterns) to cater for these limitations. The GSP algorithm is made more flexible by including time constraints, flexibility in the definition of transaction and inclusion of taxonomies. The time constraint is used to further extend the flexibility of the algorithm by specifying the time limit within which a particular pattern can be considered frequent. The flexibility in transaction definition also introduces the freedom to regroup transactions based on a specified time interval within which transactions can be combined into one unit (one itemset). This specified time is referred to as the sliding window. The inclusion of taxonomy helps in further showing the hierarchy along which a frequent pattern can be expressed. The example below shows the transactions of books sold in a store. For simplicity, the transaction time is represented in number of days.

Sequence-Id	Transaction Time	Items
C1	1	Ringworld
C1	2	Foundation
C1	15	Ringworld Engineers, Second Foundation
C2	1	Foundation, Ringworld
C2	20	Foundation and Empire
C2	50	Ringworld Engineers

Taxonomy $\mathcal{T}$



Example 2.1 The example demonstrating additional functionalities of GSP algorithm.

If the minimum support is specified as 2, the frequent 2- element patterns are: {(Ringworld) (Ringworld Engineers)} and {(Foundation) (Ringworld Engineers)}. However, when a sliding window of 7 days is given, the pattern {(Ringworld) (Ringworld Engineers)} also becomes frequent since the first 2 transactions of C1 becomes one transaction. Setting a maximum constraint of 30 days generates no 2-element frequent pattern since Ringworld Engineer in C2 was bought after the 30-day limit.

Frequent patterns can also be expressed in form of their taxonomies. For example, patterns supporting {(Foundation) (Ringworld Engineers)} will also support {(Asimov) (Niven)}. This is made possible by also specifying the taxonomy as part of the input data. For example, {(Foundation) (Ringworld)} will be represented as {(Foundation, Ringworld, Asimov, Niven)}.

The GSP algorithm is a multiple-pass solution. The first pass considers the items individually and generate the frequent 1-sequence for those with support count greater than the minimum support. The 1-itemset (L_1) is used to generate the next set of candidate sequences (C_2) by a join operation with itself. Subsequent C_k are generated from L_{k-1}. A join operation (apriori gen join) is carried out only when the subsequence generated by removing the first item in sequences in L_{k-1} is the same as removing the last item during the self join on L_{k-1}. The candidate sequences are searched for using the forward and backward phase algorithm. Pruning operation is also performed to remove those candidate sequences whose k-1 contiguous sub-sequences are not frequent. The set of L_k sequences are then generated. The algorithm ends when there are no more sequences generated in L_k .

Using Table 17 as an example, frequent 3-sequences is self joined using the apriori gen join which states that a join can only occur if by removing the first item in sequences in L_{k-1} , the subsequence left is the same as removing the last item during the self join on L_{k-1}. For example, sequence ⟨(1,2) (3)⟩ joins with ⟨(2) (3,4)⟩ since the last 2 items (2,3) in the first sequence and the first 2 items (2,3) in the second sequence are the same. Equally, ⟨(1,2) (3)⟩ joins with ⟨(2) (3) (5)⟩. The join operation then produces ⟨(1,2) (3,4)⟩ and ⟨(1,2) (3) (5)⟩. Sequence ⟨(1,2) (3) (5)⟩ is however pruned since its contiguous sub-sequence ⟨(1,2) (5)⟩ is not frequent (not in frequent 3-sequence).

Frequent 3-Sequences	Candidate 4-Sequences	
	after join	after pruning
⟨(1, 2) (3)⟩	⟨(1, 2) (3, 4)⟩	⟨(1, 2) (3, 4)⟩
⟨(1, 2) (4)⟩	⟨(1, 2) (3) (5)⟩	
⟨(1) (3, 4)⟩		
⟨(1, 3) (5)⟩		
⟨(2) (3, 4)⟩		
⟨(2) (3) (5)⟩		

Table 17: The table explaining GSP algorithm

The GSP algorithm checks for candidate sequence in 2 phases; the forward and backward phase. The forward phase finds successive elements of candidate sequence in a sequence databases given that the difference between the start time of the new element and the end time of the previous element is within the maximum time (constraint) specified. When the difference is more than the maximum time, the algorithm calls the backward phase. If no element(s) is found in this phase, the sequence database does not contain the candidate sequence.

The backward phase backtracks and checks the previous elements. This is to ascertain that no occurrence of the immediate previous element has its start time earlier than the difference between end time of the present element and the maximum gap constraint. The backtracking continues until such an occurrence is found or the very first element of the candidate sequence is pulled up. A new set of elements are then searched with the forward phase starting from the last point of backtrack.

Transaction-Time	Items
10	1, 2
25	4, 6
45	3
50	1, 2
65	3
90	2, 4
95	6

Table 18: The data sequence for candidate sequence check

If Table 18 is used as an example with maximum gap of 30, minimum gap of 5 and window size 0. Candidate sequence ⟨(1, 2) (3) (4)⟩ can be searched by first finding (1, 2) at transaction time 10 and then (3) at time 45. Because the time between the 2 elements is 35 days (greater than max gap of 30), element (1, 2) is pulled up by the backward phase of the algorithm. The first occurrence of (1, 2) with transaction time after 15 (45 – 30) is searched for. Since no such element exists and the first element in the candidate sequence is pulled up already, the forward phase continues with a fresh search of the candidate sequence. The first occurrence of (1, 2) is found at time 50. The next element (3) also

found at 65. Since the gap between these two elements is within thin the minimum and maximum gap, the next element (4) is then found at transaction time 90
Which is also within the maximum gap relative to immediate past element (3) found at time 65. Therefore the data sequence shown in Table 18 contains the candidate sequence ⟨(1, 2) (3) (4)⟩ under the conditions stated.

2.2 Pattern approaches

2.2.1 FP-growth algorithm

Han et al. (2004) aim at solving problem of candidate set generation during frequent pattern mining process. They found that candidate set generation can be costly especially when many patterns are present and when such patterns are long.

Han et al. (2004) proposed a frequent mining algorithm, FP-growth, based on frequent pattern tree (FP-tree) data structure. Through this, the authors contributed to the study by ensuring large database is compressed into FP-tree therefore removing repetitive database scan. This divide and conquer, conditional mining approach also remove candidate set generation.

The FP-tree is built on the intuition that if frequent items are used to re-order items in the database, multiple transactions sharing same itemset can be represented with the same path in FP-tree by registering their counts. FP-tree is constructed after a first scan of the database is carried out where frequent items are found and ordered. The order is then used to enter items into the FP-tree during the second scan. Given the database below:

TID	Items Bought	(Ordered) Frequent Items
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

Table 19: The sample database

It is assumed that the minimum support is 3. The items are re-ordered according to descending order of frequent items (f:4, c:4, a:3, b:3, m:3, p:3). Items h, I, j, k, l have been removed from the database since they are not frequent.

The items are then inserted into the FP-tree in the ordered fashion, registering the count on each occasion. A header link table is also constructed to show the order in which nodes of same type are inserted in order to aid mining process. The FP-tree generated is as given below:

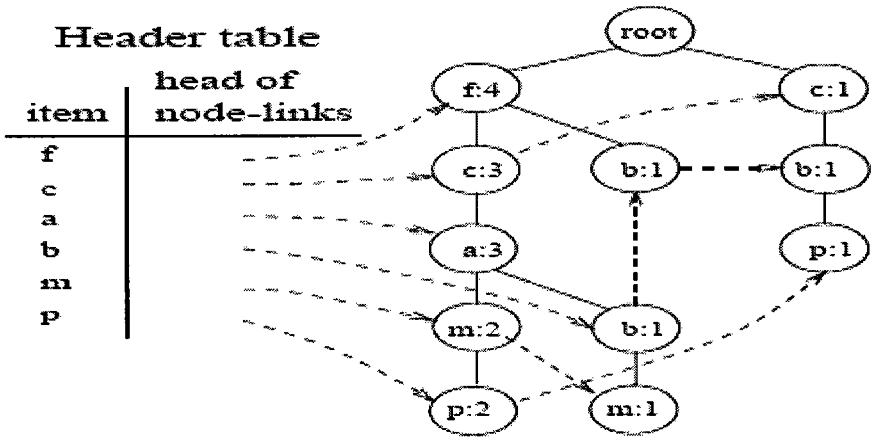


Figure 1: The constructed FP-tree

The mining process is done by starting from the bottom of the header table. Item p has 2 tree paths f:4, c:3, a:3, m:2, p:2 and c:1, b:1, p:1. Removing p and ensuring only counts for item p are present gives p's conditional base tree: f:2, c:2, a:2, m:2, and c:1, b:1. Only item c make the minimum count of 3 (sum), therefore forms p's conditional FP-tree (c:3). Pattern 'cp:3' is therefore frequent. The process is then repeated for all items on the header table. The conditional FP-tree is repeatedly mined when more then one frequent items are found. Table 20 summarises the operation:

Items	Conditional pattern base	Conditional FP-tree	Final Frequent pattern
p	(f:2,c:2,a:2,m:2), (c:1,b:1)	c:3	cp:3
m	(f:2, c:3,a:2), (f:1,c:1,a:1,b:1)	f:3, c:3, a:3	am:3, cm:3, fm:3, cam:3, fam:3, fcam:3
b	(f:1,c:1,a:1), (f:1), (c:1)	-	-
a	(f:3,c:3)	f:3, c:3	ca:3, fa:3, fca:3
c	(f:3)	f:3	Fc:3
f	-	-	-

Table 20: The FP-growth mining process

This approach is however limited to finding frequent patterns in non-sequential data.

2.2.2 Freespan

Han et al. (2000) proposed FreeSpan in order to solve the problem of sequential pattern mining. It also removes some of the drawbacks of Apriori algorithm including, several scan of the database, generation of huge candidate sequences and difficulty associated with long sequential patterns.

The authors' approach is based on the integration of frequent pattern approach with that of sequential patterns and uses projected databases to limit the search and growth of subsequence fragments. Han et al. (2000) contributions include design of frequent item matrix where frequent length 2-sequences are discovered. The authors also defined how generation of item repeating patterns and projected databases for each frequent item are found.

The algorithm works by using the first scan to generate the frequent items arranged in order on f-list. The second scan of the database then generates the frequent item matrix. From the matrix, the frequent 2-length sequences are noted. The sequences corresponding to the frequent diagonal elements are noted in the annotation for repeating items. Other non-diagonal sequences are also noted when the equivalent diagonal elements of the constituent items are frequent. Annotations that form triple F[i,j], F[k,j], F[i,k] with i<j and k < i (all corresponding pairs are frequent) are then used in defining projected column set by including k and annotations containing i and j. The database is then scanned the third time to generate the final item repeating patterns and the projected databases.

Given the sequence database as shown below:

Sequence_id	Sequence
10	$\langle (bd)cb(ac)\rangle$
20	$\langle (bf)(ce)b(fg)\rangle$
30	$\langle (ah)(bf)abf\rangle$
40	$\langle (be)(ce)d\rangle$
50	$\langle a(bd)bcb(ade)\rangle$

Table 21: The sample sequence database

The frequent item list are generated and ordered as follows: (b:5, c:4, a:3, d:3, e:3, f:3). A 6x6 frequent item matrix is then formed with the frequent items. A triangular matrix is then formed since half of the matrix is similar. The diagonal elements have one value while other cells have three values with the first two representing the count of the constituent items in the database when their positions are noted while the last value represents the count when they both occur together. Table 22 represents this.

b	4					
c	(4,3,0)	1				
a	(3,2,0)	(2,1,1)	2			
d	(2,2,2)	(2,2,0)	(1,2,1)	1		
e	(3,1,1)	(1,1,2)	(1,0,1)	(1,1,1)	1	
f	(2,2,2)	(1,1,0)	(1,1,0)	(0,0,0)	(1,1,0)	2
	b	c	a	d	e	f

Table 22: The frequent item matrix.

Taking the example of the second cell on the first column (4,3,0). This means ⟨b,c⟩ occurs 4 times, ⟨c,b⟩ 3 times and 0 means both ⟨(b,c)⟩ never occur together at a time. The frequent length 2 sequences, annotation on repeating items and annotation on projected databases are then generated as in Table 23:

Item	Output length-2 sequential pattern	Ann. on repeating items	Ann. On projected DB
f	⟨bf⟩:2, ⟨fb⟩:2, ⟨(bf)⟩:2	$\{b^+ f^+\}$	-
e	⟨be⟩:3, ⟨(ce)⟩:2,	$\langle b^+ e\rangle$	⟨(ce)⟩: {b}
d	⟨bd⟩:2, ⟨db⟩:2, ⟨(bd)⟩:2, ⟨cd⟩:2, ⟨dc⟩:2, ⟨da⟩:2,	$\{b^+ d\}$, $\langle d\ a^+\rangle$	⟨da⟩:{bc}, {cd}:{b}
a	⟨ba⟩:3, ⟨ab⟩:2, ⟨ca⟩:2, ⟨aa⟩:2,	$\langle a\ a^+\rangle$, $\{a^+ b^+\}$ $\langle d\ a^+\rangle$	⟨ca⟩:{b}
c	⟨bc⟩:4, ⟨cb⟩:3	$\{b^+ c\}$	-
b	⟨bb⟩:4	$\langle b\ b^+\rangle$	-

Table 23: Pattern generation

The length 2-sequential patterns are those entries that are greater than or equal to minimum support of 2. Since F[b,b] and F[f,f] are frequent, b and f can be repeated in as many times as possible (+) and in any order ({}) as shown in the annotation for repeating item column. Since no other frequent item exist with f apart from b, the annotation for projected database is null. For the second row (e), F[e,e] is infrequent, as a result e is not a repeating item as shown $\langle b^+,e\rangle$. Angular bracket is also used to restrict order of the item since only one of the 3 values of (b,e) on the item matrix is frequent (3,1,1). Set notation ({}) is used if otherwise to indicate that both (b,e) and (e,b) are frequent.

The items on row 2 also form a triple F[c,e], F[b,e], F[b,c]. Set {b} is therefore included in the projected database with the annotation containing c and e. In this case, ⟨(ce)⟩ is used since it is the only frequent annotation. The third database scan is then done using the item repeating annotation and projected database as guide to obtaining frequent sequential pattern of longer length. The mining operation is then restricted to the patterns contained in the projected database header set. The frequent patterns obtained in the third scan are as follows: {⟨bbf⟩:2, ⟨fbf⟩:2, ⟨(bf)b⟩:2, ⟨(bf)f⟩:2, ⟨(bf)bf⟩:2, ⟨(bd)b⟩:2, ⟨bba⟩:2, ⟨aba⟩:2, ⟨abb⟩:2, ⟨bcb⟩:3, ⟨bbc⟩:2}. There might be need to construct a new frequent item matrix for projected databases whose annotation contains more than 3 items and recursively mine its sequential pattern.

This approach, with the help of projected database, helps reduce the search space for sequential patterns and make the mining process faster. There is however the need to recursively reconstruct frequent item matrix when there are more patterns to be mined.

2.2.3 PrefixSpan

A sequence database is one that records events in its database in the order in which they occur. As a result, it takes into account the time at which events occur. For example, the sequences (abc) is different from (bac). Though both sequences contain the same item/events, event ‘a’ comes before event ‘b’ in the first sequence and vice versa in the second sequence. Therefore, in sequence databases, the two sequences are regarded as two different sequences. Sequence of transactions can also be represented in the order in which they occur and at the same time indicate events of the same transaction. For example, the sequence ⟨a(abc)(ac)d(cf)⟩ contains (a), (abc), (ac), (d) and (cf) transactions (elements) that occur separately in the order shown.

The Prefix-Projected Sequential Pattern Mining (PrefixSpan) by Pei et al. (2004) is designed to solve the problem of mining sequential pattern. The approach uses a divide and conquer method by projecting the sequence database over common frequent prefixes and recursively mining each projected database. A prefix of a sequence is the part of a sequence that strictly precedes it. For example, sequences ⟨a⟩, ⟨a(ab)⟩, ⟨a(abc)⟩ are prefixes of sequence ⟨a(abc)(ac)d(cf)⟩. On the other hand, sequences ⟨ab⟩ and ⟨a(bc)⟩ are not.

The algorithm works by first making a scan on the database to find the frequent 1-sequential patterns. The database is then divided into n projected databases, defined by sequences starting with each of the n 1-sequaential database. Each of these projected databases are then recursively mined by further projection to determine the frequent patterns they contain. Given the database as shown in Table 24 and minimum support of 2, the database is scanned to generate the frequent 1-sequentila pattern: a:4, b:4, c:4, d:3, e:3 and f:3

Sequence_id	Sequence
10	$\langle a(abc)(ac)d(cf)\rangle$
20	$\langle (ad)c(bc)(ae)\rangle$
30	$\langle (ef)(ab)(df)cb\rangle$
40	$\langle eg(af)cbc\rangle$

Table 24: The sequence database

The sequence database is then divided into 6 parts (6-projected databases) based on sequences preceded by each of the frequent 1-sequential patterns. The second column of Table 25 depicts this. A-projected database is found for example, by scanning the database to find all sequences prefixed by ‘a’. The first 2 sequences with sequence_ids 10 and 20 are preceded by ‘a’. The sequences ⟨a (abc) (ac) d (cf)⟩ and ⟨(ad) c (bc) (ac)⟩ minus ‘a’ forms their respective projected sequences ⟨(abc) (ac) d (cf)⟩ and ⟨(-d) c (bc) (ac)⟩. The sequence ⟨(ef) (ab) (df) (cb)⟩ is preceded by ‘a’ at element (ab). Element (ef) is removed and the a-prefix sequence formed is ⟨(-b (df) (cb)⟩. The same method is used to complete column 2 of Table 25.

Each projected database is then recursively mined by projection to discover any possible frequent pattern. The a-projected database is for example, mined to discover if another ‘a’ is frequent. The a-projected database shows that a is frequent (a:2) thereby making ⟨aa⟩ a frequent pattern as shown in column 3 of Table Table 25. The resulting aa-projected database found from a-projected database is {⟨(-bc)(ac)(d)(cf)⟩, ⟨(-e)⟩}. No further frequent item can be found from this, therefore no frequent pattern prefixed by ‘aa’ can be found. The algorithm, in the same manner, mines for ‘b’ from a-projected database to check if ⟨ab⟩ is frequent. Item ‘b’ is found 4 times, making ⟨ab⟩ frequent. The resulting ab-projected database is ⟨(-c) (ac) (d) (cf)⟩, ⟨(-c)(ae)⟩, ⟨(c)⟩, Recursively mining the ab-projected database, using same method, gives ⟨aba⟩, ⟨abc⟩, ⟨a(bc)⟩ and ⟨a(bc)a⟩.

