ITC 3/48
Journal of Information Technology and Control
Vol. 48 / No. 3 / 2019
pp. 415-431
DOI 10.5755/j01.itc.48.3.21137

HARPP: HARnessing the Power of Power Sets for Mining Frequent Itemsets

Received 2018/11/13 Accepted after revision 2019/07/16

http://dx.doi.org/10.5755/j01.itc.48.3.21137

HARPP: HARnessing the Power of Power Sets for Mining Frequent Itemsets

Muhammad Yasir
Department of Computer Science, University of Engineering and Technology Lahore, Faisalabad Campus, Pakistan, phone: +92 333 666 2312; e-mail: muhammadyasir@uet.edu.pk

Muhammad Asif Habib
Department of Computer Science, National Textile University, Faisalabad, Pakistan, Phone: +92 332 333 1979; e-mail: drasif@ntu.edu.pk

Shahzad Sarwar
Punjab University College of Information Technology, University of the Punjab, Lahore, Pakistan; e-mail: s.sarwar@pucit.edu.pk

Chaudhry Muhammad Nadeem Faisal, Mudassar Ahmad, Sohail Jabbar
Department of Computer Science, National Textile University, Faisalabad, Pakistan; e-mails: nadeem.faisal@ntu.edu.pk, mudassar@ntu.edu.pk, sjabbar.research@gmail.com

Corresponding author: drasif@ntu.edu.pk

Modern algorithms for mining frequent itemsets face the noteworthy deterioration of performance when minimum *support* tends to decrease, especially for sparse datasets. Long-tailed itemsets, frequent itemsets found at lower minimum *support*, are significant for present-day applications such as recommender systems. In this study, a novel power set based method named as HARnessing the Power of Power sets (HARPP) for mining frequent itemsets is developed. HARPP is based on the concept of power set from set theory and incorporates efficient data structures for mining. Without storing it entirely in memory, HARPP scans the dataset only once and mines frequent itemsets on the fly. In contrast to state-of-the-art, the efficiency of HARPP increases with a decrease in minimum *support* that makes it a viable technique for mining long-tailed itemsets. A performance study shows that HARPP is efficient and scalable. It is faster up to two orders of magnitude than FP-Growth algorithm at lower minimum *support*, particularly when datasets are sparse. HARPP memory consumption is less than that of state-of-the-art by an order of magnitude, on most datasets.

KEYWORDS: Association Rules, Frequent Itemset Mining, Apriori, FP-Growth, Recommendation Systems.

1. Introduction

In data mining, mining association rules is a key problem that determines associations among items such that the existence of some items implies the occurrence of other items, in the same transaction. Its maiden use in retails helped companies in making better business decisions, such as which items should be put on sale, which items to be placed jointly on shelves, and how to tailor strategies of marketing [1]. The ever-increasing use of association rule mining has become an indispensable tool due to its tremendous power of extracting and furnishing profound insights about data. The applications span over various domains that include medical applications [7, 15, 24, 27, 43], Internet and web security [11, 35, 62, 65], predicting natural disasters [41], recommender systems [3, 44, 48, 61], weather forecasting [57], and market basket analysis [2].

Mining of association rules is a two-step process [1]. In the first step, all frequent itemsets are discovered. Frequent itemsets are those itemsets that are present in an adequate number of transactions higher than the minimum *support,* a predefined threshold for a minimum number of transactions. In the second step, association rules are learned in a straightforward manner by using the frequent itemsets discovered in the first step. Thus, the performance of association rule mining techniques is heavily dependent on discovering frequent itemsets, in the first step.

Frequent itemset mining is regarded as a vital task due to its wide-ranging use in data mining, such as mining association rules, correlations, and episodes [31]. It is a process of finding groups of items from transactions contained within a database [1]. A transactional database contains a sequence of transactions where each transaction corresponds to a basket of items purchased by a customer. Giant retailers such as Amazon, Netflix, YouTube, and e-Bay additionally recommend pertinent products/items of interest to the user, based on frequent itemsets techniques utilizing the history of previous similar users [4].

State-of-the-art algorithms for mining frequent itemsets have been comprehensively investigated on dense datasets. Mining performance on these datasets has been greatly improved over the years. However, these algorithms have not been adequately validated on real sparse datasets. Their efficiency is below par on these datasets and becomes more inferior when the minimum *support* threshold is further decreased. In this paper, a novel method, HARnessing the Power of Power sets (HARPP) is proposed that efficiently discovers the frequent itemsets at lower minimum *support* thresholds especially from real sparse datasets.

The rest of the paper is structured as follows. Section 2 presents related work. Section 3 presents important definitions and problem description in detail. HARPP algorithm is presented in Section 4. An example of HARPP is discussed in Section 5. Section 6 presents the detailed experimental results and performance study. Section 7 summarizes the study and highlights future research issues.

2. Related Work

The naive brute-force frequent itemset mining algorithm [26] generates all possible itemsets first and then counts their *support*. It discards the itemsets whose *support* is less than minimum *support* threshold. Brute force algorithm traverses each itemset (except empty set) in the database to count its *support*. This algorithm is not a complex one but it is quite inefficient as the number of itemsets grows in exponential order. If I be the set of all distinct items, then for a reasonably large I, this algorithm wants the enormous memory to store 2^I itemsets which makes it not viable in veracity. Today's giant retailers such as Amazon and Netflix might store thousands of items, thus I tends to be very large. If one can circumvent the massive hunger of memory, the performance of mining can be made better significantly.

Apriori is a benchmark algorithm for mining frequent itemsets. It is based on *candidate-set generation* and *test* approach [2]. To improve the performance, Apriori adheres to the principle of hierarchical monotonicity, which states that a subset of a frequent itemset must be frequent too. Similarly, a superset of an infrequent itemset must be infrequent. For mining, Apriori has to scan the entire database numerous times to create candidate itemsets and then to discover frequent itemsets. In the beginning, all items in the database are declared candidate k-itemsets. The entire database is then scanned and the *support* count of each candidate itemset is incremented. Based on the minimum *support* threshold, the frequent itemsets are then generated. These frequent k-itemsets help in generating the candidate (k+1) itemsets. A rescanning of the database is performed for counting the *support* of each candidate itemset. This procedure is repeated until frequent k-itemsets cannot be generated anymore. The number of candidate k-itemsets generated is reduced because all combinations of k-itemsets are not considered. Therefore, Apriori is better than the brute-force algorithm.

Apriori heuristic-based approaches are adopted by a number of studies [2, 30, 36, 39, 45, 49-50, 54], however, they have to deal with the over-abundant generation of candidate itemsets and then to count their *support*. Repetitive scanning of the database [32] is another substantial limitation. To avoid repetitive database scanning, many vertical mining techniques were proposed [14, 52, 63-64]. Each itemset is represented vertically (such as diff-set or Tid-set). Set intersection is used for *support* counting of itemsets, which is advantageous as the entire database is not scanned for this purpose.

Highlighting the processing overhead associated with the generation of a massive amount of candidate itemsets by Apriori, direct hashing pruning (DHP) algorithm was proposed [47]. It was claimed that most of the processing overhead occurs during the creation of large 2-itemsets. Therefore, by improving the initial generation of candidate itemsets, the performance of the algorithm can be improved. Second, a performance-affecting factor highlighted was the quantity of the data that were scanned during the discovery of frequent itemsets. DHP provided frequent itemset generation in an efficient manner and reduced the size of the database effectively too. Perfect Hashing and Pruning (PHP) [46] optimized DHP by using perfect hashing while creating hash tables for $C\kappa$+1. This effectively eliminates the hash table collisions that were evident in DHP and consequently $C\kappa$+1 contains the actual counts of the $C\kappa$+1 itemsets lessening the need to recount the occurrence of $C\kappa$+1 itemsets in D.

To further improve the performance, sampling [55] and the dynamic counting of itemsets [12] were also proposed, which made the stringent division between counting and producing candidates a bit softer. As soon as a candidate itemset qualifies the minimum *support* threshold, based on it DIC begins the process of generating further candidates. DIC employs a prefix tree and performs faster. To devise more efficient solution following Apriori principle, Cluster-Based Association Rule (CBAR) algorithm was proposed that utilized clustering mechanism [56]. CBAR performs database scanning once and then it constructs cluster tables. During scanning of the database, it clusters a transaction record, whose length is k to the k-th cluster table.

The procedure of generating candidate 2-itemsets in CBAR is similar to that of the Apriori algorithm. CBAR contrasts candidate 2- itemsets against the 2nd cluster. If an itemset meets the minimum threshold requirement, it is believed as frequent itemset and it is not further checked within the larger clusters. Similarly, the candidate 3- itemsets are compared against the 3rd cluster and so forth. CBAR is advantageous as it generates frequent itemsets by contrasting with the partial cluster tables as compared to Apriori where each itemset is compared with the whole database.

An enhanced cluster based association rule mining algorithm was proposed and based on this, a recommender system prototype was implemented [23]. It performed two optimizations in the CBAR algorithm. Firstly, it minimized the database by eradicating all the infrequent items before clustering the transactional dataset. It performed two database scans, first to identify the large 1- itemsets and to discover the infrequent itemsets. Another scanning was done to eradicate infrequent itemsets from all the transactions to do their clustering. The second optimization was the addition of a counter for each transaction. If a transaction comes for the first time, the counter was set to one and the transaction is added to the appropriate cluster table. But if a certain transaction comes more than once, then it is not entered into the cluster, instead, the counter is incremented by one. This optimization reduced the size of cluster tables and the counter helped to identify the frequent large itemsets easily.

Based on extended prefix-tree structure, FP-Growth stores the database in a trie structure and each item has a linked list going through all transactions in which that item exists. This data structure is denoted by FP-tree (Frequent-Pattern tree), which is highly condensed [32]. A counter is stored in each node to keep track of the number of transactions sharing the branch through that node. A link is also stored, which points to the next existence of that particular itemset in the FP-tree. In this way, it links together all occurrences of an itemset in the FP-tree. A header table is also maintained, which contains each distinct item together with its *support* and a connection to the first occurrence of the item in the FP-tree. It incorporates a divide-and-conquer strategy for mining frequent itemsets.