The same process is used for all the projected databases to find the respective frequent patterns shown in Table 25. The union of these frequent patterns gives set of frequent patterns for the database.

prefix	projected (suffix) database	sequential patterns
$\langle a\rangle$	$\langle(abc)(ac)d(cf)\rangle$, $\langle(\_d)c(bc)(ae)\rangle$, $\langle(\_b)(df)cb\rangle$, $\langle(\_f)cbc\rangle$	$\langle a\rangle$, $\langle aa\rangle$, $\langle ab\rangle$, $\langle a(bc)\rangle$, $\langle a(bc)a\rangle$, $\langle aba\rangle$, $\langle abc\rangle$, $\langle(ab)\rangle$, $\langle(ab)c\rangle$, $\langle(ab)d\rangle$, $\langle(ab)f\rangle$, $\langle(ab)dc\rangle$, $\langle ac\rangle$, $\langle aca\rangle$, $\langle acb\rangle$, $\langle acc\rangle$, $\langle ad\rangle$, $\langle adc\rangle$, $\langle af\rangle$
$\langle b\rangle$	$\langle(\_c)(ac)d(cf)\rangle$, $\langle(\_c)(ae)\rangle$, $\langle(df)cb\rangle$, $\langle c\rangle$	$\langle b\rangle$, $\langle ba\rangle$, $\langle bc\rangle$, $\langle(bc)\rangle$, $\langle(bc)a\rangle$, $\langle bd\rangle$, $\langle bdc\rangle$, $\langle bf\rangle$
$\langle c\rangle$	$\langle(ac)d(cf)\rangle$, $\langle(bc)(ae)\rangle$, $\langle b\rangle$, $\langle bc\rangle$	$\langle c\rangle$, $\langle ca\rangle$, $\langle cb\rangle$, $\langle cc\rangle$
$\langle d\rangle$	$\langle(cf)\rangle$, $\langle c(bc)(ae)\rangle$, $\langle(\_f)cb\rangle$	$\langle d\rangle$, $\langle db\rangle$, $\langle dc\rangle$, $\langle dcb\rangle$
$\langle e\rangle$	$\langle(\_f)(ab)(df)cb\rangle$, $\langle(af)cbc\rangle$	$\langle e\rangle$, $\langle ea\rangle$, $\langle eab\rangle$, $\langle eac\rangle$, $\langle eacb\rangle$, $\langle eb\rangle$, $\langle ebc\rangle$, $\langle ec\rangle$, $\langle ecb\rangle$, $\langle ef\rangle$, $\langle efb\rangle$, $\langle efc\rangle$, $\langle efcb\rangle$.
$\langle f\rangle$	$\langle(ab)(df)cb\rangle$, $\langle cbc\rangle$	$\langle f\rangle$, $\langle fb\rangle$, $\langle fbc\rangle$, $\langle fc\rangle$, $\langle fcb\rangle$

Table 25: The Projected database and frequent patterns

The technique has removed the need to generate candidate sequences during mining. It also results in shrinking of projected databases as the mining process progresses. The limitation of this technique is the need to construct and store the projected databases. Pei et al. (2004) proposed pseodoprojection method to solve this problem. This is done by replacing the physically constructed projected databases with their various sequence identifiers and offsets where their suffixes begin. This makes the projected fit into memory.

2.2.4 The WAP mine algorithm

The WAP mine algorithm of Pei et al. (2000) is based on the construction of Web Access Pattern (WAP) Tree constructed from the web log sequences after non-frequent events have been removed.

The algorithm can be described in 3 stages. The web access sequence database (WASD) is first scanned to determine the frequent 1-sequence. It is then scanned the second time in order to remove non-frequent event from the sequence. These sequences are then used to build the WAP tree. The tree is built by representing events with nodes showing labels and counts of the node along that path. Each event has a queue linking all its nodes in the order in which they are inserted. The head of each queue is registered in a header table for the WAP tree. Table 26 and Figure 2 represent these 2 stages

TID	Web access sequence	Frequent subsequence
100	*abdac*	*abac*
200	*eaebcac*	*abcac*
300	*babfaec*	*babac*
400	*afbacfc*	*abacc*

Table 26: The Web Access Sequence Database

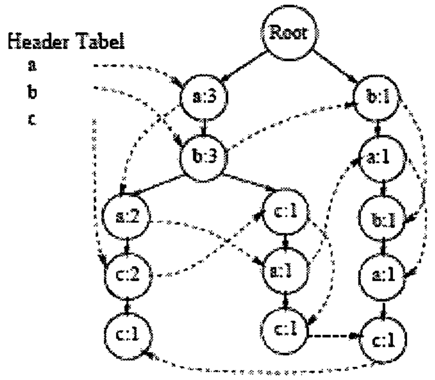


Figure 2: The Web Access Pattern tree

The third stage then recursively mines the WAP tree using conditional search. The search is based on looking for all sequences with common suffix. As the suffix becomes longer, the remaining search space becomes shorter. The header table created with the WAP tree above helps construct conditional sequence base for each of the events considered as suffixes by following the link to all nodes in the WAP tree. This is then used to construct conditional WAP tree for every suffix considered. The conditional WAP tree is then recursively mined for frequent events, each time concatenating newly discovered frequent event with the old ones. The process continues until there are no trees to mine or no more frequent events discovered.

Using the above WAP tree as an example, with 75% minimum support, the conditional WAP tree for suffix sequence 'c' is as shown in Figure 3. This tree is constructed from all prefix sequences of suffix sequence 'c'. The possible prefix sequences, called conditional pattern base of suffix sequence 'c' are: aba:2, ab:1, abca:1, ab:-1, baba:1, abac:1, aba:-1. The negative sequences indicate an overlap in the count because they are already contained in some other prefixes of 'c'. Because the count of 'c' is less than 75%, it is removed from the sequence. The resulting sequences are aba:1, aba:1, baba:1, aba:1 from which Figure 3 is constructed. The same process can be used to construct conditional WAP tree for other suffix sequences, for example 'ac' from Figure 3. The resulting tree is as shown on Figure 4.

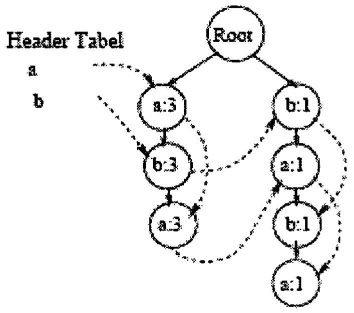


Figure 3: |c

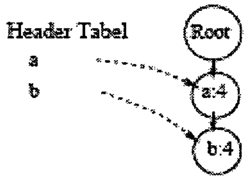


Figure 4: |ac

Figures 3 and 4 show the conditional WAP tree for suffix sequence 'c' and 'ac' respectively.

The conditional WAP tree 'ac' now confirms that frequent sequences with 'c' as suffix are ac, aac, bac, abac. The same process can be done with the WAP tree in Figure 2 with suffixes 'b' and 'a'.

The limitation of this technique is the recursive construction of intermediate WAP tree for suffixes under consideration.

2.2.5 PLWAP algorithm

In order to eliminate the need to recursively construct intermediate trees during mining, PLWAP algorithm was proposed. The approach of Ezeife and Lu (2005) uses position codes generated for each node such that antecedent/descendant relationships between nodes can be discovered from the position code. The same WAP tree originally created is then mined prefix-wise using the position codes as identifiers thereby eliminating generation of fresh intermediate WAP trees for each sequence mined. The concept of binary tree is used in generating the position codes. The root has a position code of null. Starting from the root, all tree nodes are assigned a position code using the following rule. The position code of the leftmost child node is the position code of its parent concatenated with '1' at the end; the position code of any other node is the same as appending '0' to the position code of its closest left sibling.

The algorithm first scans the web access sequence database (WASD) to obtain support count for all events in it. Events with support greater than or equal to a specified threshold are said to be frequent. A second scan is then used to eliminate non-frequent events from the original sequences. These new sequences are then used to construct a PLWAP-tree with each node representing label, count and position code of the event along a particular path. The PLWAP-tree is then traversed in a pre-ordered fashion starting from the root to the left sub-tree followed by the right sub-tree to build to build the header node linkages. Each event has a queue linking all its nodes in the order in which they are inserted. The head of each queue is registered in a header table for the PLWAP tree.

The PLWAP algorithm then recursively mines the PLWAP tree using prefix conditional sequence search. Given the WASD as shown in Table 27:

TID	Web access sequence	Frequent subsequence
100	*abdac*	*abac*
200	*eaebcac*	*abcac*
300	*babfaec*	*babac*
400	*afbacfc*	*abacc*

Table 27: The Web Access Database

With the minimum support threshold of 75%, the sequence is reduced to that shown on the frequent sub sequences column shown above since events 'e' and 'f ' are both 50% frequent. Events 'e' and 'f ' are not frequent, they are therefore removed in the frequent subsequences. The resulting reduced sequences are then used to build the PLWAP tree as shown in Figure 5.

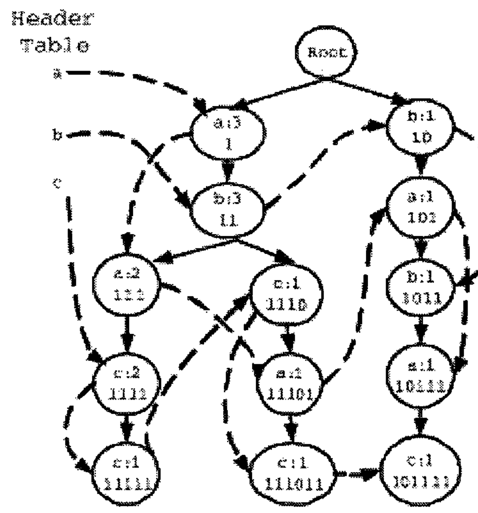


Figure 5: The PLWAP tree with the header linkages.

The mining starts by mining sequences with the same prefix. Starting from frequent 1-sequence, the PLWAP algorithm mines starting from the root (for the first time). Using the header linkage, it traverses the tree to identify frequent 1-sequences by searching for the first occurrence of an event say 'a'. The position code then helps in preventing duplicate count of support of an event in the same suffix sub tree. The addition of these counts is then used to compare with the specified minimum support threshold.

In the case of the above example, the event ‘a’ is counted with nodes a:3:1 and a:1:101. The total count for ‘a’ is 4, making it a frequent 1-sequence. In order to find the next frequent sequence with ‘a’ as prefix, the PLWAP tree is rooted at b:3:11 and b:1:1011. The first occurrence of say ‘a’ in these sub-trees are then noted with the counts. For ‘a’ event, the following nodes are identified using b:3:11 and b:1:1011 as roots: a:2:111, a:1:11101 and a:1:10111. Again ‘a’ occurs 4 times, making it frequent. Therefore, ‘aa’ is frequent sequence. The next 3-sequence with ‘aa’ as its prefix can be found by setting the root of the PLWAP tree to c:2:1111, c:1:111011 and c1:101111. There are no other events except ‘c’ with counts of c:2:1111, c:1:111011 and c1:101111, making it a total of 3. This makes ‘c’ frequent. Therefore, ‘aac’ is frequent. Shifting the root to c:1:11111 gives no other frequent event. The algorithm then backtracks to find other possible frequent sequence combinations at each level. The same procedure is followed and the following frequent sequences were found: (a,aa,aac, ac, ab, aba, abac, abc,b, ba, bac, bc, c).

The advantage associated with this algorithm includes efficiency in terms of I/O and memory utilisation since it eliminates the need to store intermediate WAP tree. The need to continuously store and reconstruct intermediate WAP trees has been eliminated by the introduction of position codes, making tree traversal a lot easier. The use of pre-order linking of header nodes of the same suffix tree also makes searching of nodes easier.

2.3 Uncertainty approaches for mining patterns

Recent research has been done on representation and mining of data that exist with uncertainty. Uncertain data sequences are series of data that exist with doubt. The level of doubt is usually expressed with probability values attached to the elements of the sequence. These probability values are referred to as existential probability. This section gives an insight into how data with uncertainty are represented and the progress made in mining such data.

2.3.1 Trio system

Agrawal et al. (2006) introduce the Trio system as a system aimed at representing data, uncertainty of the data and data lineage. The uncertainty shows the level of doubt attached to the data. Lineage of a tuple is defined as the information showing how the tuple is derived. It shows the origin of the tuple. The authors demonstrate how Uncertainty and Lineage Databases (ULDBs), a new scheme earlier proposed by the authors, can be represented using the 'crime solver' application. The application is designed to express doubts of observers on driver information and crime-vehicle sightings. They also show TriQL, a query language of this scheme, that can operate on ULDBs.

The authors represent uncertain data with x-tuple notations indicating the possible combinations of database instances. An x-tuple is a tuple in a database that has alternative values. The values of x-tuple are not 100% defined. There is also the need to represent a 'may be' (denoted with '?') occurrence of an x-tuple indicating the x-tuple may not exist as shown in Table 28. Confidence (probability) values are attached to each x-tuple representing the likelihood of its existence. The sum of these values is less or equal to 1 depending on a 'may be' presence on each x-tuple as shown below.

(witness, car)	
(Amy,Honda):0.7 || (Amy,Toyota):0.3	
(Betty,Acura):0.6	?

Table 28: The representation of uncertain data

It is assumed that the likelihood that Amy saw a Honda car is 0.7 and that of Toyota is 0.3. Therefore, (Amy, Honda):07 || (Amy, Toyota):03 is an x-tuple. The events are mutually exclusive, indicating only one must exist at a time. The second tuple indicates a 0.6 confidence that Betty saw Acura. Though not an x-tuple, the '?' sign indicates the tuple may not exist. As a result, there is a 0.4 confidence that Betty did not see an Acura. From these uncertainties, it is possible to have 4 possible instances, referred to as the 'possible worlds', of this relation as follows:

(Witness, Car)
Amy, Honda
Betty, Acura

(Witness, Car)
Amy, Toyota
Betty, Acura

(Witness, Car)
Amy, Honda

(Witness, Car)
Amy, Toyota

Example 2.3.1 Demonstration of possible instances in uncertain databases

The authors have been able to propose a method for representing uncertain data. The representation is found to be useful in preparing data sequence for mining as confidence values can equally be attached to possible sequences in web log mining.

2.3.2 Databases with uncertainty and Lineage

Benjelloun et al. (2006) further gives a formal representation of databases with uncertainty and lineage. Lineage in database shows the origin of a particular tuple when a new relation is formed. They found that completeness of uncertainty in databases is possible as a result of lineage attached to relations when new relations are formed. Much of the work is on querying of Uncertainty and Lineage Databases (ULDBs). The work of Widom (2005) that is centered on incorporating data lineage and uncertainty in general purpose database management system provides a motivation for this work.

The technique presented here is highly related to that of Agrawal et al. (2006) except that some level of formalism is introduced and more emphasis is laid on lineage representation as a means of ensuring completeness of uncertainty representation. Since lineage gives information about the origin and how a particular tuple is derived, such information is found to be useful in understanding the uncertainty attached to the tuple. The authors claim that formalism for representing an uncertain database is said to be complete if it can represent any finite set of possible instances.

Given the example below:

ID	Saw(witness, car)	
21	(Amy, Mazda) || (Amy, Toyota)	?
23	(Betty, Honda)	

ID	Drives(person, car)
31	(Jimmy, Mazda)
32	(Jimmy, Toyota)
33	(Billy, Mazda)
34	(Billy, Honda)

ID	Accuses(witness, person)		
41	(Amy, Jimmy)	?	$\lambda(41,1)=\{(21,1),(31,1)\}$
42	(Amy, Jimmy)	?	$\lambda(42,1)=\{(21,2),(32,1)\}$
43	(Amy, Billy)	?	$\lambda(43,1)=\{(21,1),(33,1)\}$
44	(Betty, Billy)	?	$\lambda(44,1)=\{(23,1),(34,1)\}$

Example 2.3.2 Explanation of databases with uncertainty and lineage

Joining the tables SAW (witness, car) and DRIVES (person, car) gives the table Accuses (witness, person). Adding the lineage function λ helps identify the origin of each tuple in Accuses table. It also helps determine which tuple may exist as a result of the existence of its parent tuple. The function $\lambda(41,1) = [(21,1) , (31,1)]$ shows that the first and the only possible value of tuple with ID 41 in Accuses is derived from the first possible values of IDs 21 and 31 from tables Saw and Drives respectively. In the same manner, function $\lambda(42,1) = [(21,2) , (32,1)]$ shows that the first and the only possible value of tuple with ID 42 in Accuses is derived from the second possible value of ID 21 and the first possible value of ID 32 from tables SAW and DRIVES respectively.

A ULDB Database D is therefore represented as a triple (R,S,λ) where R is a set of x-relations, S is a set of symbols in instances of R and λ is a lineage function showing the source of a newly generated relation.

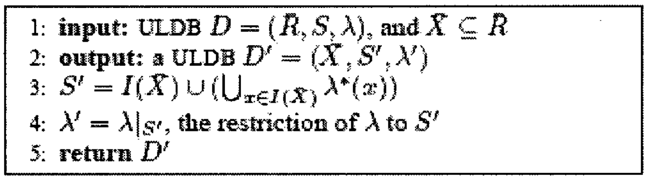
The following definition then shows the importance of lineage in ensuring completeness in ULDBs. Given that D_k is an instance of D, S(i,j) represents the ith tuple and jth alternative and $S_k \supseteq S$, then

1. If $S_{(i,j)} \in S_k$, then every $j' \neq j$, $S_{(i,j')} \notin S_k$
2. $\forall\, S_{(i,j)} \in S_k$, $\lambda(S_{(i,j)}) \subseteq S_k$

3. If for some x-tuple t_i , there does not exist a $S_{(i,j)} \in S_k$, then t_i is a may be x-tuple, and $\forall\, S_{(i,j)} \in t_i, \lambda(S_{(i,j)}) = \phi$ or $S_k \subset \lambda(S_{(i,j)})$

Condition 1 formalises the mutual exclusiveness of instances on uncertain databases. Condition 2 enforces the semantics of lineage. This implies that if an alternative is present in a possible instance, the parent tuple from which it was derived must also have alternative(s). The last condition indicates that an x-tuple, which is the tuple that does not have 100% existence (Agrawal et al., 2006), might not be present at all in an instance if it is a maybe tuple. Its lineage is null or it does not belong to the class of all possible symbols of lineages recognised.

The authors then provide the following algorithm that extracts query results from the join operation on database tables with uncertain data by enforcing the three conditions listed above. This ensures no impossible instance is returned:



```
1: input: ULDB $D = (R, S, \lambda)$, and $X \subseteq R$
2: output: a ULDB $D' = (X, S', \lambda')$
3: $S' = I(X) \cup (\bigcup_{x \in I(X)} \lambda^*(x))$
4: $\lambda' = \lambda|_{S'}$, the restriction of $\lambda$ to $S'$
5: return $D'$
```

Statements 1 and 2 give the input and output representations of ULDB when queries are issued and results returned. The ULDB is represented by a triple (R,S,λ) with R representing the relations involved, S the set of tuple values and λ represents the lineage of each tuple derived. Statement 3 shows that new set of symbols in the new relations to be returned are generated from symbols of instances of the original relations and that of their lineages if they exist. If the parent relations are not generated from a join operation, the parent relation will not have lineage. Statement 4 ensures that conditions such as mutual exclusion and certainty of existence stipulated from the parent relation λ are maintained in λ′ and by extension S′. The new ULDB D′ is then returned.

Using the example 2.3.2, the set of relations in R are tables SAW and DRIVES. The lineages of tables SAW and DRIVES are empty since they are not derived. Accuses is the new relation. The set of symbols in S′ include Amy Betty, Billy, Jimmy and the tuple IDs shown in the lineage λ′. The lineage $\lambda'(i.j) = \{(i', j'), (i'', j'')$ is defined where i represents the tuple ID of the new relation, and j denotes the instance. The tuple IDs i', i'' and instances j', j'' denote the instances from which new relations are formed in the parent relations. $\lambda'(41,1) = \{(21,1),(31,1)\}$ for example shows that tuple ID 41 of instance 1 in the new relation Accuses is generated from the tuple IDs 21 and 31 in SAW and DRIVES respectively. The first instance values are used in both Saw and Drive. The lineages λ' are:

$\lambda'(41,1) = \{(21,1),(31,1)\}$

$\lambda'(42,1) = \{(21,2),(32,1)\}$

$\lambda'(43,1) = \{(21,1),(33,1)\}$

$\lambda'(44,1) = \{(23,1),(34,1)\}$

From this representation, the restriction of statement 4 can be maintained by ensuring tuples with similar ID values but different instance values do not exist together in an instance of the returned relation. For example $\lambda'(41,1) = \{(21,1),(31,1)\}$ and $\lambda'(42,1) = \{(21,2),(32,1)\}$ must not exist in the same instance since they both have ID 21 but instances 1 and 2 respectively. In the same vein, tuples $\lambda'(41,1) = \{(21,1),(31,1)\}$ and $\lambda'(43,1) = \{(21,1),(33,1)\}$ must exist together in same instance since they both have ID 20 and instance 1 from the parent relation.

Benjelloun et al. (2006) therefore introduced the formal presentation of databases with uncertainty and lineage. They show the completeness of instances of uncertain database as a result of inclusion of lineage parameters. This is found to be useful in deriving relations from existing relations without returning impossible instances of x-tuples in queries.

2.3.3 Working Model for uncertain data

Sarma et al. (2006) explain the usefulness and importance of representing uncertain data with simple models that might be incomplete in general terms but is sufficient enough in handling the domain of the application under consideration. The authors argue the need to have an application depending on uncertain data modelled with the simplest possible model in order to ensure 'user friendliness' and simplicity without losing 'completeness' within the domain of the application. The authors propose a 2-layer approach that satisfies general completeness at the background and equally support simpler modelling at the surface. Several possible simple models are proposed and the relationship in terms of hierarchy is shown in Figure 6.

The technique used by the authors is based on the fact that a-tuple (same As x-tuple) explained by Agrawal et al. (2006) which uses mutual exclusion. Sarma et al. (2006) represents this with R^A indicating several possible instances of a tuple in a relation without any constraint linking it with the existence of other tuples. This is however found to be incomplete on general case scenario. An example of this is a probability data base in which a tuple A can only exist if another tuple B already existed as shown in the following example:

```
I1: empty
I2: [Carol, 12/25/04, Los Altos, bluebird]
I3: [Carol, 12/25/04, Los Altos, bluebird],
    [Carol, 12/26/04, Los Altos, bluebird]
```

Example 2.3.3 Tuple dependency constraint

The above example shows three possible instances of a database in which it might be empty, contain record [Carol, 12/25/04, Los Altos, bluebird] only or compulsorily contain both [Carol, 12/25/04, Los Altos, bluebird] and [Carol, 12/26/04, Los Altos, bluebird]. In instance 3 (I3), the existence of [Carol, 12/26/04, Los Altos, bluebird] is dependent on existence of [Carol, 12/25/04, Los Altos, bluebird].

The authors therefore make use of complete modelling based on $R^A{}_{prop}$ defined as:

- A multiset of a-tuple, T = t1...tn, and
- A Boolean formula f(T)

For the above example, its model can be defined as:

```
t1 = [Carol, 12/25/04, Los Altos, bluebird]
t2 = [Carol, 12/26/04, Los Altos, bluebird]
constraint:  t2 => t1
```

This means that modelling this type of database will demand using constraint t2 which is dependent on t1 in addition to the possibility of having multiset of a-tuples (x-tuples).

That is, a tuple might have many possible values. The constraint implies that if tuple [Carol, 12/26/04, Los Altos, bluebird] exists, tuple [Carol, 12/25/04, Los Altos, bluebird] must also be present in the database.

Variants of this model are then defined as shown below with their corresponding hierarchies. These variants are not complete in all scenario but they can be sufficient to model a particular situation. The models are also closed under the operations shown.

Model	$\mathcal{R}^A$	$\mathcal{R}_?$	$\mathcal{R}^A_?$	$\mathcal{R}_{\oplus\equiv}$	$\mathcal{R}_2$	$\mathcal{R}^A_2$	$\mathcal{R}_{sets}$
Building Block	a-tuple	tuple	a-tuple	tuple	tuple	a-tuple	tuple
Constraints	none	?	?	binary $\oplus$, $\equiv$	2-clause	2-clause	n-way choice

Table 29: Models and their constraints

Model R^A is a model that allows for a-tuples (x-tuple) with no constraint. That is, no additional condition is allowed to be specified on inter tuple existence. Model $R_?$ allows only precise tuples but such tuples might not exist (maybe tuple). Model $R^A_?$ allows both a-tuple and maybe constraint specification. Model $R_{\oplus\equiv}$ allows precise tuples with conditions of only binary exclusive OR and equivalence operations. Model R_2 allows precise tuples with 2 conditions specified. Model R^A_2 allows for a-tuples and 2 conditions specified. Model R_{sets} allows precise tuples with n possible conditions. The expressive power models in Figure 6 shows that model R^A can be used to express models $R^A_?$, R_{sets} and R^A_2 . Model $R^A_?$ can be used to express models R_{sets} and R^A_2 . Model $R_?$ can be used to express all models except R^A .

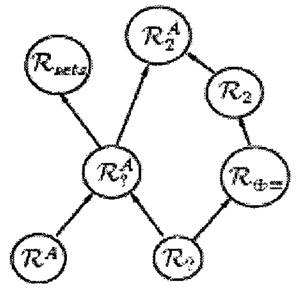


Figure 6: Hierarchies of the models

The various operations that can be performed on the models expressing their closure property is as given below

Closure-Model	$\mathcal{R}^A$	$\mathcal{R}_?$	$\mathcal{R}_?^A$	$\mathcal{R}_{\oplus\equiv}, \mathcal{R}_2, \mathcal{R}_2^A, \mathcal{R}_{sets}$
Union	Y	Y	Y	Y
Select(ee)	Y	Y	Y	Y
Select(es)	N	Y	Y	Y
Select(ss)	N	Y	N	Y
Intersection	N	Y	N	N
Cross Product	Y	N	N	N
Join	N	N	N	N
Difference	N	Y	N	N
Projection	Y	Y	Y	Y
Duplicate Elimination	N	Y	N	N
Aggregation	N	N	N	N

Table 30: Closure property of the incomplete model

ee- both operands are exact values

es- one operand is an exact value, the other or set

ss – Both or-set

Sarma et al. (2006) therefore state that the 2-layer model is far more useful in terms of its simplicity and user friendliness. They also proposed that complete model must be present at the background which can be referred to when needed. The authors therefore propose a 2-layer model for modelling uncertain data in which the underlying layer is complete ($R^A{}_{prop}$) and the top layer is simpler, closed under the operation intended and more user friendly (variant of $R^A{}_{prop}$). For example, $R^A{}_2$ can be used to model a union operation Select (ee) since it is closed under the operation. Model R^A can then be used at the background for greater accuracy.

2.3.4 U-apriori and local trimming, global pruning and simple-pass patch up strategy (LGS) algorithm

Chui et al. (2007) proposed U-apriori and LGS algorithms to solve the problem associated with mining frequent patterns in uncertain sequences. The authors stress that existing works have only been able to mine frequent patterns in doubtless facts. The paper identifies situations in which data exist by chance, expressing their likelihood (uncertainty) by probability values. The problem then lies in being able to mine frequent patterns in sequences that exist with uncertainty. Areas in which such scenario can be applied include medical records where some symptoms are subjective and can be best represented by probability values. The authors also point out the use of their approach in pattern recognition applications where satellite pictures are used in determining presence of a target. However, certainty of the images can be affected by noise and resolution.

The authors found that due to the probabilistic nature of items in the dataset, the traditional definition support count is changed to expected support. Strongly linked with this is the notion of "possible world", representing different possible states of the database due to the probabilistic nature of the data it contains. The authors found that expected support count can be calculated by summing the product of probability values of all items in the sequence under consideration, for all tuples in the database D given as:

$$\text{Expected support (X)} = \sum_{j=1}^{|D|} \prod_{x \in X} P_{t_j}(x). \qquad \text{.......... Equation (I)}$$

Where:

|D| is the size of the uncertain database D

t_j is the jth tuple in the database

x is an item in the sequence X in the database.

It is assumed that probability values of the items x contained in sequence X are independent. Therefore, the product of all probabilities of items in X gives the existence of X. This is then done for all X in the database, the sum of which gives the expected support of X.

Given the databases below:

Cust-Id	Sequences
10	abcd
20	acbd
30	abdc

Table 31: Sequence without probability

Cust-Id	Sequences
10	a:1, b:0.5, c:1, d:0.7
20	a:1, c:0.2, b: 0.1, d:1
30	a:0.5, b:1, d:1, c:1

Table 32: Sequence with uncertainty

The support count of sequence ‘ac’ in Table 31 is 3 whereas the expected support count of ‘ac’ in Table 32 will be calculated as (1x1) + (1x0.2) + (0.5x1) = 1.7, as defined in equation (I).
It can be seen from above that several calculations might be involved in arriving at expected support counts especially when sequence under consideration is long. These long calculations might also be wasteful when probability values of items are very low and the minimum support count required is high. The authors therefore suggested that simply modifying apriori algorithm to account for the expected support count (U-apriori) might not be efficient. The authors proposed local trimming, global pruning and simple-pass patch up strategy (LGS).

The Local trimming phase introduces trimming threshold value that is used to remove items that are unlikely to have a significant effect on the expected support count. This trimming threshold is calculated by sorting all probability values for each item in ascending order and performing a cumulative sum for each item until it is equal to the minimum support threshold. The probability value at this point is chosen as the trimming threshold. The probability values for the trimmed items are also kept for error estimation which is used at the pruning stage.

The pruning is based on discovering sequences that are potentially frequent. A sequence X in the trimmed database is potentially frequent if the sum of its expected support count (S^T_e (X)) and the upper bound of the error estimated for its expected support (e (X)) is greater than the minimum support. That is sequence X is potentially frequent if:
$S^T_e (X) + e (X) \geq$ minimum support value.

The set of frequent and potentially frequent sequences generated during pruning are verified using the simple-pass patch up strategy. During the simple patch up strategy, the original database (untrimmed database) is scanned once to determine the frequent sequences. The result is then used to authenticate those found in potentially frequent sequences from the trimmed database.

2.3.5 UF-growth algorithm

Leung at al. (2008) proposed a tree-based algorithm in order to discover frequent patterns in situations where data items exist with probability. The authors state that such situations are found in patients' records where physicians may suspect but not guarantee certain ailment until further tests are carried out.

The authors' contributions include the proposal of a tree structure, UF-tree for capturing the uncertain data. They also proposed UF-growth algorithm for mining frequent patterns from the UF-tree. The authors then proposed reduction of probability values of items to 2 decimal places to reduce the possible tree paths. They also improve the algorithm by removing the need to build conditional tree for sub-tree since all subsets of an extracted tree path can be enumerated and their expected supports summed to find the frequent ones.

Based on the definition of expected support count in Chui et al. (2007), items existential probabilities are used to calculate the expected support counts. This approach is however better in that it eliminates the candidate sequence generation associated with U-apriori by Chui et al. (2007).

The tree UF-tree is constructed in a way similar to FP-tree by Han et al. (2000) except that nodes also store the probability value of their items in addition to the label and their number of occurrences. As a result, similar items may be represented with different nodes when they share similar prefixes if they have different probability values. The proposed solution first scans the uncertain database to calculate expected support count of items. The frequent items greater than the minimum support are then ordered by the value of their support counts. This order is then used to scan the database the second time to construct the UF-tree after the items have been arranged in this order and the infrequent ones removed.

Given the following database:

Transactions	Web logs	Frequent sub-sequences
t_1	(a:0.9, c:0.81, d: 71875, e:0.72)	(a:0.9, c:0.81, d: 71875, e:0.72)
t_2	(a:0.9, d:0.72, e:0.71875, f:0.8)	(a:0.9, d:0.72, e:0.71875)
t_3	(b:0.875, c:0.85714)	(b:0.875, c:0.85714)
t_4	(a:0.9, d:0.72, e:0.71875)	(a:0.9, d:0.72, e:0.71875)
t_5	(b:0.875, c:0.85714, d:0.05)	(b:0.875, c:0.85714, d:0.05)
t_6	(b:0.875, f:0.85714)	(b:0.875)

Table 33: The uncertain database

Each tuple in the uncertain database contains items with their associated existential probabilities. For example, tuple t_1 in Table 33 contains items 'a', 'c', 'd', and 'e with existential probabilities 0.9, 0.81, 0.71875 and 0.72 respectively. The existential probability of each item in the database is then scanned in each tuple. The sum of the existential probabilities for each item represents the expected support count of the item. The support count of item 'a' for example, is found by adding 0.9 + 0.9 + 0.9 = 2.7 as found in t_1, t_2 and t_4. Using the minimum support of 1, items a, b, c, d and e are found to be frequent with expected support count of a = 2.7, b = 2.625, c = 2.52429, d = 2.20875 and e = 2.1575. Item f is infrequent with support count of 0.9. The transactions are arranged in descending order of their expected support whenever they appear in the database. The order used is a, b, c, d, e since the expected support count of item 'a' is the highest and that of 'e' is the lowest. Item f is removed from the database since it is not frequent. The resulting ordered and frequent sub-sequences are as shown in the last column of Table 33.

The UF-tree is then constructed by scanning the frequent sub-sequences. Starting from the root node, the first frequent sub-sequence (a:0.9, c:0.81, d: 71875, e:0.72) is entered into the tree as the leftmost sub-tree since there no previously existing nodes. The count for each node initialised to 1 and their corresponding existential probability values are recorded. The second frequent sub-sequence (a:0.9, d:0.72, e:0.71875) is read. Starting from the root, since a node with label 'a' already exist with the same existential probability, the count of this node is incremented by 1. Item 'd' is read. A new node is created as no node with label 'd' exists along this path. Its count is set to 1 and the existential probability 0.72 is recorded. New node is also created for item 'e' along this path and the existential probability 0.71875 recorded with the count is also set to 1. The same process is repeated for sequence (b:0.875, c:0.85714) with which a new right sub-tree is created from the root node since no node with label 'b' exists as child to the root node. As a general rule, new nodes are created if no node already exist with the same label and existential probability along a path. If a node with the same label **but different** existential probability exist in a path, a new node must be created. Nodes with the same label are then linked in order to help in building conditional suffix tree during mining. The completed UF-tree is shown in Figure 7.

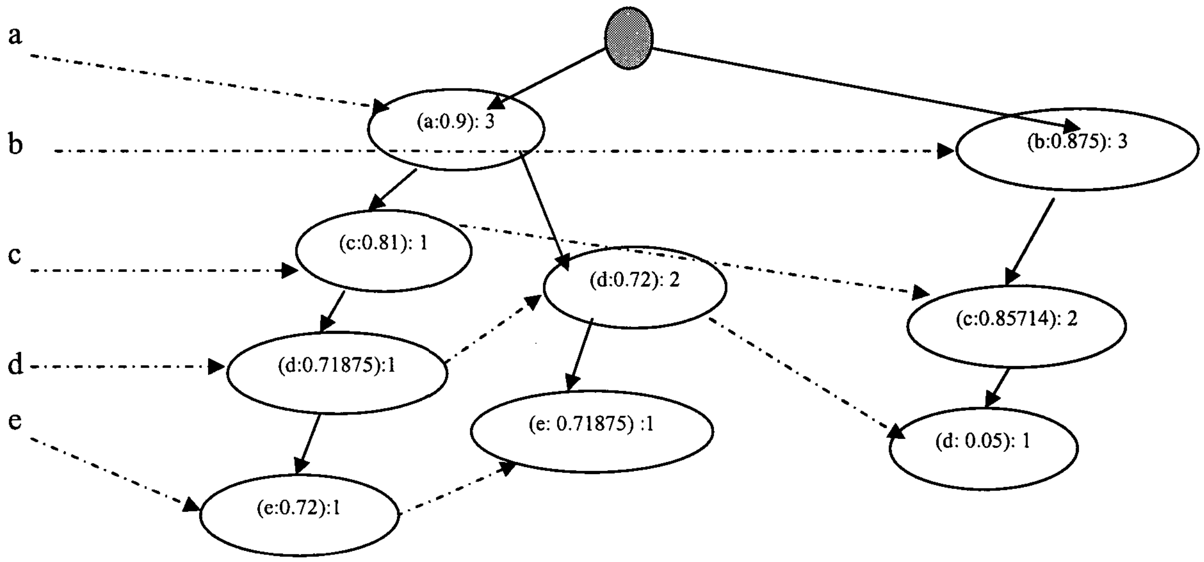


Figure 7: The UF-tree

UF-growth mining also operates like the FP-growth, except that for each extracted tree path, the expected support of a pattern is calculated by the sum of the product of the expected support of its constituent items.