Like FP-tree, PPC-trees (Pre-order Post-order Code trees) were proposed to store the information of frequent itemsets [21]. PPC tree was found to be more efficient than FP-tree because the algorithm traverses the tree once to determine the N-list of frequent 1-itemsets. Whereas algorithms employing FP-tree have to traverse the tree a number of times. Based on PPC-tree the PrePost algorithm was proposed [22]. It first builds a PPC-tree by using a tree construction algorithm, then N-lists corresponding to 1-itemsets are generated. N-list represents transaction ID list (TID list) in compressed form, which depicts the features of an itemset. Then a divide-and-conquer technique is used to discover frequent itemsets. It is better than FP-tree because it does not construct further additional trees in forthcoming iterations.

Nodeset is another itemset representation based on PPC tree, in which encoding of a node is done by pre-order or post-order code. An algorithm FIN is proposed based on Nodeset, which is as efficient as the PrePost but consumes less memory [19]. PrePost algorithm has a limitation due to following Apriori-like approach to mine frequent itemsets, even it has adopted single-path property of N-list for pruning search space. To overcome this limitation, PrePost+ algorithm was proposed [20]. To represent frequent itemsets, PrePost+ uses N-list and mines frequent itemsets directly. It employs Children-Parent Equivalence pruning to reduce the search space and to avoid the repetitive search.

The concept of *subsume index* was proposed to further enhance the mining efficiency [53]. The subsume index of a frequent 1-itemset represents a list containing frequent 1-itemsets that are co-occurring with it. Based on subsume index, an algorithm NSFI was proposed [58]. NSFI combined N-Lists and the concept of subsume index to mine frequent itemsets more efficiently while capturing less memory. NSFI used a hash table to create N-lists corresponding to frequent 1-itemsets to gain more efficiency. Furthermore, it has improved the procedure of N-list intersection and mined frequent itemsets without determining their associated N-lists by using subsume index. PrePost$^+$ is found to be the most efficient one.

On dense datasets, state-of-the-art techniques including FP-Growth algorithm and its high-flying successors such as PrePost [22], FIN [19], PrePost+ [20], and NSFI [58] achieve great efficiency. In these studies, the majority of the datasets used for the evaluation are dense and comprised of limited transactions, having less than 100,000 transactions except *Accidents* dataset. Further, it is worth mentioning that for higher minimum *support*, several million frequent itemsets are discovered from these datasets. Predominantly, there is no substantial distinction among efficiencies of state-of-the-art techniques when applied on these datasets. However, when the minimum *support* is low, the distinction becomes noticeable in some cases, such as the FIN on dataset *connect*, the PrePost on dataset *Accidents*, and the PrePost$^+$ on some datasets. But abundant frequent itemsets are discovered at this minimum *support* level, whose significance is somewhat arguable.

Due to their inherent characteristics, the performance of these algorithms is not up to the mark when applied on sparse datasets [22]. For example, NSFI is applied to only one sparse dataset (*Retail* dataset) and its running time is slightly less than that of PrePost. FP-Growth, FP-Growth*, and PrePost achieved almost the same running time while working on sparse datasets [22]. Likewise, PrePost+ is applied to only one sparse dataset (*kosarok* dataset). Results are evident that FIN, PrePost, and PrePost+ have achieved similar running time when applied on this dataset. Moreover, a drastic increase in their running times can be seen when minimum *support* tends to decrease further (less than 0.4% for *kosarok* dataset) [20]. The efficiency of FP-Growth degrades when the patterns become longer and/or minimum support decreases because it constructs conditional FP-trees in abundance during mining [29]. This shows that on sparse datasets mining efficiency of these algorithms is far from a satisfactory level, especially when minimum *support* is low.

Moreover, these algorithms make two passes over the database, therefore, they have to store the entire database in memory. If the database is too large to be stored in memory, they are unable to run [28]. Frequent itemsets discovered at lower minimum *support* are more interesting for recommendation systems [42]. In fact, these itemsets represent 80% long-tailed items that are not frequently rated but have absolute importance. Recommendation systems have to exhibit scale-free behavior [8] to recommend these itemsets. Whereas, modern algorithms avoid to discover them, due to swift growth in their running time at lower *minsup* [28].

To thwart these issues, a novel method, HARnessing the Power of Power sets (HARPP) is proposed to find frequent itemsets at lower *minimum* support thresholds. Additionally, a novel measure called *agility* is introduced, which reveals how much accelerative an algorithm is. It refers to the number of frequent itemsets discovered per second with a decrease in minimum *support*. Three real sparse datasets along with one dense dataset have been chosen to compare the efficiency and scalability of HARPP with that of the FP-Growth and Apriori algorithm. Rigorous experimentation has revealed that a decrease in minimum *support* affects the efficiency of FP-Growth and Apriori to a great extent, whereas HARPP performs even better.

3. Basic Concepts

This section presents the concepts related to HARPP. Table 1 presents the notations and their descriptions.

Let $I = \{i_1, i_2, i_3, \ldots, i_m\}$ represent the set of all items. Let $DB = \{T_1, T_2, T_3, \ldots, T_n\}$ represent a database that contains n transactions, where each T_k $(1 \leq k \leq n)$ is a transaction that is a set of items such that T_k is a subset of I. HARPP considers each transaction, T in DB as a *set*, which enables HARPP to perform *set*-related operations (such as union, intersection, and power set) on it. X denotes an itemset if X is a set of items. A transaction T contains X if and only if X is a *Sub-Set* of T. T may contain one or more itemsets. If T contains only one itemset such as {A}, then it contains a single itemset / *Sub-Set*, 'A', neglecting empty set. An example dataset, *DB*, is presented in Table 2 that will be used for illustration throughout this paper.