Starting with item e, the conditional suffix tree of item e is built by extracting all prefix pattern of item 'e' without including item 'e'. Each of the existential probability values of the items in the pattern is then multiplied by the existential probability value and count of 'e'. The sequences are {(a:0.9 x 0.72):1, (c:0.81 x 0.72):1, (d:0.71875 x 0.72):1} and {(a:0.9 x 0.71875):2, (d:0.72 x 0.71875):2}. The support count of the items is then based on the sum of products of existential probabilities and count of item 'e' for each item in the pattern. Support count of item 'a' is found to be (0.9 x 0.72 x 1) + (0.9 x 0.71875 x 2) = 1.94175. That of item 'c' = (0.81 x 0.72 x 1) = 0.5832 and item 'd' = (0.71875 x 0.72 x 1) + (0.72 x 0.71875 x 2) = 1.5525. Therefore, 'ae' and 'de' are frequent patterns. Since the support count of item 'c' is less than the minimum support 1, item 'c' is removed and the resulting sequence is used to build the conditional prefix tree of e (|e) as shown in Figure 8.

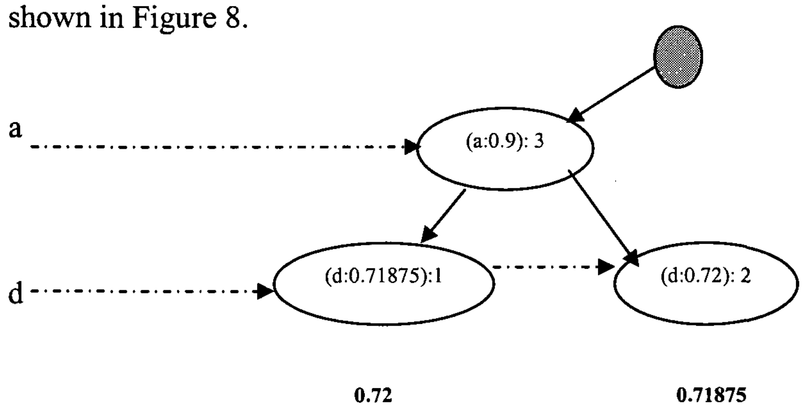


Figure 8: |e

The conditional prefix tree |de can also be built from figure 8 by extracting item 'a' along the paths (a:0.9 x 0.71875 x 0.72 x 1) and (a:0.9 x 0.72 x 0.71875 x 2). The total support count of item 'a' = (0.9 x 0.71875 x 0.72 x 1) + (0.9 x 0.72 x 0.71875 x 2) = 1.39725. Since the 1.39725 is greater than the minimum support 1, pattern 'ade' is also frequent. The |de is shown in Figure 9.

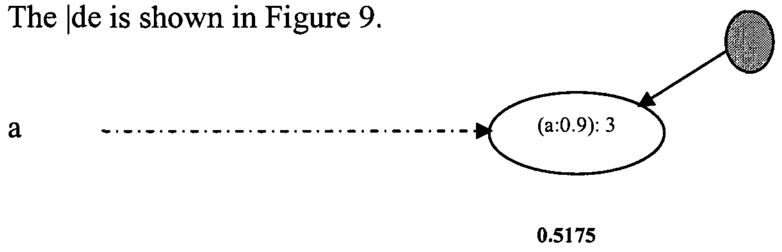


Figure 9: |de

The same procedure is used to build conditional suffix tree for items d, c, b and a. The following patterns are found to be frequent: (a), (a,d), (a,d,e), (a,e), (b), (b,c), (c), (d), (d,e) and (e). The limitation associated with this technique is that it will only cater for uncertainty within the domain of non-sequential pattern. This is as a result of re-ordering of the items in the sequence of the original database. The algorithm only discovers frequent patterns in uncertain data but fail to ascertain the order in which the frequent pattern occurs. For instance in the example above, the algorithm only discovers (a,d,e) as a frequent pattern but can not confirm the order in which the item appear. We can only assume at least one of (ade), (aed), (dae), (dea), (ead) or (eda) is frequent.

3. PROPOSED MINING OF UNCERTAIN WEBLOG SEQUENCES WITH ACCESS HISTORY PROBABILITIES

Recently, there has been the emergence of novel database applications in various non-traditional domains, including World Wide Web, location-based services, sensor networks, RFID systems, and biological and biometric databases. Traditionally, data mining has been widely used to reveal interesting patterns in the vast amounts of data generated by such applications. However, for most of these emerging domains, data are often riddled with uncertainty, arising, for instance, from inherent measurement inaccuracies, sampling errors, and network latencies.

It is in the light of this that Uncertain Position Coded Pre-order Linked Web Access Pattern (U-PLWAP) algorithm is proposed to provide solution in mining data that are associated with uncertainty. The uncertainty can be introduced, for example, in web logs, based on web log histories of different users. Leung et al. (2008) suggest that probability values (existential probabilities) demonstrating uncertainty of items can be deduced from history of the data.

Given the history of web logs of users over a particular time, the web log can be grouped by session and user. The likelihood of the last session being a true reflection of each user can then be expressed in probability values calculated from the history of each user. The result of the mining done with this data can then be useful in rearranging web pages for convenient browsing. Equally, it removes biases and makes the result more accurate since users' web log history is considered and the influence of the history only affect existential probability values of the user only. For instance, given four different web log sessions as shown in Tables 34a, 34b, 34c and 34d, user IDs 10 and 30 participate in all the 4 sessions while user ID 20 only takes part in twice. The most current session is time 4. The four web log sessions are then merged into one database and ordered by user ID and time so that the most current web log for each user comes first as shown in Table 35.

User_ID	Time	Web log
10	1	(a,c,a,c)
20	1	(a,c,b)
30	1	(a,e)

a

User_ID	Time	Web log
10	2	(a,d,c,a)
20	2	(a,d,e,d)
30	2	(a,e,d,e)

b

User_ID	Time	Web log
10	3	(a,d,a,b)
30	3	(a,c,e,b)

c

User_ID	Time	Web log
10	4	(a,b,c,d)
30	4	(a,b,d,b)

d

Table 34: The 4 different web log sessions

User_ID	Time	Web logs
10	4	(a,b,c,d)
	3	(a,d,a,b)
	2	(a,d,c,a)
	1	(a,c,a,c)
20	2	(a,d,e,d)
	1	(a,c,b)
30	4	(a,b,d,b)
	3	(a,c,e,b)
	2	(a,e,d,e)
	1	(a,e)

Table 35: The merged and sorted logs for the different sessions

The four different sessions have been merged and sorted by their user IDs and time. This makes it possible to define the existential probability of the latest log based on the history of the other sessions for each user.

The existential probability prob(u(e)), of each event e, for each user u, in the most current log is given as :

$$\text{prob}(u(e)) = \frac{\text{Number of existence of event e in any record of u}}{\text{Total number of records of u}}$$

The likelihood of having the latest log to be a true reflection of how each user browses the site over a number of sessions is calculated as follows:

User ID 10: Since 'a' appears at least once in all the 4 logs, existential probability of 'a' is given as 4/4 = 1. Item 'b' appears only in the first two logs, its existential probability is 2/4 = 0.5. Item 'c' appears 3 times, its existential probability is 3/4 = 0.75 and that of d is 2/4 = 0.5.

Using the same method, user ID 20's existential probability values for the integrated log is:

'a' = 2/2 = 1

'd' = 1/2 = 0.5

'e' = 1/2 = 0.5

User ID 30:

'a' = 4/4 = 1

'b' = 2/4 = 0.5

'd' = 2/4 = 0.5

The uncertain sequence to be mined is therefore given as in Table 36:

User_ID	Web logs
10	(a:1,b:0.5,c:0.75,d:0.5)
20	(a:1,d:0.5,e:0.5,d:0.5)
30	(a:1,b:0.5,d:0.5,b:0.5)

Table 36: Uncertainty in sequences

For User_ID 10, for example can therefore be said to have visited page 'a' at 100% certainty for every session. Pages 'b', 'c', and 'd' have 50%, 75% and 50% certainty of being visited for every session of User_ID 10.

3.1 Observation on uncertain sequences

Given the databases below:

Cust-Id	Sequences
10	abcd
20	acbd
30	abdc

Table 37: Sequence without probability

Cust-Id	Sequences
10	a:1, b:0.5, c:1, d:0.7
20	a:1, c:0.2, b: 0.1, d:1
30	a:0.5, b:1, d:1, c:1

Table 38: Sequence with uncertainty

The support count of sequence 'ac' in Table 37 is 3 whereas the support count of 'ac' in Table 38 will be calculated as (1x1) + (1x0.2) + (0.5x1) = 1.7

It is noted here that unlike in traditional precise sequences where occurrence count of an item automatically contributes towards the support count of an item, the sum of the product of the existential probability values are used in arriving at support counts.

An equally important observation with uncertain data item is that items with the same label can have different existential probability values. It is therefore important to record item's label, occurrence count and existential probability values in order to accurately determine frequent sequences.

3.2 Preparing data for U-PLWAP algorithm

In order to generate existential probabilities for weblogs as described earlier, weblogs for different sessions need to be collected and cleaned. These weblog sessions are merged into one file after each record is identified by combination of customer ID and the session number/time. The file is then sorted by customer ID and partially by session number (in order to pick the latest session for each customer ID). The existential probability (denoted as prob (u(e))) of each event e, for each user u, in the most current log is given as

$$\text{prob}(u(e)) = \frac{\text{Number of existence of event e in any record of u}}{\text{Total number of records of u}}$$

The computation of existential probabilities is presented formally as shown in Figure 10:

```
Algorithm 1: Existential probability algorithm
Input: Web Access Sequence Databases (WASDs) , Minimum Support
Output: Uncertain Web Access Sequence Database (UDB)
Intermediate: event_sequence, prob_sequence
Begin:
    1. Merge all logs into one file
    2. Sort the file by customer ID and partially by session number (the latest weblog session is at the top for
       each set of unique customer ID)
    3. For each sequence in merged file
           Scan customer ID, latest sequence into event_sequence
           Find prob_sequence by dividing the total number of presence of each item in event_sequence by the
           total number of records with the unique customer ID
           Write into UDB customer ID, event_sequence, prob_sequence
    4. Return UDB
End
```

Figure 10: The Existential Probability algorithm

The length of access history period (LAH) plays a major role in determining the execution time of the existential probability algorithm specified in Figure 10. Longer access period will most likely generate more web logs, provided that the traffic is uniform during the period, thereby increasing the execution time. The length of each session (LOS) also affects the speed of execution of existential probability algorithm. If for example web logs are collected over a year period (LAH = 1 year), they can be divided into sessions of 12 months (LOS = 1 month). This therefore generates a maximum of 12 tuples (1 for each month session) for each user over the 1 year access period. If however sessions are based on a weekly basis (LOS = 1 week), each user has a maximum of 52 tuples. This therefore increases the number of tuples to be sorted in the existential probability algorithm, which in effect increases the execution time of existential algorithm. It is important to note that the length of each tuple in the LOS of 1 week will be shorter than that of 1 month. The U-PLWAP algorithm in Figure 11 therefore takes longer time to run when LOS is 1 month compared to LOS of 1 week as shown in section 4.1.3. Both LAH and LOS are specified by the users of the result of the mining process in a way that best suits their need.

It is also important to note that the existential probability algorithm generates the same value for repeated events/items of the same tuple. This is because the total sample space of each event is restricted to each user and only events in the latest web logs are considered for the existential probability computation. As a result, the problem that may arise due to existence of different existential probability values for repeated events in a tuple is resolved. If however this scenario occurs in another application domain, the highest existential probability value among the various values can be chosen from each tuple. For example if a tuple consists of (a:1, c:0.5, a:0.5, b:0.25, a:0.4), the value 1 is chosen to compute the expected support count of event 'a' in the U-PLWAP algorithm in Figure 11

3.3 Building U-PLWAP algorithm

The motivation of the proposed solution came from the numerous advantages that are associated with PLWAP algorithm discussed in section 2.3.4. However, since PLWAP only mines precise sequences, designing an algorithm based on PLWAP but extended to cater for uncertain sequences will be a welcome idea. The following extensions are made to PLWAP in order to mine uncertain data. The extensions are based on the observations noted in section 3.1.

- The use of the most current web log access history to define each item's existential probability for each user
- Each node representing items in U-PLWAP tree records item's label, occurrence count, position code and set of existential probability values
- Sequences that share the same prefix/suffix but differ in existential probability of constituent items are combined into same node, making the U-PLWAP tree a more compact and faster to mine. UF-growth algorithm, however creates separate nodes for similar items having different existential probabilities.
- The mining process is done on each suffix tree by dynamically generating cumulative product sequence when new temporary nodes are found. This eliminates the need to search for existential probabilities of items found from the root of the tree to the newly found temporary nodes.

3.4 The U-PLWAP algorithm

This newly proposed algorithm works by first scanning the sequence database to discover the frequent 1-sequences. The second scan of the database is used to remove non-frequent items from the sequence database. The resulting sequence is then used to build U-PLWAP tree. The header table is then created for each frequent 1-sequence, using it to link all identical nodes in a pre-ordered version. The U-PLWAP tree is then mined recursively using prefix conditional search with the data items created on the header table. U-PLWAP eliminates the need to search for existential probability from root of the tree to the newly found temporary roots (as would have been done if direct application of PLWAP is to be used) by dynamically generating cumulative products of items found along each suffix tree. This is because there is need to find the products of all existential probability values of items found from the root to the temporary roots of the U-PLWAP tree at any point in time during mining operation. U-PLWAP is also richer than

UF-growth in output since U-PLWAP discovers frequent sequential pattern while UF-growth does not. Comprehensive description of the steps involved is as follows:

Step1: The U-PLWAP scans the sequence database to discover the frequent 1-sequences.This is done by adding up all existential probability values for each item whenever they occur in the database. Whenever an item is repeated in a particular sequence, its existential probability is only added once. The frequent 1-sequences are those items with counts greater than or equal to the minimum support threshold.

Step2: Each of the frequent 1-sequences is used to create entries in the header table.

Step3: A second scan is used to create U-PLWAP tree after the non frequent data items have been removed from the sequence. The U-PLWAP tree is created starting from a null root. Sequences are read from the database and nodes are created for each item in the sequence. Each node contains the item's label, occurrence counts and position code, denoted as label:count:position code. Since similar labels with different existential probabilities in a path are merged into one node, each sequence read is identified by its sequence ID and recorded against the existential probabilities of its items in every node. A left node is created if no node already exist, otherwise, a right node is created and the count initialised to 1. If node already exists, its count is incremented by 1. The position code of each nth leftmost node is determined by appending the binary value of 2^{n-1} to the end of the position code of its parent. The item label and its existential probabilities are read from the sequence database. Entries created in the header table are then used to link their corresponding nodes by traversing the U-PLWAP tree in a pre-ordered fashion (from root to left node first before right node). This makes U-PLWAP tree traversal faster and more efficient since similar nodes in the same suffix tree are brought closer.

Step4: The U-PLWAP tree created is then mined recursively using prefix conditional search. Starting from the root with a particular frequent item 'a' (in order to find all frequent items starting with a) on the header table, all sub-trees are traversed to search for the first occurrence of item 'a'. Its expected support count is calculated by summing all existential probabilities recorded for each sequence ID for all nodes found. Item 'a' is frequent if its expected support count is greater or equal to the specified minimum support count. The sub-trees are rooted at this point. A sequence of cumulative product of existential probabilities of items contained in each sequence ID from the root of U-PLWAP tree to all the new roots is then generated. Since this is the only item found so far, the entries in the sequence of the cumulative product of the existential probabilities are the same as the values of the existential probabilities for each sequence ID present in the new root found. The search continues down the tree in order to find another frequent sequence say 'aa'. The same process is repeated once another first occurrence of 'a' is found on all sub-trees. The sequence of cumulative product of the existential probability of items found for each sequence ID is then updated with the product of the existential probability of the newly found 'a' and the last cumulative product of existential probability having the same sequence ID. The expected support count of 'aa' is found by summing all entries in the updated sequence of cumulative products of existential probabilities. If no such 'a' is found, the algorithm backtracks to the last root and search for another item 'b'. The process continues recursively until no more item is found. The algorithm then backtracks to the null root to start mining for sequences starting with a fresh item from the header table.

The terms used in the algorithm are defined as follows:

1. The root of the U-PLWAP tree referred to as root
2. UDB is the uncertain web access sequence database derived from the history of the web log sessions
3. Link header table is a list containing all frequent 1-events from which identical labels in the U-PLWAP tree can be linked to form event queue
4. Root_set is the set of temporary nodes found in each suffix tree
5. Minimum support is the threshold with which counts of events should be greater than or equal to in order to be frequent

Formal presentation of the U-PLWAP algorithm is as shown in Figure 11:

Algorithm 2 : The U-PLWAP algorithm

Input: Uncertain Web Access Sequence Database (UDB) , Minimum Support λ ($0< \lambda \leq 1$)
Output: Complete set of frequent patterns fp in UDB
Begin:

```
   1. Find the frequent 1-items as by calling the Frequent_1_events algorithm (Figure 12)
   2. Insert all frequent 1-items in the Link header table
   3. Build U-PLWAP tree from the UDB by calling U-PLWAP_tree algorithm on Figure 13
   4. Recursively mine the U-PLWAP tree by calling Mine algorithm on Figure 14
End
```

Figure 11: The U-PLWAP algorithm

Algorithm 3: The Frequent_1_event algorithm

Input: Uncertain Web Access Sequence Database (UDB) drawn from the set of events E with existential probabilities p attached to each event e in sequence , Minimum Support λ
Output: Complete set of frequent 1-items F_1, frequency f
Intermediate variables: Iterator k, Check set Ch (records events found already in a sequence)

Begin:

```
   1. f = $\lambda$ * number of tuples in UDB
   2. Initialize $F_1$ to empty set
   3. For each sequence S in UDB
        Begin
          Initialize Check set Ch to empty set
          For each event e in sequence S
            Begin
              If event e $\in F_1$
                  If e $\notin$ Ch
                      Add existential probability e.p to count of e in $F_1$
                      Add e to Check set Ch /*To avoid counting e.p more than once in a sequence */
              Else
                  Add event e to $F_1$
                  Set count of e in $F_1$ to e.p
            End
        End

   4. Set k to the start of $F_1$
   5. While k <> end of $F_1$
        Begin
          If the count of item at k < f
              Remove event at k from the $F_1$
          k= k + 1
        End
   6. Return $F_1$ , f
End
```

Figure 12: Frequent_1-events algorithm

```
Algorithm 4: The U-PLWAP_tree algorithm

Input: Uncertain Web Access Sequence Database (UDB) , Link header table L
Output: U-PLWAP Tree T, Linked Link header table
Intermediate variables: Boolean Variable NodeFound

Begin:
   1. Create root node of T with event label set to NULL, position code set to NULL and count set to 0
   2. For each access sequence S in UDB
        Begin
          Extract frequent sub-sequence S′ from S by removing all events in S but not in L
          Let S′ = $e_1e_2e_3 \ldots e_n$ where $e_i$ are events in S′ and n is the length of S′
          Let current_node point to the root node of T
          For i = 1 to n do
            Begin
              If left child of current_node is NULL, then create a new child node ( $e_i$: 1) and its position code
                  is formed by appending "1" at the end of the position code of the current_node
                  Register existential probability of $e_i$ against the sequence ID of S′
                  Make the current_node point to the newly created node

              Else If current_node is labeled $e_i$, then set Nodefound to true
                  Increase support count by 1 and register existential probability of $e_i$ against the
                  sequence ID of S′

              Else
                  Let the current_node point to current_node.rightsibling, and keep checking whether
                  current node is labeled $e_i$, until there is no more right sibling or $e_i$ is found

              If NodeFound
                  Then increase the count of $e_i$ by 1 and register existential probability of $e_i$ against the
                   sequence ID of S′
                   Make the current_node point to the node of $e_i$
              Else
                   Create a new child node ( $e_i$: 1) and its position code is formed
                   by appending "0" at the end of the position code of the current_node
                   Register existential probability of $e_i$ against the sequence ID of S′
                   Make the current_node point to the newly created node
            End
        End

   3. Entries in the Link header table are then linked to corresponding nodes in the U-PLWAP tree in a pr-
      ordered fashion (from root to left child and then right child)
   4. Return T with the linked Link header table
End
```

Figure 13: U-PLWAP_tree algorithm

The U-PLWAP tree algorithm works by first creating the root nodes and initialising its label and position code to null while its count set to 0. Each sequence is then extracted from the database. For each sequence scanned, its constituent items are checked if they are frequent (from the list of frequent1-items in L). Current node is first made to point to the root node. Starting from the root node, the left child is checked if it exists. If it is NULL, a left child is created and the label set to the item while count is set to 1. The position code is found by appending '1' to the end of position code of current node, the existential probability of the item is also registered against its sequence ID in the set of existential probability values. When a left child already exists and has same label as the item read, its count is incremented by 1 and its existential probability entered against the sequence ID in the set of existential probability values. If however, the left child has a different label from the item read, the U-PLWAP algorithm continues to search all the right siblings for a match. When a match is found, its count is incremented by 1 and its existential probability entered against the sequence ID in the set of existential probability values. In the situation where no match is found in all the right siblings, a fresh right sibling is created. The label of the newly created right sibling is set to the item while count is set to 1. The position code is found by appending '0' to the end of position code of the current node and the existential probability of the item is registered against its sequence ID in the set of existential probability values. The current node is then set to the newly created node.

The next item is then read while still keeping track of the point reached in the tree through the current node pointer. The whole process is repeated until the end of the sequence fetched is reached. A new sequence is then fetched, this time the current node pointer is set to the root of the U-PLWAP tree.

Algorithm 5: The Mine Algorithm

Input: U-PLWAP tree T, Root_set R, Link header table L, set of cumulative Product of existential probabilities K, Frequent n-sequence F, Minimum support λ (R contains root, K and F are empty when the algorithm is called the first time)
Output: Frequent n-sequence F′
Intermediate variables: S stores the information of node whether it is the ancestor of the following node in the queue of similar nodes, C stores the total count of event e_i in different suffix tree

```
Begin:
    1. If R is empty, return
    2. For each event $e_i$ in L, find the suffix tree of $e_i$ in T ( $e_i$ |suffix tree)
          Save first event in $e_i$ –queue to S
          Following the $e_i$ –queue
             If event $e_i$ is the descendant of any event in R, and is not the descendant of S
                Insert node of $e_i$ into new root set R′
                Replace S with $e_i$
                If K is empty
                   Add all existential probability entries in node $e_i$ into C
                   Enter all entries of existential probabilities in node $e_i$ into K′, each identified by sequence ID
                Else
                   Find products of corresponding entries in K and entries of set of existential probability values
                   in node $e_i$
                   Add all the products found to C
                   Enter each product of existential probability values in node $e_i$ into K′ , each identified by
                   Sequence ID

          If C is greater than $\lambda$
             Append $e_i$ to end of F to form F′ and output F′
             Call algorithm 4 passing R′, F′ and K′
End
```

Figure 14: Mine algorithm

The Mine algorithm works by accepting the root of the U-PLWAP tree (The Root_set R is initialised with the root of the U-PLWAP tree), the set of cumulative product K and set of frequent pattern F as input (K and F are set to empty set). The link header table L is also an input. Mining starts by finding frequent sequence that begins with entries in the header table. For each entry in L, the algorithm finds its set of suffix tree(s). Each node found is entered into new set of R (R′) if it is the first occurrence of event e_i along each suffix tree path. The queue linking all nodes of type e_i is followed in order to discover the first nodes in each suffix tree path. The sum of all existential probability entries for these nodes is then added to count C if K (cumulative products found so far) is empty. Each of the existential probability entries is also entered into K′. If K is not empty (Not the first time of calling algorithm 3), each corresponding entries in the cumulative products of existential probabilities of events found before (i.e contained in K) is found and multiplied with the entries of existential probabilities of the current nodes, the sum of which is added to variable count C. The set K′ is updated by each of the products of existential probabilities found with each entry identified by its sequence ID.

The value of count C is then compared to the minimum support value λ. If count is greater or equal to λ, e_i is appended to the end of F and printed as output. The mine algorithm is then called again with new sets R′, F′ and K′. If count is less than λ, then this signals the end of all frequent sequences starting with event e_i. The next event is selected from the header table L and the whole process repeated.

3.5 Example of mining using U-PLWAP

Given the uncertain sequence database shown below:

User_ID	Web logs	Frequent sub-sequences
10	(a:1,b:0.5,c:0.75,d:0.5)	(a:1,b:0.5,c:0.75,d:0.5)
20	(a:1, b:0.25, d:0.5, c:0.25,d:0.5, e:0.2)	(a:1, b:0.25, d:0.5, c:0.25,d:0.5)
30	(a:1,b:0.5,c:0.75,d:0.25, e:0.5)	(a:1,b:0.5,c:0.75,d:0.25)
40	(b:1, c:0.5, a:1, d:0.5, c:0.5, f:0.2)	(b:1, c:0.5, a:1, d:0.5, c:0.5)

Table 39: Sample uncertain sequence mined with U-PLWAP

Assuming the minimum support value is 1 out of 4 tuples, that is 25%. The expected support count values for all the items are calculated by adding up the existential probabilities of the item in each tuple. For example, the existential probability of item 'a' in user ID 10 is 1. The same value (1) is present in user IDs 20, 30 and 40. The expected support count of item 'a' is therefore given as 1 + 1 + 1 + 1 = 4. In the same vein, item 'b' has existential probabilities 0.5, 0.25, 0.5 and 1 in user IDs 10, 20, 30 and 40 respectively. The support count of item 'b' is therefore calculated as 0.5 + 0.25 + 0.5 + 1 = 2.25. The same process is used to find expected support counts for items 'c', 'd', 'e' and 'f' as follows:

Item a = 1+ 1 + 1 + 1 = 4

Item b = 0.5 + 0.25 + 0.5 + 1 = 2.25

Item c = 0.75 + 0.25 + +0.75 + 0.5 = 2.25

Item d = 0.5 + 0.5 + 0.5 +0.25 = 1.75

Item e = 0.2 + 0.5 = 0.7

Item f = 0.2

Items 'e' and 'f' are non-frequent and are removed from the sequences. The resulting frequent sub-sequences are as shown on the last column of Table 39.

The U-PLWAP tree is then built by first creating the root which is null. The first sequence a:1,b:0.5,c:0.75,d:0.5 is entered into the tree as the leftmost sub-tree since there no previously existing nodes. The count for each node initialised to 1 and their corresponding existential probability value are recorded. Each of the existential probabilities are also recorded against the sequence ID. The position codes for the nodes are given using the rule specified in step 3 of section 3.3. Figure 15 shows this process. The second sequence a:1, b:0.25, d:0.5, c:0.5,d:0.5 is read. Starting from the root, since a node with label 'a' already exist with the same existential probability, the count of this node is incremented by 1. Its sequence ID is recorded against its existential probability. Item 'b' is read. The existential probability 0.25 is recorded against sequence ID 20 and the count of label 'b' set to 2. Item 'd' is read. Since no node exists with label 'd' along this path, a new child of node b:2:11 is created. New nodes are also created for items 'c' and 'd' as shown in Figure 16. The other two sequences are read following the same rules as shown in Figure 17 and Figure 18. Header table linking the corresponding nodes in a pre-ordered fashion (by visiting the root, leftmost sub-tree first and then right sub-tree) is also created. The completed linked U-PLWAP tree is as shown on Figure 19

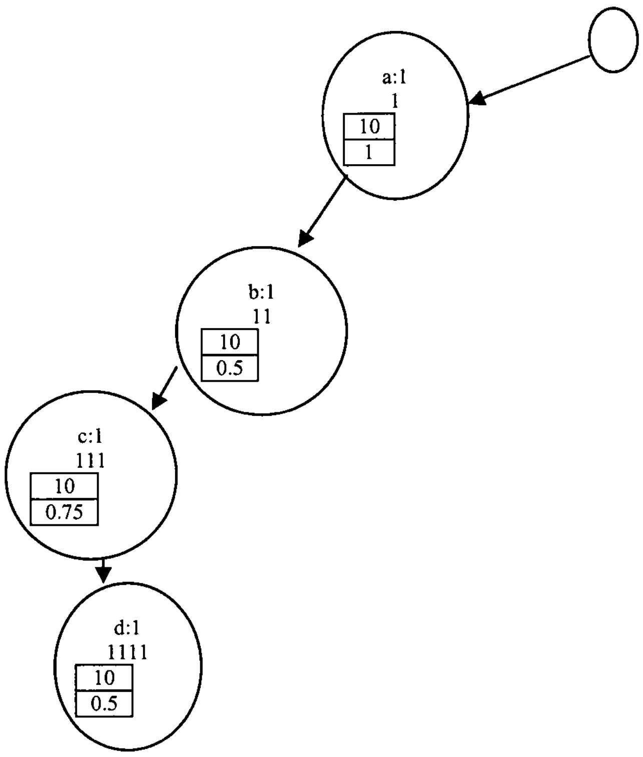


Figure 15: The U-PLWAP tree after reading the first sequence.