Table 1
Notations

Notation	Description
DB	Database of transactions.
T	A *set* ADT representation of a transaction
|*DB*|	Total number of transactions in *DB*
X	An itemset
P	Powerset of *T*
minsup	Given threshold
Sub-Set	A subset of *T* stored in *P* except empty subset.
F	A *set* ADT containing all frequent *Sub-Set*s / Itemsets
Dict	A *dictionary* ADT to store each *Sub-Set* as key and its support (frequency of occurrence) as value.

Table 2
A database of transactions, *DB*

TID	*T*
T_1	A, B, C, D
T_2	A, B, C
T_3	A, B, D
T_4	A, B
T_5	C, D, E

Here, T_1, T_2, T_3, T_4, and T_5 contain 4, 3, 3, 2, and 3 itemsets, respectively.

The power set, P of T is the *set* containing T and all of its *Sub-Set*s neglecting empty subset. The following formula calculates the total number of *Sub-Set*s in P.

Total *Sub-Set*s in P of $T = 2^{(\text{No. of } X \text{ in } T)} - 1$

Sample transactions and their corresponding power sets are shown in Table 3. Power set, P of T_1, T_2, and T_3 contain 7, 3, and 15 *Sub-Set*s respectively. There exist a number of similar/overlapping *Sub-Set*s (itemsets present in more than one power sets) such as {W}, {X}, and {W, X}.

Table 3
Sample transactions with their power sets

TID	*T*	Total *Sub-Set*s in *P* of *T*
T_1	W, X, Y	{{W}, {X}, {Y}, {W, X}, {W, Y}, {X, Y}, {W, X, Y}}
T_2	W, X	{{W}, {X}, {W,X}}
T_3	W, X, Y, Z	{{W}, {X}, {Y}, {Z}, {W, X}, {W, Y}, {W, Z}, {X, Y}, {X, Z}, {Y, Z}, {W, X, Y}, {W, X, Z}, {W, Y, Z}, {X, Y, Z}, {W, X, Y, Z}}

5. HARPP: The Proposed Method

HARPP borrows the concept of power set from set theory. It iteratively generates power sets to make combinations of overlapping varying-sized subsets of *I,* where *I* is a set of items in a large database.

The five distinguishing factors of HARPP are the following:

1 It does not store the entire database in memory for mining frequent itemsets. Therefore, it requires the least amount of memory.

2 It makes a single pass over the database, and mines frequent itemsets on the fly.

3 Efficiency is achieved by using *set* and *dictionary* data structures. Most of their operations such as (itemset containment check and insertion /deletion) take constant running time.

4 HARPP gradually becomes more efficient as minimum *support* tends to decrease, whereas state-of-the-art techniques perform inversely.

5 It has presented a novel yet simple approach for solving an intricate problem.

The pseudocode of HARPP is presented in Figure 1. HARPP consists of the procedure *Find_Frequent*( ), which does the following tasks.

1.1. Step (1) repeats the subsequent steps for each transaction, *T*.

Figure 1
Pseudocode of HARPP

```
Algorithm: HARPP Algorithm
Input: A database of transactions DB, minimum support (minsup), and |DB| be the total number of
transactions in DB.
Output: F
Call Find_Frequent (DB, minsup, |DB|)
Find_Frequent (D, minsup, |DB|)
(1) For each T in DB, do
(2)   If T is not already present in F, then do
(3)     Make P of T
(4)     For each Sub-Set in P, do
(5)       If Sub-Set is not already present in F, then do
(6)         If Sub-Set is not already present in Dict, do
(7)           Store Sub-Set as a key in Dict and set its value equal to 1
(8)         Else, do
(9)           Increase value of Sub-Set by 1
(10)        If value of Sub-Set = minsup, then do
(11)          Pop out Sub-Set from Dict and store into F
(12)        End If
(13)        End If
(14)      End If
(15)    End For
(16) Delete Dict
(17) Return F
```

1.2. Step (2) checks if *T* is already present in *F*. This step is referred to as 1st containment check. If *T* is already present then it is considered as a frequent itemset and discarded. The procedure then goes back to Step (1) to read next *T*. Otherwise, the power set, *P* of *T* is generated at Step (3).