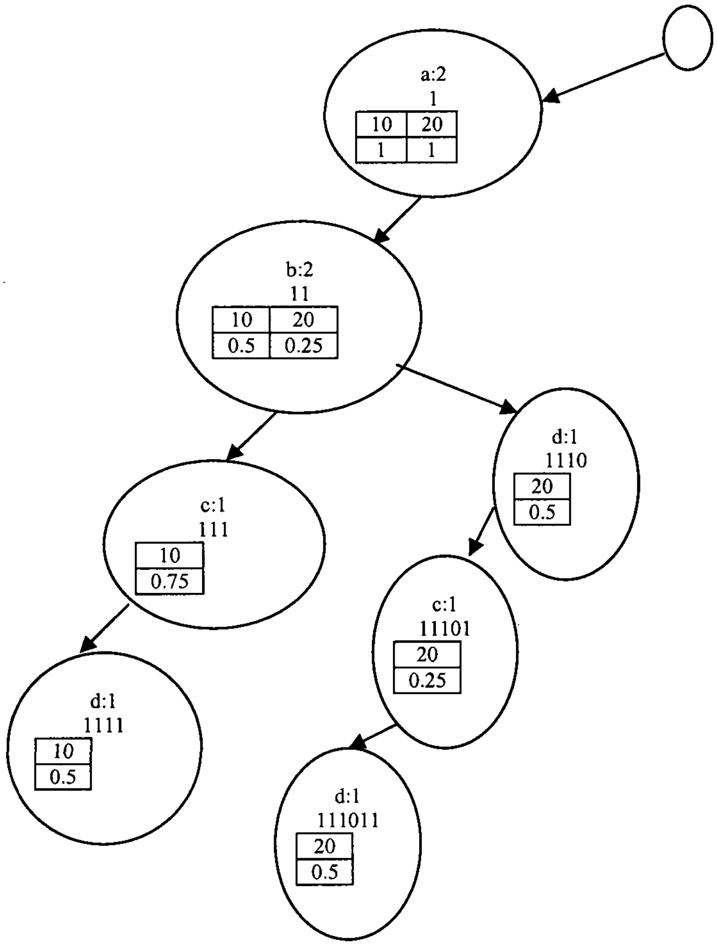


Figure 16: The U-PLWAP tree after the second sequence is read

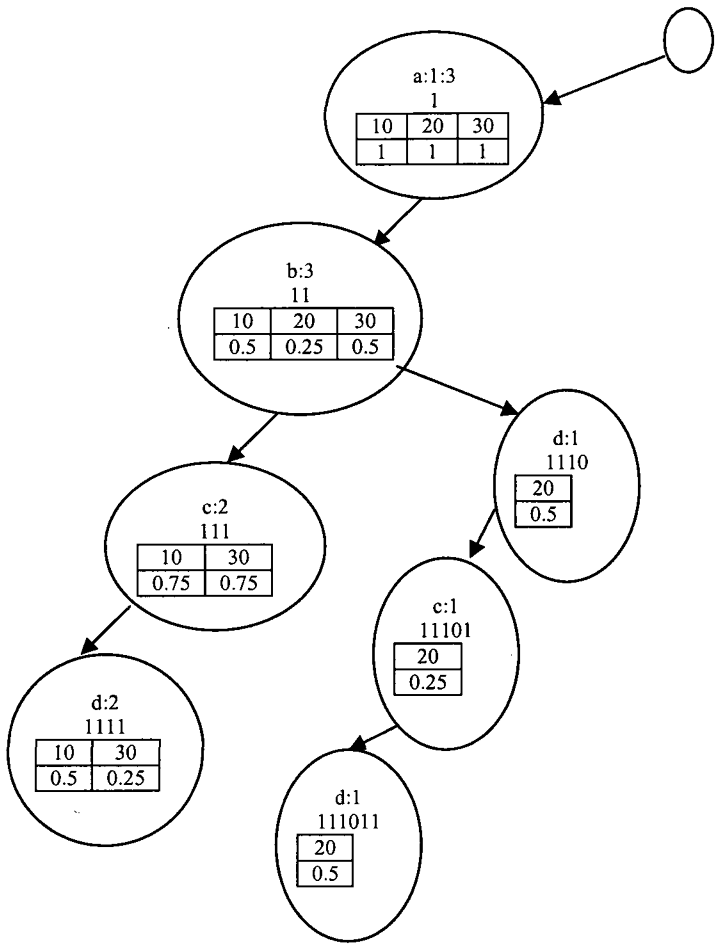


Figure 17: The U-PLWAP tree after the third sequence is read

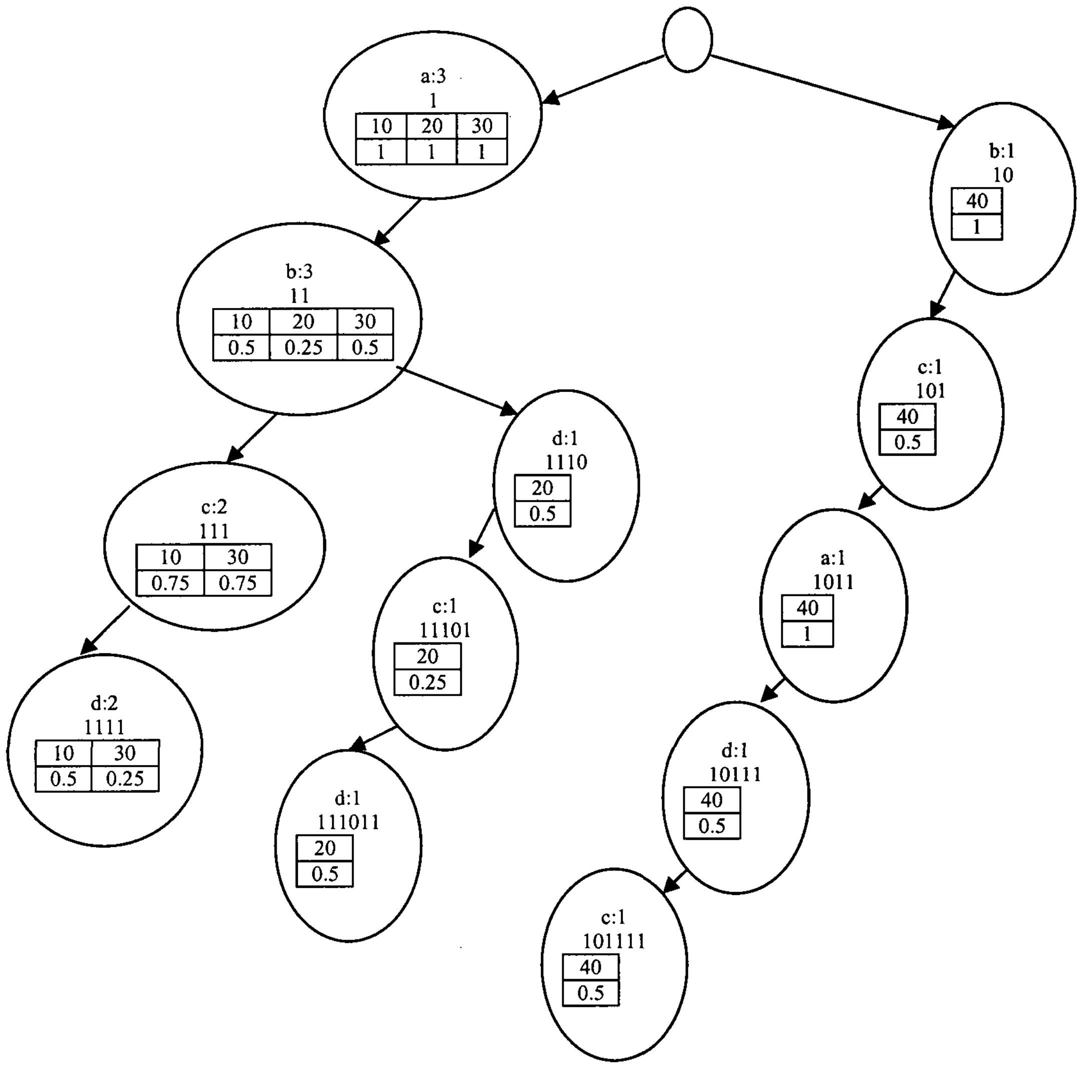


Figure 18: The U-PLWAP tree after the fourth sequence is read

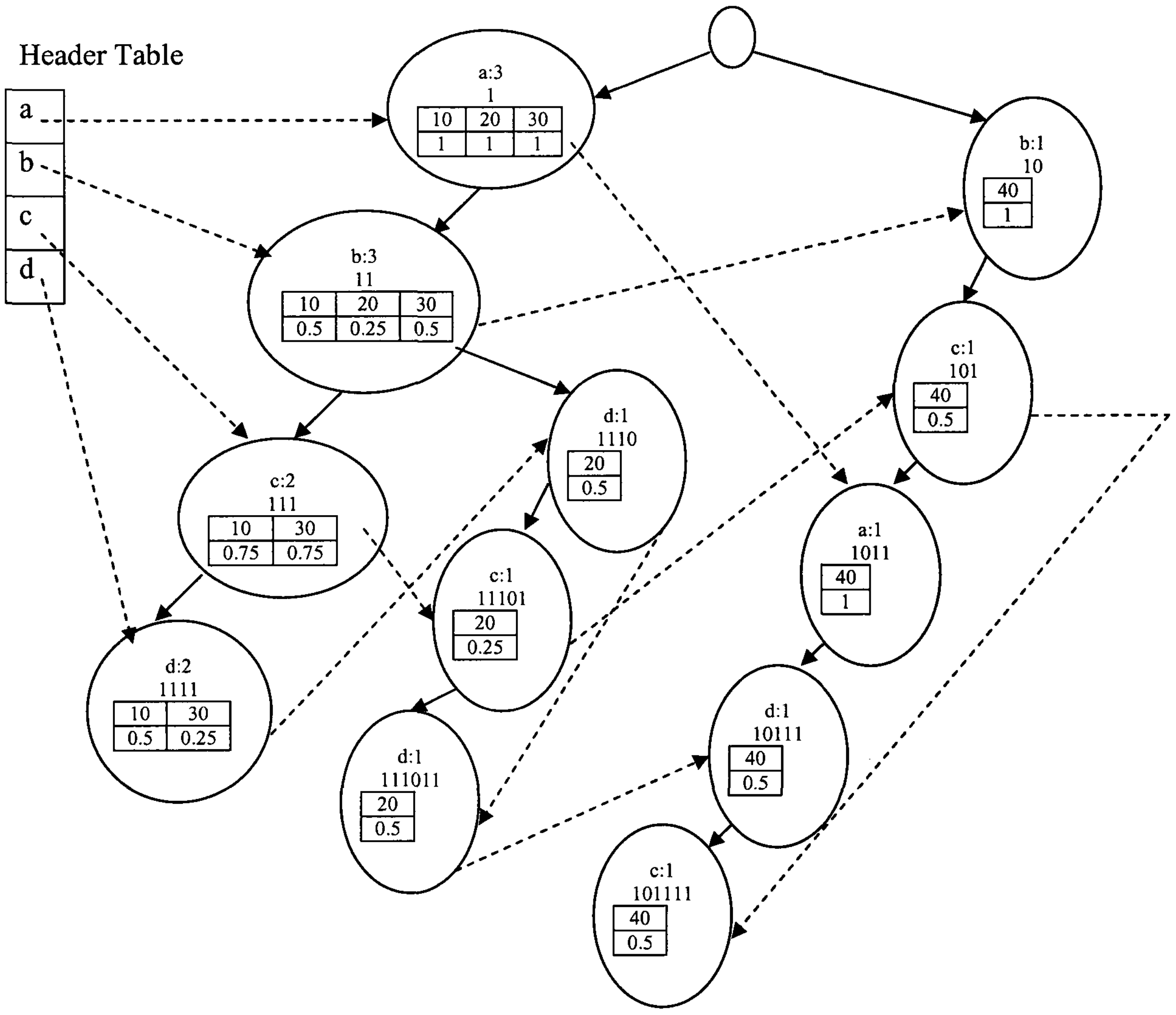


Figure 19: The complete linked U-PLWAP tree

If C is the sequence of cumulative products of sequences found at an immediate old roots and S is the sequence of existential probabilities of new root, with entries of both sequences identified by sequence ID, the expected support count of the sequence (X) found at new root is given as:

Expected support (X) = $\sum(C_{ID} * S_{ID})$

Where C_{ID} and S_{ID} are the entries at corresponding sequence IDs of sequence of cumulative product and sequence of existential probability respectively. It is assumed that the items exist independent of one another (independence of events).

Using Figure 19, mining starts by checking for frequent sequences starting with 'a'. From the root, the U-tree is traversed for the first occurrence of a in the suffix trees of the root node. In this case: a:3:1 and a:1:1011. The expected support of 'a' is then calculated as (1 +1+1) + (1) = 4. This confirms 'a' is frequent since 4 is greater than the minimum support 1. A sequence of cumulative products of existential probability of all items found so far is created at this point. Since these are the first set of items, the sequence is (1, 1, 1, 1) representing each of the 4 paths 10, 20, 30 and 40 respectively. Figure 20 shows this process.

The tree is then rooted at points a:3:1 and a:1:1011 and their suffix trees are traversed for another occurrence of 'a' to check if 'aa' is frequent. No occurrence of 'a' is found. The algorithm then backtracks to the roots a:3:1 and a:1:1011. A new sequence 'ab' is then checked by traversing the suffix trees to search for the first occurrence of 'b'. Item 'b' is found at b:2:11. The suffix trees are then rooted at this point. The expected support for 'ab' is then calculated by summing the product of entries representing each path in the cumulative product sequence and the existential probability values of items in the new found 'b'. The expected support for 'ab' is then found as: (1 x 0.5) + (0.25 x 1) + (1 x 0.5) = 1.25. Since 1.25 is greater than 1, 'ab' is frequent. The entries of the new sequence of cumulative product of existential probability is then generated as (0.5, 0.25, 0.5) representing the paths 10, 20 and 30 respectively as shown in Figure 21.

Sequence of cumulative product of probabilities

10	20	30	40
1	1	1	1

Header Table

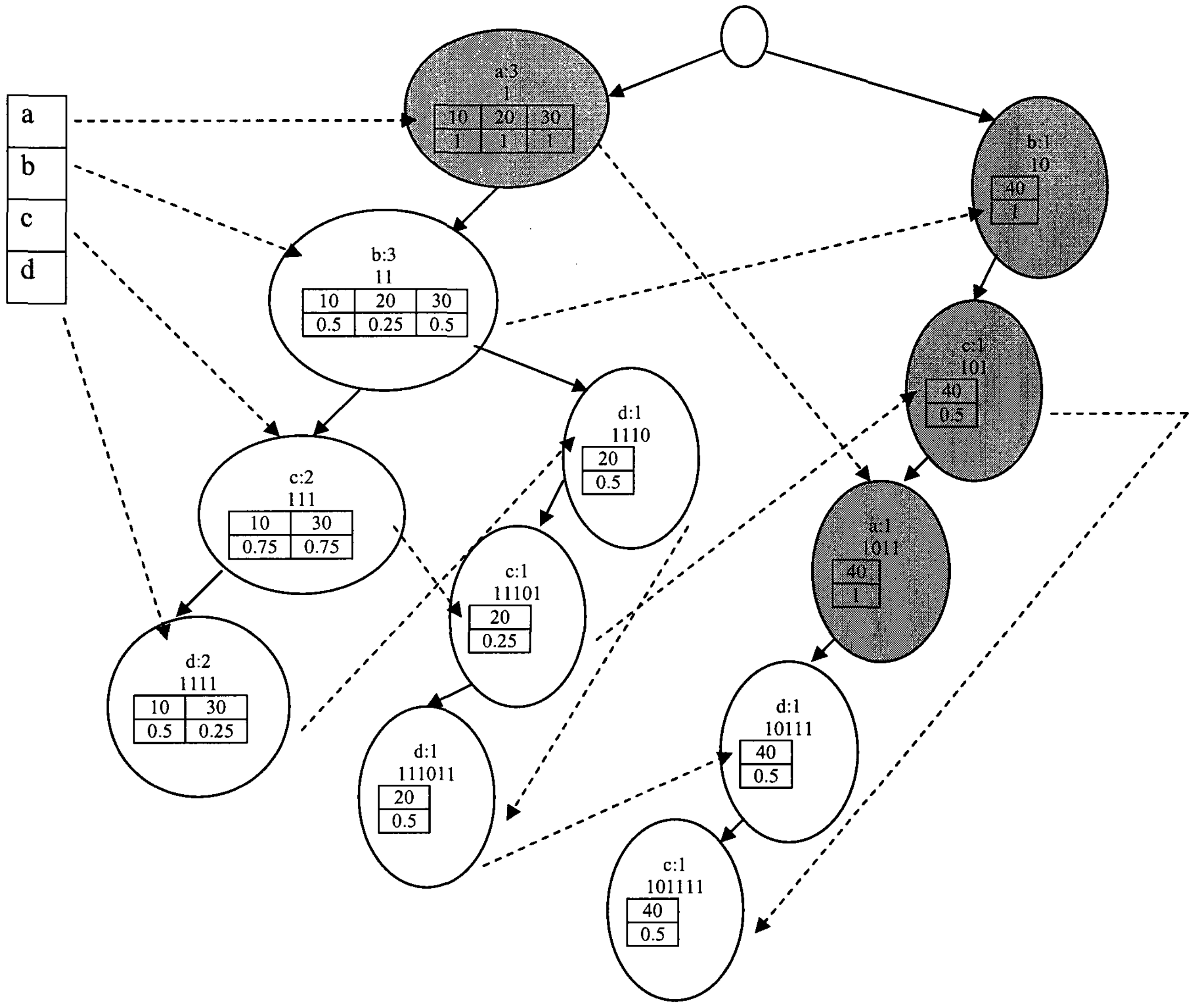


Figure 20: Conditional suffix tree of 'a'

Sequence of cumulative product of probabilities

10	20	30	40
1	1	1	1
0.5	0.25	0.5	-

Header Table

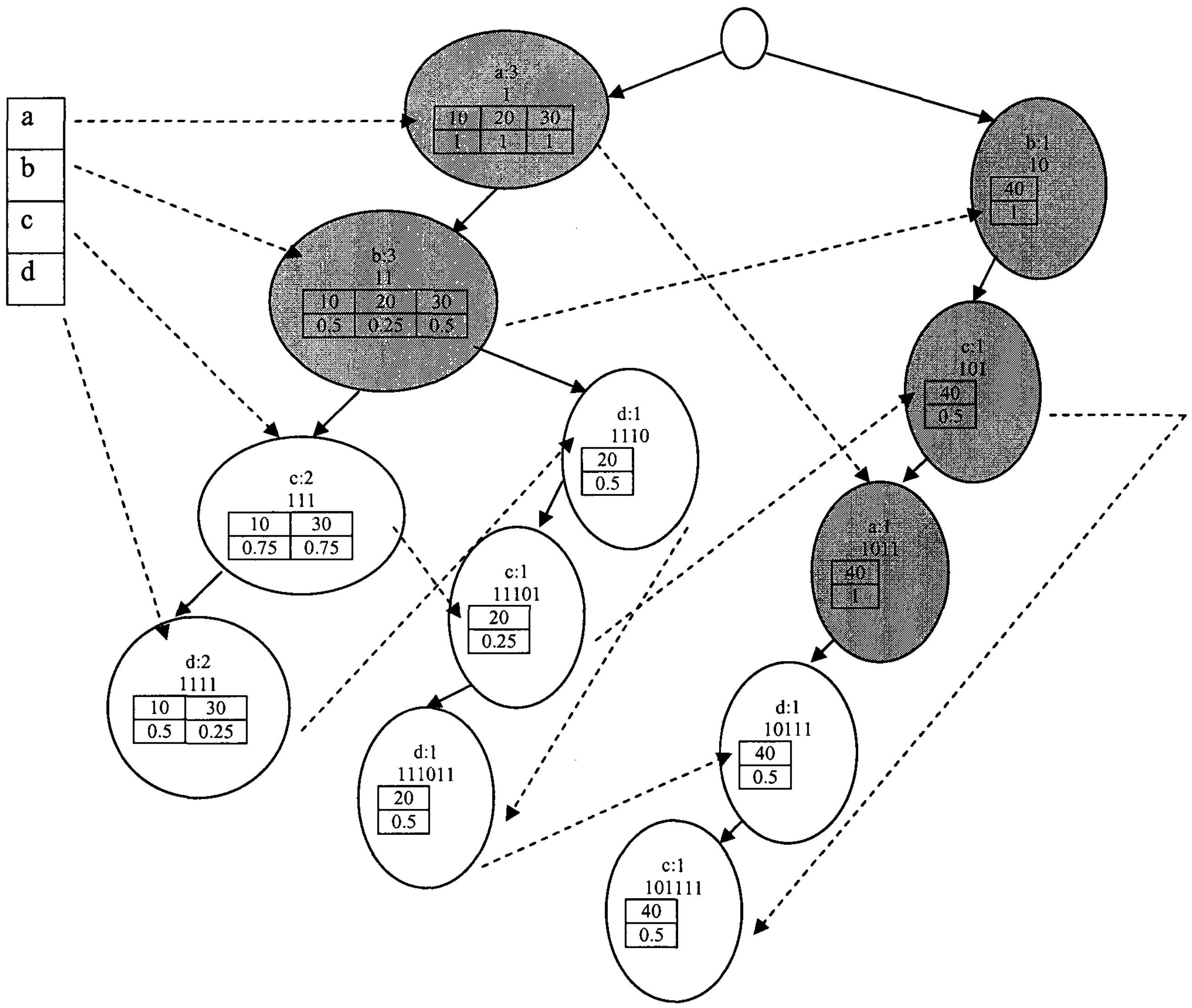


Figure 21: Conditional suffix tree of 'ab'

Using the same technique, 'aba', and 'abb' were not found. Item 'c' is found at nodes c: 2:111 and c:1:11101. The suffix trees are rooted at these nodes. The expected support

of 'abc' is calculated as: (0.5 x 0.75) + (0.5 x 0.75) + (0.25 x 0.25) = 0.8125. 'abc' is therefore not frequent.
The process is recursively repeated following the same logic and the frequent sequences found are: a, ab, ac, ad, b, bc, ba, bd, c, d.

Using the same example in table 39 with the U-apriori technique gives same result. To start with, the frequent 1-items are: a, b, c, d with expected support of 4, 2.25, 2.25 and 1.75 respectively. A self join of this gives the following candidate sequences: aa, ab, ac, ad, ba, bb, bc, bd, ca, cb, cc, cd, da, db, dc, dd. The database is scanned to detect the frequent 2-items. The sequences are ab, ac, ad, bc, ba, bd. Apriori gen join is performed on these sequences and database scanned all over again. The next candidate sequences are: aba, abc, abd, bac, bad. The database is then scanned again to calculate each of the candidate sequence expected support count. None of these is found to be frequent as shown in Table 40 and Table 41.

C2	Expected Support count	Frequent?
aa	-	-
ab	1.25	Yes
ac	2.25	Yes
ad	1.75	Yes
ba	1	Yes
bb	-	-
bc	1.3125	Yes
bd	1	Yes
ca	0.5	No
cb	-	-
cc	0.25	No
cd	0.9375	No
da	-	-
db	-	-
dc	0.25	No
dd	0.25	No

Table 41: U-apriori details for C2

C3	Expected. Support count	Frequent?
aba	-	-
abc	0.8125	No
abd	0.5	No
bac	0.5	No
bab	-	-
bad	0.5	No

Table 40: U-apriori details for C3

4. COMPARATIVE ANALYSIS

This chapter shows analysis of how the proposed solution is more efficient than previously existing U-apriori technique. While PLWAP is an efficient algorithm, it is designed to mine precise data sequence only. UF-growth algorithm though mine uncertain data sequences, it only produces non-sequential frequent patterns.

4.1 Comparing U-PLWAP with U-apriori and UF-growth

The approach of Leung et al. (2008), though cater for uncertain data, only generate non-sequential patterns. Comparing U-PLWAP with UF-growth might not be suitable since they produce different results. Chui et al. (2007) found that LGS outperforms U-apriori algorithm. It is however unclear how the trimmed sequence does not affect the final results in LGS. U-apriori is based on apriori principle that generates candidate sequences. The technique still inherits the problem of repeatedly scanning database and handling long sequences.

In this section, experiments are conducted to see the effect of varying different parameters on both U-PLWAP and U-apriori. The experiments are conducted on Sun Fire 880, UltraSparc III+ processor (8) (1200MHz x 8) with 16GB RAM. The clock speed is 150MHz. The operating system is Unix and the implementation is done with C++. Synthetic data is used from the IBM data generator. Because the IBM data generator does not completely serve the needed input data sequences, the U-PLWAP approach is simulated by generating 3 different versions of the synthetic data from which existential probabilities are generated as described in Figure 10 for each experiment. The following notations are used for the description of the datasets:

|D| = The data size

|C| = The average length of sequence

|N| = Number of unique items

4.1.1 Effect of minimum support on execution time

The following parameters describes the dataset used for this experiment

|D| = 20K

|C| = 6

|N| = 2119

Minimum Support	0.002	0.003	0.004	0.005
U-apriori (sec)	20012	4767	1099	233
U-PLWAP (sec)	10	4	3	2
UF-growth (sec)	22	11	6	3

Table 42: The performance of U-PLWAP, UF-growth and U-apriori with different minimum support

The experiment demonstrates that U-PLWAP is more than 100 times faster than U-apriori in all cases examined. The execution time increases with lower minimum support values as more frequent sequences are found with lower minimum support value. The U-PLWAP algorithm only traverse the tree to find and calculate potentially frequent sequences and their support counts while U-apriori first generate candidate sequences (potentially frequent sequences) and for each sequence scan the database for its support count. Figure 23 also shows that U-PLWAP is at least 50% faster than UF-growth in addition to the richer result U-PLWAP generates.

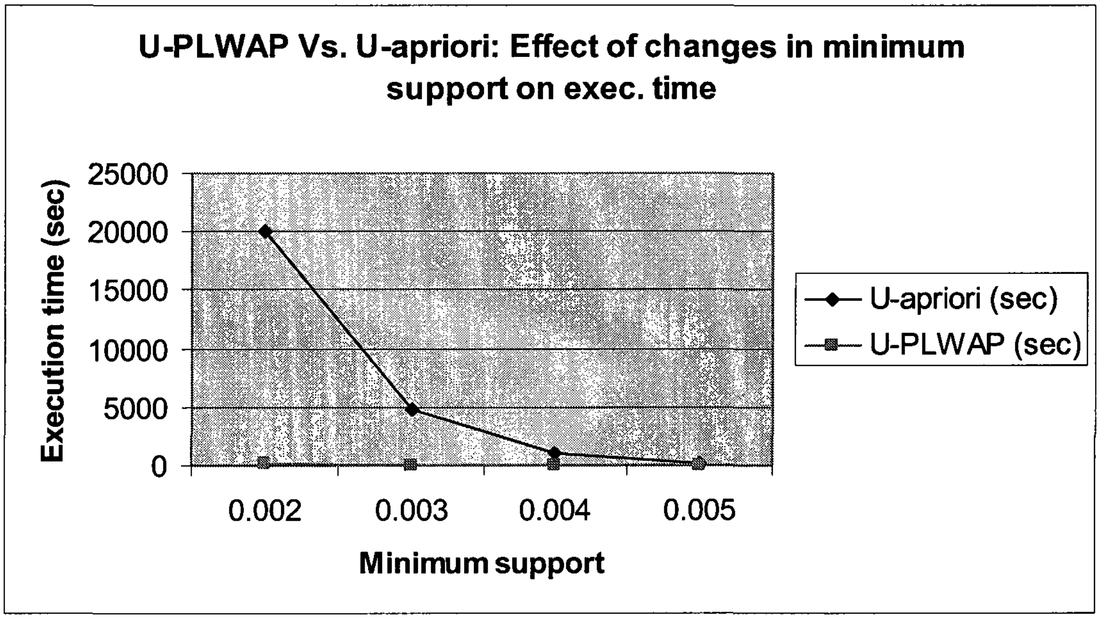


Figure 22: The graph comparing execution time of U-PLWAP with U-apriori when minimum support values vary

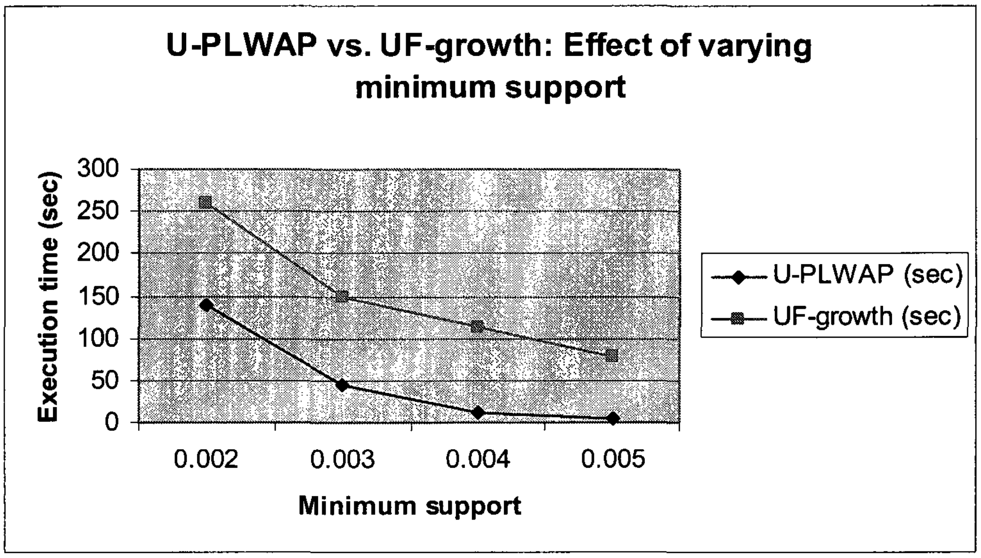


Figure 23: The graph comparing execution time of U-PLWAP with UF-growth when minimum support values vary

4.1.2 Effect of varying data size on execution time

The following parameters are used for the dataset

Minimum support = 0.005

|C| = 6

Average |N| = 2249

Data size (K)	20	40	60	80
U-apriori (sec)	223	474	603	903
U-PLWAP (sec)	2	5	6	9
UF-growth (sec)	3	7	8	12

Table 43: The performance of U-PLWAP, UF-growth and U-apriori with different data sizes

The effect of varying the data size while maintaining the same minimum support value also demonstrates that U-PLWAP is extremely faster with the execution time increasing with increased data size. The U-PLWAP is more than 50 times faster in the cases shown. Figure 25 also demonstrates that U-PLWAP is 33% faster than UF-growth despite the fact that UF-growth only generates frequent non-sequential patterns.

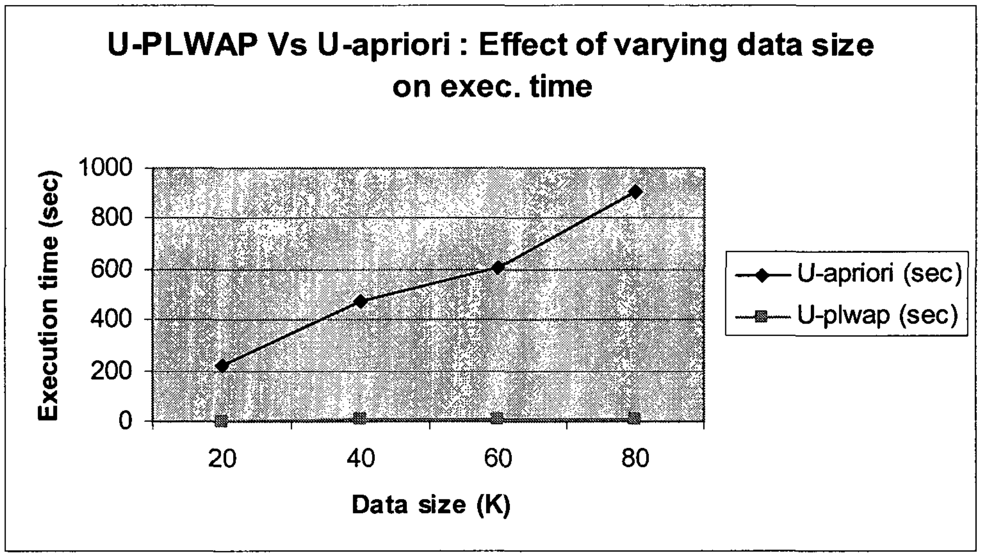


Figure 24: The graph comparing speed of U-PLWAP and U-apriori when data sizes vary

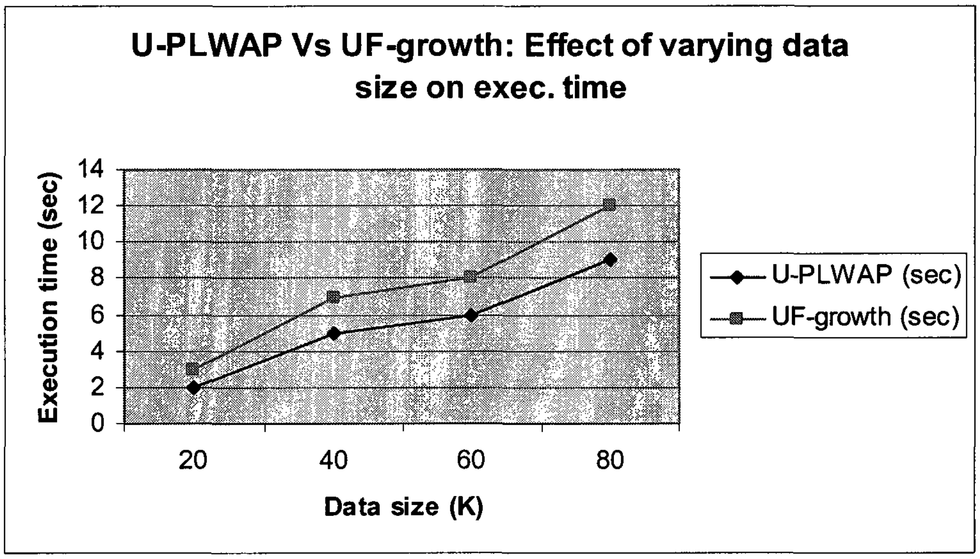


Figure 25: The graph comparing speed of U-PLWAP and UF-growth when data sizes vary

4.1.3 Effect of varying length of sequence on execution time

The parameters used are:

Minimum support = 0.004

|D| = 10K

Average |N| = 6832

Sequence length	10	20	30
U-apriori (sec)	2	1392	30357
U-PLWAP (sec)	1	4	14
UF-growth (sec)	2	6	34

Table 44: Comparison of U-PLWAP, UF-growth and U-apriori with different sequence length

The U-PLWAP algorithm outperforms both U-apriori and UF-growth when length of sequence is varied, but the number of operations involved in U-PLWAP increases with longer sequence. This is because the depth of the U-PLWAP tree is longer therefore taking longer time to traverse the tree during mining. The effect of increased length is also felt in U-apriori during candidate sequence generation. However the bulk of the execution time lies in the database scan. Since the database size is kept constant at 10K for the 3 databases, U-PLWAP is at least 2 times faster than U-apriori. The U-PLWAP is also at least 50% faster than UF-growth. It is however important to note that while U-PLWAP is a frequent sequential mining algorithm, UF-growth is a frequent non-sequential mining algorithm.

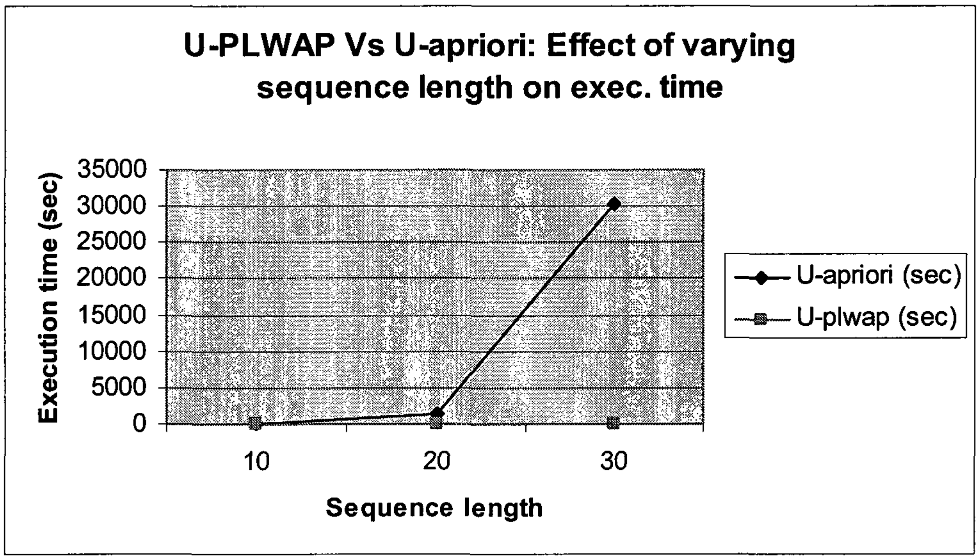


Figure 26: The graph demonstrating the effect of varying the length of sequence on U-PLWAP and U-apriori

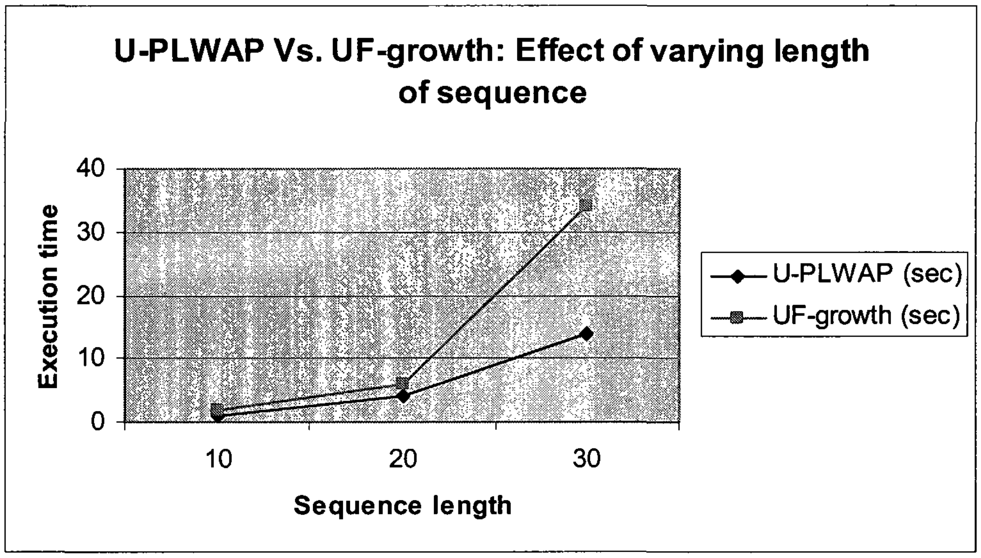


Figure 27: The graph comparing speed of U-PLWAP with UF-growth when length of sequences vary

4.1.4 Effect of minimum support on memory use

|D| = 20K

|C| = 6

|N| = 2119

Minimum Support	0.002	0.003	0.004	0.005
U-apriori (KB)	138000	38000	10000	3680
U-PLWAP (KB)	10000	4624	3080	2464
UF-growth (KB)	4016	2960	2440	2232

Table 45: Memory consumption of U-PLAWP, UF-growth and U-apriori with different minimum support

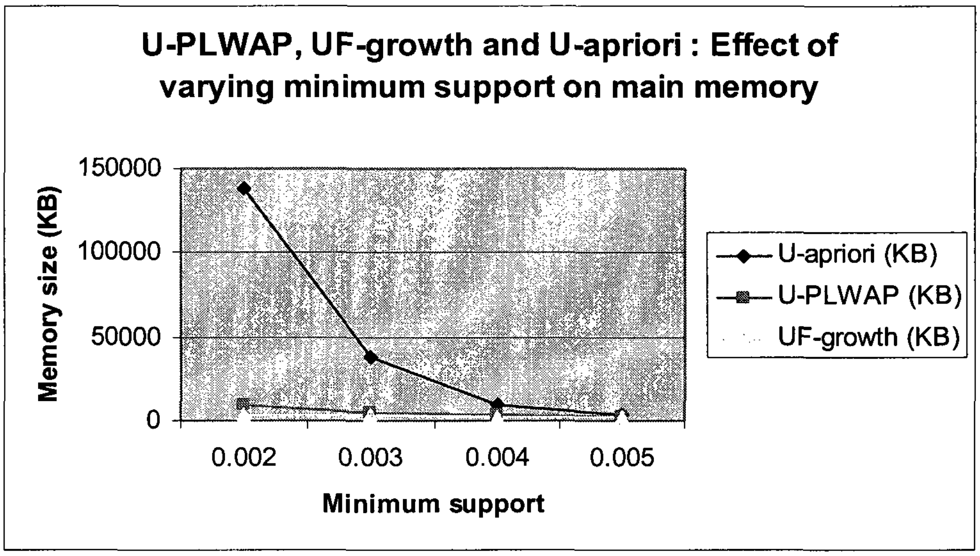


Figure 28: The graph demonstrating the memory need U-PLWAP, UF-growth and U-apriori with different minimum support values

The memory requirement of U-apriori is more than that of U-PLWAP in all cases but the gap shrinks as minimum support increases. The memory requirement of U-apriori reduces drastically with increased minimum support as smaller numbers of candidate sequences are generated. No extra storage is needed for potentially frequent sequences in U-PLWAP as they formed on-the-fly from the U-PLWAP tree traversal. However, lesser memory is required for UF-growth as there is no need to keep track of the order in which items exist.

4.1.5 Effect of varying data size on memory use

The parameters use for the data sets are:

Minimum support = 0.005

|C| = 6

Average |N| = 2249

Data size (K)	20	40	60	80
U-apriori (KB)	3680	3768	3512	3680
U-PLWAP (KB)	2464	2888	3192	3648
UF-growth (KB)	2232	2280	2288	2328

Table 46: Memory size requirement for different data sizes

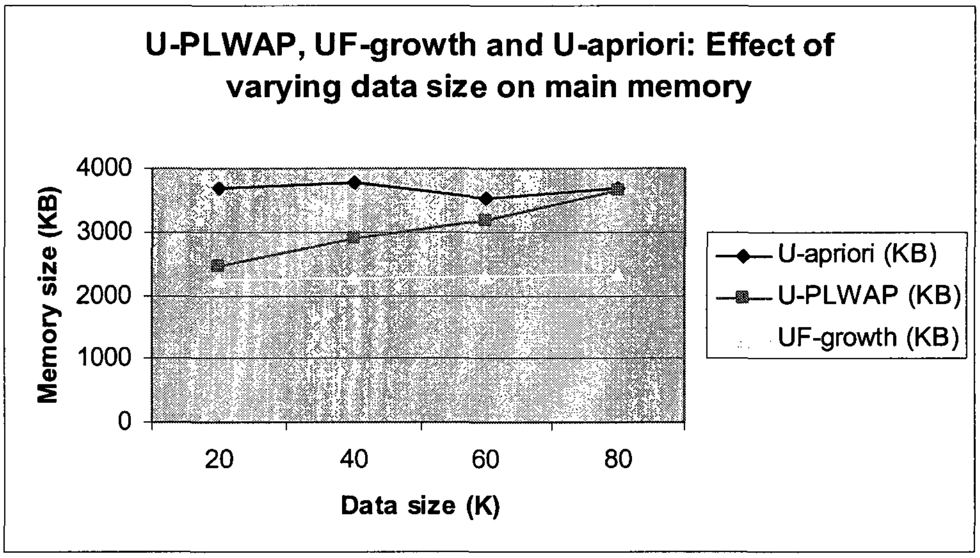


Figure 29: The graph showing memory requirement for the 3 algorithms when data sizes are varied

The memory requirement of U-PLWAP increases with increase in data size but the same cannot be said of U-apriori. This can be attributed to the fact that increase in data size does not automatically increase the amount of data generated during candidate sequence generation. The memory requirement of U-PLWAP is lesser than U-apriori. The UF-growth algorithm has the smallest memory requirement due to its limitation of generating only non sequential frequent patterns. No additional memory is required to track the order of items.