1.3. In Step (4), for each *Sub-Set* of *P*, the following sub-tasks are done.

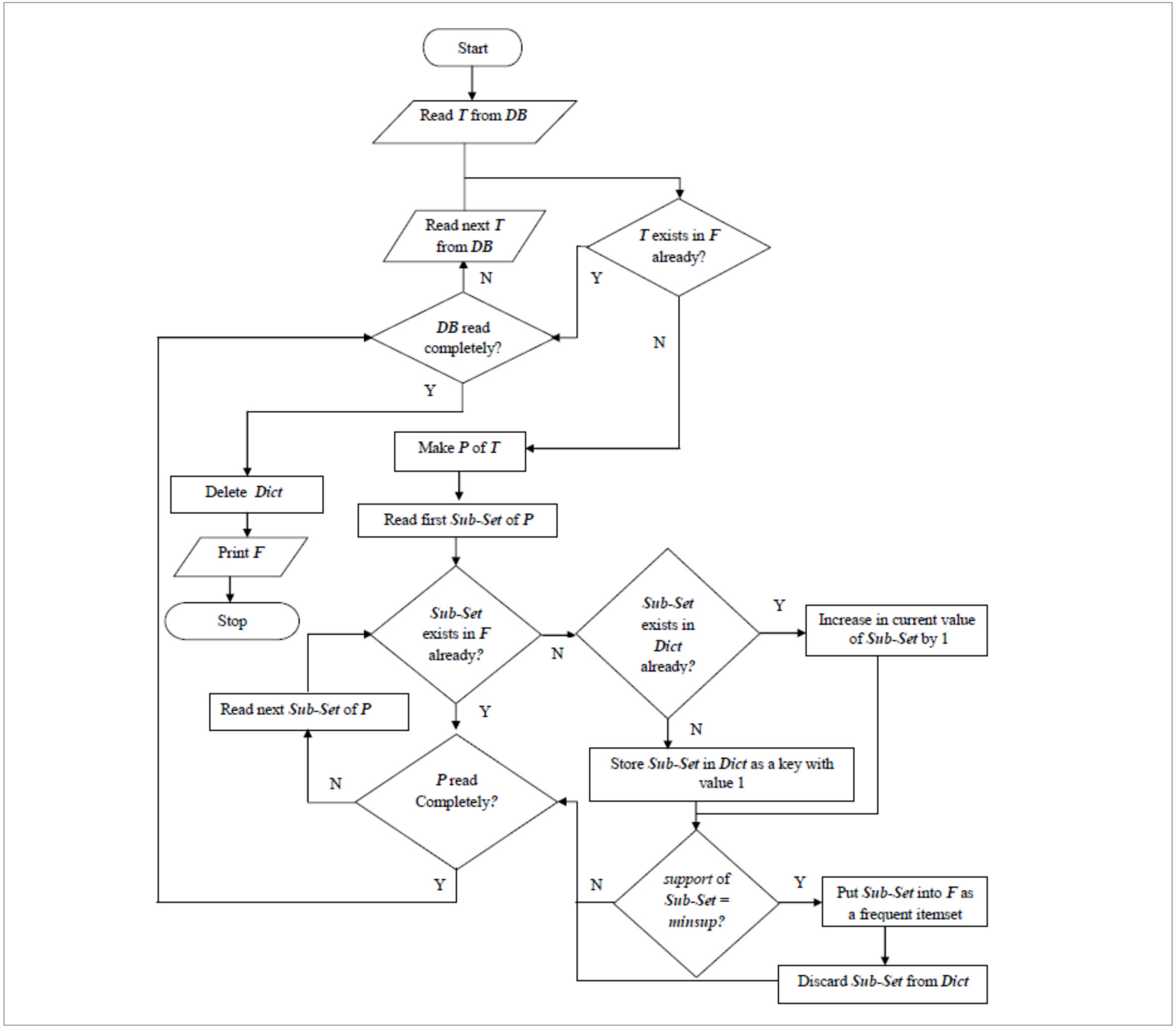
   1.3.1. Step (5) reads a *Sub-Set* and checks if it exists in *F* already. This step is called 2nd containment check. If a *Sub-Set* exists in *F*, it is believed to be a frequent itemset, discarded, and next *Sub-Set* is read from *P-Set*. Otherwise, Step (6)-(7) stores this *Sub-Set* as a key in *Dict* with value (*support*) equal to 1, if not present already. If *Sub-Set* is already present in *F*, then Step (8)-(9) increments its existing value by 1.

   1.3.2. After storing the *Sub-Set* in *Dict*, its value is compared with *minsup* in Step (10). If value becomes equal to *minsup*, this *Sub-Set* is declared frequent itemset.

   1.3.3. Then the frequent itemset (*Sub-Set)* is deleted from *Dict* and stored in *F* in Step (11).

The flow chart of the procedure *Find_Frequent* ( ) is shown in Figure 2.

Figure 2
Flow chart of the procedure *Find_Frequent*( )


Start
Read T from DB
T exists in F already?
Read next T from DB
DB read completely?
Y
N
Make P of T
Delete Dict
Print F
Stop
Read first Sub-Set of P
Sub-Set exists in F already?
Sub-Set exists in Dict already?
Increase in current value of Sub-Set by 1
Read next Sub-Set of P
Store Sub-Set in Dict as a key with value 1
P read Completely?
support of Sub-Set = minsup?
Put Sub-Set into F as a frequent itemset
Discard Sub-Set from Dict

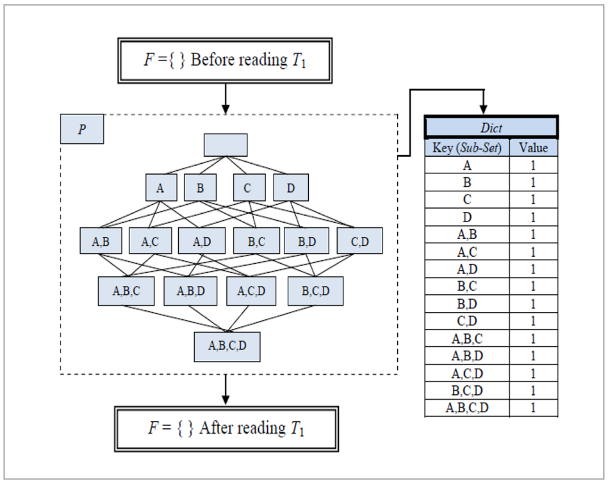
5. An Example

The dataset presented in Table 2 is used for this example and *minsup* is set to 60%. According to this dataset, if an itemset is present in 3 transactions (60%), it will be declared a frequent itemset. HARPP calls the procedure, *Find_Frequent* ( ). Figures 3-8 depict the processing of this procedure.

In Figure 3, it reads T_1 and checks its existence in F. Since F is empty before reading T_1, it means that T_1 is not frequent so far. Therefore the procedure proceeds and power set, P of T_1 is created. Then iteratively, one *Sub-Set* of this *P-Set* at a time is stored into *Dict* as a key with value 1, and a comparison of its value (*support*) and *minsup* is made. If its *support* becomes equal to *minsup*, it is declared frequent. This is shown in Step (10) of the pseudocode given in Figure 1. Because the *support* of all *Sub-Sets* is less than *minsup* so far, none of them is declared frequent. F remains empty after reading T_1.

Figure 3

Power set of T_1 and states of F and *Dict*



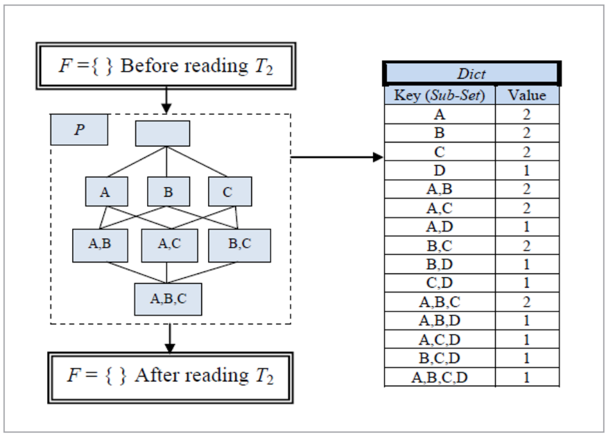
Dict	
Key (*Sub-Set*)	Value
A	1
B	1
C	1
D	1
A,B	1
A,C	1
A,D	1
B,C	1
B,D	1
C,D	1
A,B,C	1
A,B,D	1
A,C,D	1
B,C,D	1
A,B,C,D	1

In Figure 4, HARPP reads T_2, checks its existence in F. Since F is still empty, the power set of T_2 is made. After storing each *Sub-Set* of T_2 as a key into *Dict*, value (*support*) of some *Sub-Sets*, {A}, {B}, {A,B}, {A,C}, {B,C}, and {A,B,C} becomes 2, as they are already present there. This is shown by Step (9) of the pseudocode in Figure 1.

In Figure 5, HARPP reads T_3 and checks whether it exists in F or not. Since F is still empty, the power set of T_3 is made. After storing each *Sub-Set* as a key into *Dict*, value (*support*) of some *Sub-Sets* such as, {A}, {B}, and {A, B} becomes 3, which is equal to *minsup*. As soon as the value of a *Sub-Set* becomes equal to *minsup*, it is declared frequent, discarded from *Dict* and stored into F. This is shown by Steps (9)-(10) of the pseudocode. So these three *Sub-Sets* are popped out and stored into F.

Figure 4

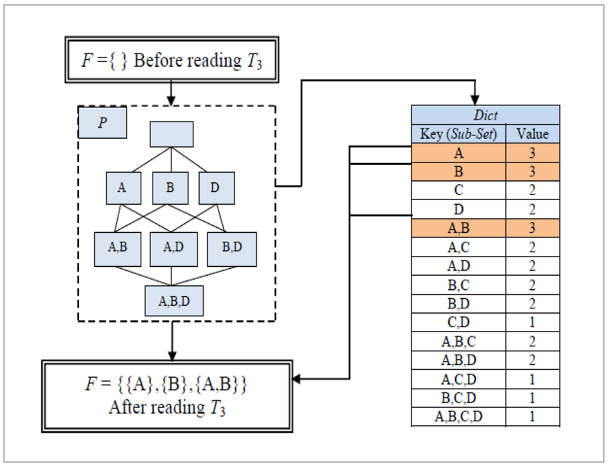
Power set of T_2 and states of F and *Dict*



Dict	
Key (*Sub-Set*)	Value
A	2
B	2
C	2
D	1
A,B	2
A,C	2
A,D	1
B,C	2
B,D	1
C,D	1
A,B,C	2
A,B,D	1
A,C,D	1
B,C,D	1
A,B,C,D	1

Figure 5

Power set of T_3 and states of F and *Dict*



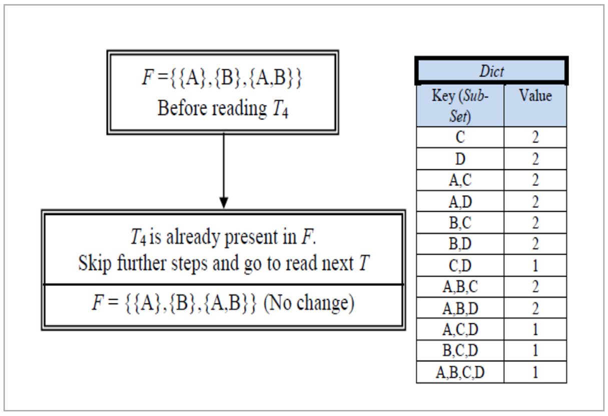
F = { } Before reading T3
P
A
B
D
A,B
A,D
B,D
A,B,D
F = {{A},{B},{A,B}} After reading T3