4.1.6 Effect of varying length of sequence on memory use

The parameters are as follows:

Minimum support = 0.004

|D| = 10K

Average |N| = 6832

Sequence length	10	20	30
U-apriori (KB)	2240	14000	162000
U-PLWAP (KB)	2232	2816	10000
UF-growth (KB)	2288	2568	4864

Table 47: Memory requirement of U-PLWAP, UF-growth and U-apriori with different length of sequence

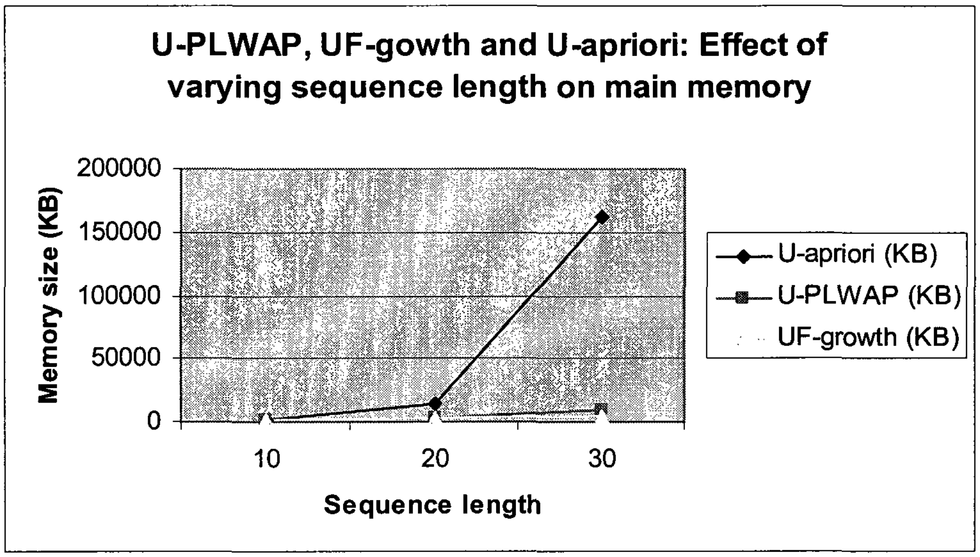


Figure 30: The graph demonstrating the memory requirement of the algorithms with different length of sequence

The memory requirement for U-PLWAP increases with an arithmetic progression while that of U-apriori increases geometrically. U-PLWAP has a lower memory utilization which can be attributed to its non candidate sequence generation. As the length increase, memory is required to hold possibly longer candidate sequence generated. The UF-growth algorithm requires the least memory as no additional memory is required to track the order of items.

4.2 Time complexity analysis

The estimate of the time complexity of both U-apriori and U-PLWAP is shown here. The worst case scenario is considered in both cases. The following notations are used:

N – Number of sequence in the database

F – Number of frequent 1-events

L – Length of the longest sequence

M – Length of the longest event queue

4.2.1 Time complexity of U-PLWAP

The U-PLWAP algorithm also finds frequent 1-events at the first scan of the database, builds the tree U-PLWAP tree with the second database scan. Each frequent 1 event found is then linked with its identical nodes in U-PLWAP tree. The mining is then done recursively with the number of recursive calls for each frequent 1 item bounded by the longest possible sequence (Worst case).

The number of operations for computing the frequent 1-events is:

$A = N \times L$

The number of operations needed to construct the U-PLWAP tree is:

$B = N \times F \times L$

The number of operations needed for mining operation:

$C = (F \times M \times L) + (F \times M \times L\text{-}1) + (F \times M \times L\text{-}2) + \ldots (F \times M \times 1)$

Total time = $A + B + C = (N \times L) + (N \times F \times L) + (FM\Sigma(i))$

where $1 \leq i \leq L$.

Since $\Sigma(i)$ where $1 \leq i \leq L$ is the sum of first L integers,

$\Sigma(i) = ((L (L + 1))/2)$

Total time = $(N \times L) + (N \times F \times L) + FM\ ((L (L + 1))/2)$

$= (N \times L) + (N \times F \times L) + FM\ ((L^2 + L)/2)$

$= (2(N \times L) + 2\ (N \times F \times L) + FM\ ((L^2 + L))) / 2$

Time complexity = $O(2NFL + FM\ (L^2 + L))$

4.2.2 Complexity of U-PLWAP tree in terms of number of nodes

The size of the PLWAP tree is defined by the length of the longest sequence and the number of frequent 1-items in the database.

Total number of nodes is $\Sigma(F^i)$ where $1 \leq i \leq L$

This forms sum of geometric progression where F is both the first term and the common ratio.

Total number of nodes = $F(F^L - 1)/(F - 1)$

Complexity of number of nodes = $O(F^{L+1})$.

5. CONCLUSION AND FUTURE WORK

The research area in data mining and more specifically, web log sequential mining has attracted so much interest in the in the last 15 years. Their applications have been widely used in market basket analysis, re-organisation of web pages and advertisement placement on websites.

Progress has been made in sequential mining, starting from the apriori algorithm by Agrawal and Srikant (1995). Later effort sees progress in GSP, a more efficient version of apriori algorithm. The candidate generation problem associated with apriori-based approached were removed in WAP mine algorithm proposed by Pei et al. (2000). While WAP mine algorithm is much better than apriori algorithm, it is also faced with having to build intermediate trees during mining. Ezeife and Lu (2005) proposed PLWAP in order to solve the problem of recursive construction of intermediate trees. These algorithms are all based on precise and confirmed sequences of data.

Recently there has been effort in mining frequent pattern in areas where existence of data may be uncertain or error prone. The only sequential pattern algorithm proposed in this domain is apriori based which inherits the candidate generating problem of apriori algorithm. The proposed algorithm in this thesis is therefore set out to solve this problem. The U-PLWAP, based on PLWAP algorithm, generates sequential patterns in uncertain sequences without generating candidate sequences, recursive construction of intermediate trees and the need to scan the sequence database repeatedly. It also eliminates the need to traverse the various tree paths in order to scan for existential probabilities of all items found from the root. Instead a sequence of cumulative product of all existential probability is generated at each step of the mining process.

The support count values for each sequence are also calculated based on independence of events. Future work can be based on support counts calculated with conditional probability that is dependent on the probability values of constituent items. Another limitation of this approach is that nodes can grow larger in size as existential probability values are registered for each tuple. Future research will be directed to represent existential probabilities in a more compact form.

BIBLIOGRAPHY

1. Agrawal P., Benjelloun O., Sarma A.D., Hayworth C., Nabar S., Sugihara T. and Widom J. (2006) Trio: A system for Data Uncertainty and Lineage. *In Proceedings of the 32nd international conference on Very large data bases,* Seuol Korea, 1151-1154.

2. Agrawal R., Imielinski T., Swami A. (1993) Mining association rules between sets of items in large databases. *In Proceedings of the ACM SIGMOD Conference on Management of Data*, Washington D.C., USA, 207-216.

3. Agrawal R., Srikant R. (1994) Fast Algorithms for Mining Association Rules. *Proceedings of the 20th VLDB conference*, Santiago, Chile, 487-499.

4. Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. *In Proceedings of 11th International Conference on Data Engineering (ICDE).* Taipei, Taiwan, 3-14.

5. Antunes C. and Oliveira A. (2004) Sequential pattern mining algorithms: Trade-offs between speed and memory. *In Proceedings of 2nd Workshop on Mining Graphs, Trees and Seq.,* 1-12.

6. Ayres J., Flannick J., Gehrke J., Yiu T. (2002) Sequential Pattern Mining using A Bitmap Representation. *In Proceedings of ACM SIGKDD Conference*, Edmonton, Alberta, 429-435.

7. Benjelloun O., Sarma A.D., Halevy A. and Widom J. (2006) ULDBs: Databases with Uncertainty and Lineage. *In Proceedings of the 32nd international conference on Very large data bases*, Seuol Korea, 953-964.

8. Bodon F. (2005) A trie-based APRIORI implementation for mining frequent item sequences. *In Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, Chicago, Illinois, 56-65.

9. Borges J. and Leven M. (1999) Data mining of user navigation patterns. *In Proceedings of the KDD Workshop on Web mining*, San Diego, California, 31-36.

10. Buchner A.G., Baumgarten M., Anand S.S., Mulvenna M.D. and Hughes J.G. (1999) Navigation Pattern Discovery from Internet Data, available at http://www.infj.ulst.ac.uk/~cbgv24/PDF/WEBKDD99.pdf (accessed on 05/04/2008).

11. Burdick D., Calimlim M. and Gerhke J. (2001) MAFIA: A maximal frequent itemset al.gorithm for transactional databases. *In Proceeding 2001 International Conference of Data Engineering*, Heidelberg, Germany 443-452.

12. Chen J. and Cook J. (2007) Mining Contiguous Sequential Patterns from Web Logs. *In Proceedings of the 16th international conference on World Wide Web*, Alberta, Canada, 1177-1178.

13. Cherkasova (1998) Improving www proxies performance with greedy-dual-size frequency caching policy. *In HP Technical Report*, 98-69 (R.1), Palo Alto, 98-69 (R.1).

14. Chiu D., Wu Y. and Chen A.L.P. (2004) An Efficient Algorithm for mining Frequent Sequences by a New Strategy without Support Count. *In Proceedings of 20th International Conference on Data Engineering*, 375 – 386.

15. Chiang F. and Miller R. (2008) Discovering Data Quality Rule. *In PVLDB*, Auckland New Zealand, 1166-1177.

16. Chui C., Kao B. and Hung E. (2007) Mining Frequent Itemsets from Uncertainty Data. *In LNAI*, 4426, 47-58.

17. Cong G., Fan W., Geerts F., Jia X., Ma S. (2007) Improving Data Quality: Consistency and Accuracy. *In VLDB*, Vienna Austria, 315-326.

18. Cooley R. (2003) The Use of Web Structure and Content to Identify Subjectively Interesting Web Usage Patterns. *In ACM Transaction on Internet Technology*, 3(2), 93-116.

19. Cooley R., Mobasher R. and Srivastava J. (1997) Web Mining: Information and Pattern Discovery on the World Wide Web. *In Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence*, 558-567.

20. Cooley R., Mobasher B., and Srivastava (1999) Data preparation for mining World Wide Web browsing patterns. *In journal of Knowledge and Information Systems*, 1(1).

21. Dietterich T.G. and Michalski R.S. (1985) Discovering patterns in sequences of events. *In Artificial Intelligence*, 25:187-232.

22. El-Sayed M., Ruiz C., Rundensteiner E.A. (2004) FS-Miner: efficient and incremental mining of frequent sequence patterns in web logs. *In Proceedings of the 6th annual ACM international workshop on Web information and data management*, 128-135.

23. Etzioni O. (1996) The World Wide Web: Quagmire or gold mine. In Communications of the ACM, 39(1), pp. 65-68.

24. Ezeife C.I and Lu Y. (2005) Mining Web Log Sequential Patterns with Position Coded Pre-Order Linked WAP-Tree. *In Data Mining and Knowledge Discovery*, Springer Science, 10, 5-38.

25. Goethal B. and Zaki M.J. (2003) Advances in frequent itemset mining implementations. *In proceedings of the IEEE ICDM Workshop on frequent itemset mining implementations*, Florida, USA, 90.

26. Han J. and Fu Y. (1995) Discovery of Multiple level association rules from large databases. *In Proceedings of the 21st International Conference on Very Large Databases*, Zurich, Switzerland, 1-12.

27. Han J. and Kamber M. (2006) Data Mining –Concepts and Techniques. Morgan Kaufmann/Springer Verlog, Second Edition.

28. Han J., Pei J., Mortazavi-Asl B., Wang J., Chen Q., Dayal U. and Hsu M. (2000) FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. *In Proceedings 2000 ACM SIGKDD International Conference Knowledge Discovery in Databases*, 355-359.

29. Han J., Pei J., Yin Y. and Mao R. (2004) Mining frequent patterns without candidate generation: A frequent pattern tree approach. *International Journal of Data Mining and Knowledge Discovery*, 8(1), 53-87.

30. Huang Y. and Lin S. (2003) Mining Sequential Patterns Using Graph Search Technique. *In Proceedings of 27th Annual International Computer Software and Applications Conference*, 4-9.

31. Kui G., Bei-jun R., Zun-ping C., Fang-zhong S., Ya-qin W., Xu-bin D., Ning S., Yang-yong Z. (2005) A Top-Down Algorithm for Mining Web Access Patterns from Web Logs. *In Proceedings of 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Berlin, 838–843.

32. Leung C.K., Mateo M.A. and Brajczuk D. A. (2008) A Tree-Based Approach for Frequent Pattern Mining from Uncertain Data. *In Proceedings of 2008 PAKDD*, 653-661.

33. Madria S.K., Bhowmick S.S., Ng W.K. and Lim E.P. (1999). Sampling large databases for finding rules. *In Proceedings of the First International Data Warehousing and Knowledge Discovery*, DaWak99, 303-312.

34. Mannila H., Toivonen H., Verkoma A. I. (1995) Discovering frequent episodes in sequences. *In Proceedings of International Conference on Knowledge Discovery in Databases and Data Mining*, Montreal Canada, 210-215.

35. Masseglia P., Poncelet P., Teisseire M. (1999) Incremental mining of sequential patterns in large databases. In *Actes des Jouenes Bases de Donnes Avances (BDA '00)*, Blois, France.

36. Nanopoulos and Manopoulos (2001) Finding Generalized path patterns for web log data mining. *In Data and Knowledge Engineering*, 37(3), 243-266.

37. Oates T., Schmill M.D., Jensen D. and Cohen P.R. (1997) A family of algorithms for finding temporal structure in data. *In proceedings of the 6th International Workshop on AI and Statistics.*

38. Ouyang W. and Cai Q. (1998) An Incremental updating techniques for discovering generalized sequential patterns. *In Journal of Software*, 9(10), 778-780.

39. Ozden B., Ramaswamy S. and Silberschatz (1998) Cyclic Association Rules. *In Proceeding 1998 International Conference Data Eng. (ICDE)*, 412-421.

40. Parthasarathy S., Zaki M.J., Ogihara M. and Dwarkada S. (1999) Incremental and Interactive sequence mining. *In CIKM*, 251-258.

41. Pei J., Han J. and Mao R. (2000) CLOSET: An efficient algorithm for mining frequent closed itemsets. *In Proceedings 2000 ACM-SIGMOD International Workshop Data Mining and Knowledge Discovery (DMKD),* 11-20, Dallas, TX.

42. Pei J., Han J., Mortazavi-asl B. and Zhu H. (2000) Mining Access Pattern Efficiently from web logs. *Knowledge Discovery and Data Mining*, 396-407.

43. Pei J., Han J., Mortazavi-Asl B., Wang J., Pinto H., Chen Q., Dayal U. and Hsu M. (2004) Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *In IEEE Transactions on Knowledge and Data Engineering*, 16(11), 1424 – 1440.

44. Ren J. and Zhou X. (2005) A New Maintenance Algorithm for Mining Sequential Patterns. *In Proceedings of fourth International Conference on Machine Learning and Cybernetics*, IEEE Computer Society, 1605-1610.

45. Sarma D., Benjelloun O., Halevy A. and Widom J. (2006) Working models for uncertain data. *In Proceedings of International Conference on Data Engineering (ICDE)*, 7.

46. Spiliopoulou M. (1999) The laborious way from data mining to web mining. *In International Journal of Computing Systems, Science and Engineering,* 113-126.

47. Spiliopoulou M. and Faulstich L. (1999) WUM: A tool for Web utilization and analysis. *In extended version of Proc. EDBT '98. LNCS,* 1590, Springer Verlag, 184-203.

48. Srikant R. and Agrawal R. (1996) Mining Sequential Patterns: Generalization and Performance Improvements. *In Proc. 5th Int. Conference Extending Database Technology (EDBT),* Springer-Verlag, Avignon France, 3–17.

49. Srinivasan A., Bhatia D. and Chakravarthy S. (2006) Discovery of Interesting Episodes in Sequence Data. *In Proceedings of the 2006 ACM symposium on Applied computing*, 598-602.

50. Su Z., Yang Q., Lu Y. and Zhang H. (2000) Whatnext: A prediction system for web requests using n-gram sequence models. *In Proceedings of the first International Conference on Web Information System and Engineering*, 200-207, Hong Kong.

51. Wang J., Han J. and Pei J. (2003) Closet+: Scalable and space-saving closed iteemset mining. Submitted for publication.

52. Widom J. (2005) Trio: A System for Integrated Management of Data, Accuracy and Lineage. *In Proceedings of CIDR.*

53. 49. Wu S., Li Y., Xu Y., Pham B. and Chen P. (2004) Automatic Pattern-Taxonomy Extraction for Web Mining. *In Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence,* 242-248.

54. Yan X., Han J. and Afshar R. (2003) CloSpan: Mining Closed Sequential Patterns in Large Datasets. *In Proceedings of 3rd SIAM International Conference on Data Mining*, San Francisco, 1–12.

55. Yang Q., Zhang H. and Li T. (2001) Mining Web Logs for Prediction Models in WWW Catching and Pre-fetching. *In Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, San Francisco, California, 473 – 478.

56. Yang Z. and Kitsuregawa M. (2005) LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern. *In IEEE Proceedings of the 21st International Conference of Data Engineering*, 1222 – 1222.

57. Yen S. and Chen A.L.P. (1996) An efficient approach for discovering knowledge from large databases. *In Proceedings of 4th IEEE International Conference on Parallel and Distributed Information Systems*, 8-18.

58. Zaiane O.R., Xin M. and Han J. (1998) Discovering Web Access Patterns and Trends by applying OLAP and Data Mining Technology on Web Logs. *In Proceedings of Advances in Digital Libraries Conference*, 19-29.

59. Zaki M. J. (2001) SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning Journal,* 42(1) 31-60.

60. Zaki M.J. and Hsiao (2002) CHARM: An efficient algorithm for closed itemset mining. *In Proceeding 2002 SIAM International Conference Data Mining*, Arlington, VA 457-473.

VICTA AUTORIS

Olalekan Habeeb Kadri was born in 1978 in Oke-Ola-Oro, Nigeria. He graduated from Offa Grammar School in 1995. From there, he went on to the University of Ibadan, Nigeria where he obtained a B Sc. in Computer science in 2002. He equally, obtained a Master's degree in Information Systems at Roehampton University, London, UK in 2008. He is currently a candidate for Master's degree in Computer Science at the University of Windsor and hopes to graduate in Spring 2010.