Dict	
Key (*Sub-Set*)	Value
A	3
B	3
C	2
D	2
A,B	3
A,C	2
A,D	2
B,C	2
B,D	2
C,D	1
A,B,C	2
A,B,D	2
A,C,D	1
B,C,D	1
A,B,C,D	1

In Figure 6, HARPP reads T_4 and checks if it is already present in F. Since T_4 is already present in F, without doing subsequent processing of this iteration, HARPP goes to read next transaction, thereby forbidding redundant computations. This is shown by Step (2) of the pseudocode given in Figure 1.

Figure 6

Power set of T_4 and states of *F* and *Dict*

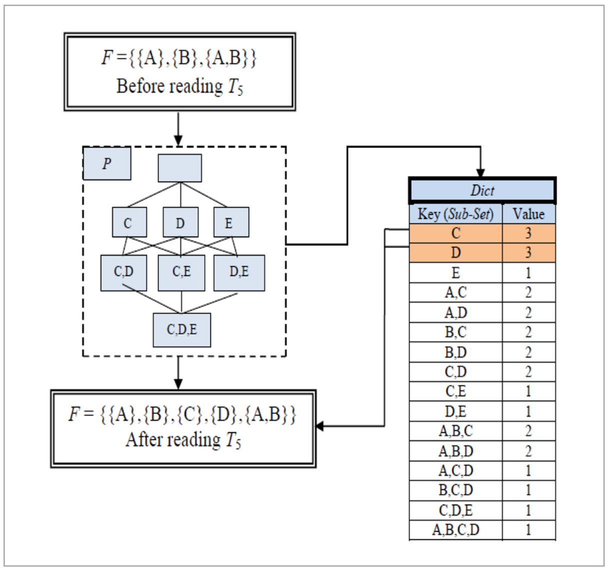

F ={{A},{B},{A,B}}
Before reading T_4
T_4 is already present in F.
Skip further steps and go to read next T
F = {{A},{B},{A,B}} (No change)

Dict	
Key (*Sub-Set*)	Value
C	2
D	2
A,C	2
A,D	2
B,C	2
B,D	2
C,D	1
A,B,C	2
A,B,D	2
A,C,D	1
B,C,D	1
A,B,C,D	1

In Figure 7, HARPP reads T_5 and then checks its existence in *F*. Though *F* is not empty but it does not contain T_5. Therefore the power set, *P* of T_5 is made. After storing each *Sub-Set* as a key into *Dict*, value (*support*) of some *Sub-Sets*, {C}, and {D} also becomes 3, which is equal to *minsup*. Therefore, both of these *Sub-Sets* are declared frequent, popped out from *Dict* and stored into *F*.

Figure 7

Power set of T_5 and states of *F* and *Dict*

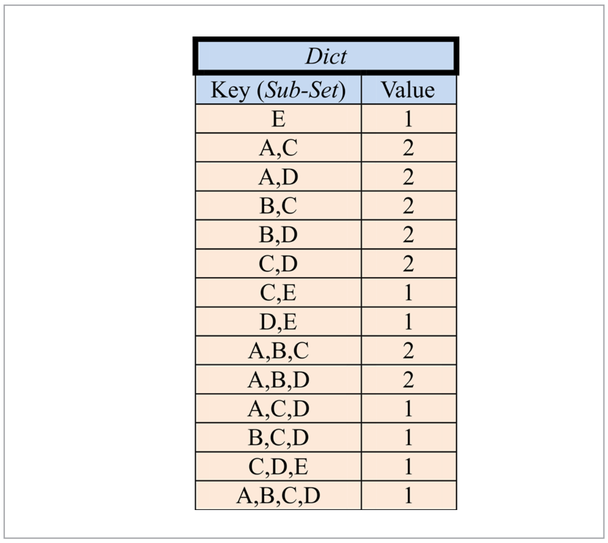


Dict	
Key (*Sub-Set*)	Value
C	3
D	3
E	1
A,C	2
A,D	2
B,C	2
B,D	2
C,D	2
C,E	1
D,E	1
A,B,C	2
A,B,D	2
A,C,D	1
B,C,D	1
C,D,E	1
A,B,C,D	1

Figure 8 shows the state of *Dict* when all frequent itemsets are deleted from it, added into *F*, and HARPP is going to terminate.

Figure 8

The state of *Dict* when HARPP is going to terminate



Dict	
Key (*Sub-Set*)	Value
E	1
A,C	2
A,D	2
B,C	2
B,D	2
C,D	2
C,E	1
D,E	1
A,B,C	2
A,B,D	2
A,C,D	1
B,C,D	1
C,D,E	1
A,B,C,D	1

6. Experimental Evaluation

The performance metrics are running time, peak memory consumption, and *agility*. The experimental results of HARPP, Apriori, and FP-Growth are presented. Moreover, the scalability of HARPP has been evaluated.

6.1. Experiment Setup

Four real-world datasets are chosen for performance testing of HARPP. Table 4 summarizes the features of the datasets.

Table 4

Features of Datasets

Dataset	Distinct items (*I*)	Transactions	Category
PowerC	140	1,040,000	Sparse
Online Retail	2,603	541,909	Sparse
Skin	11	245,057	Dense
Extended Bakery	50	75,000	Sparse

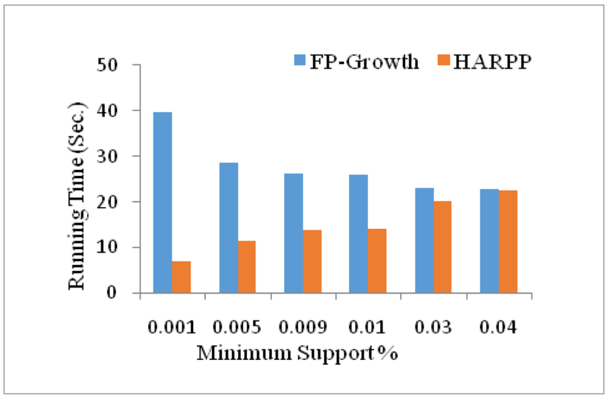
Three datasets including *PowerC*, *Online Retail*, and *Skin* are available at UCI Repository [10]. For mining frequent items, converted versions of these datasets are taken from [25]. *Extended Bakery* dataset is taken from [17]. Apriori and FP-Growth have been chosen as the baseline algorithms. FP-Growth has been chosen because the efficiency gap between itself and its successors on sparse datasets is not significant. HARPP is implemented in Python. The Python implementation of Apriori is taken from [33]. The Python implementation of FP-Growth is taken from the *pymining* library having version 0.2 [16]. A computer with 8G memory, Intel Core i7-3667U, 2.0 GHz processor, and Windows 8 Pro x64 Edition is used to perform all the experiments.

6.2. Comparison of Running Time

The comparison of running times for HARPP, Apriori, and FP-Growth are presented in Figures 9-12. Figure 9 shows the comparison of running times of HARPP, Apriori, and FP-Growth on *PowerC* dataset.

Figure 9

Running time on *PowerC* dataset

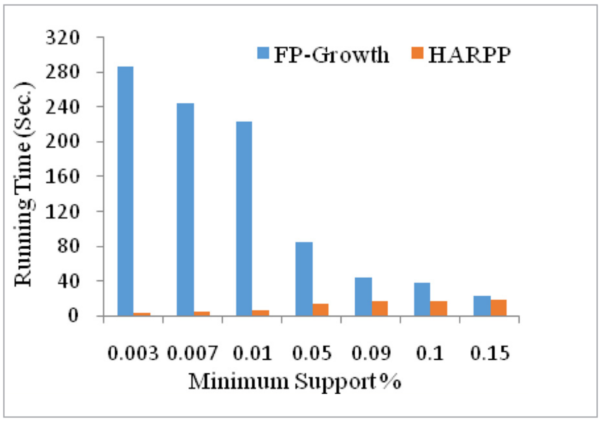


The result of Apriori is not plotted because its running time exceeds 3,000 seconds at *minsup* 0.04%, which shows that its running time is higher than HARPP by more than 2 orders of magnitude. Running time of Apriori further deteriorates as *minsup* is decreased. It is evident that the running time of HARPP decreases with a decrease in *minsup*, whereas FP-Growth performs inversely. At *minsup* 0.001%, HARPP is about 7 times faster than FP-Growth.

Figure 10 shows the comparison of the running time of HARPP, Apriori, and FP-Growth on *Online Retail* dataset.

Figure 10

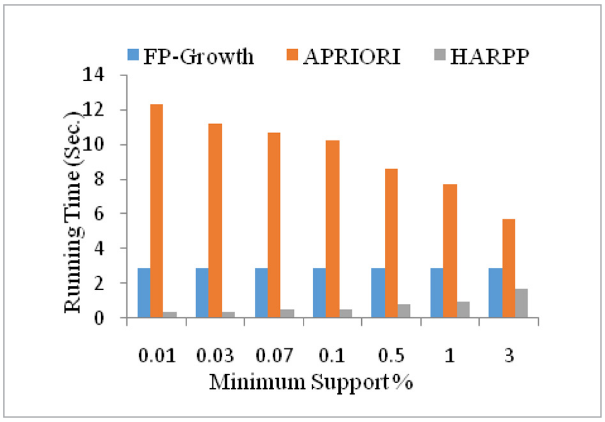
Running time on *Online Retail* dataset



The result of Apriori has not plotted again because its running time exceeded 4,500 seconds at *minsup* 0.15% and worsened when *minsup* is decreased further. This is due to the fact that as *minsup* decreases, more and more candidate itemsets are generated, which are then processed to check if they are frequent. Performance of HARPP is extensively better than FP-Growth at lower *minsup*. At *minsup* 0.003%, the running time of FP-Growth begins to grow higher than that of HARPP by more than two orders of magnitude. It is evident that the running time of HARPP decreases with the decrease in *minsup*, FP-Growth performs inversely. Figure 11 shows the performance on *Skin* dataset.

Figure 11

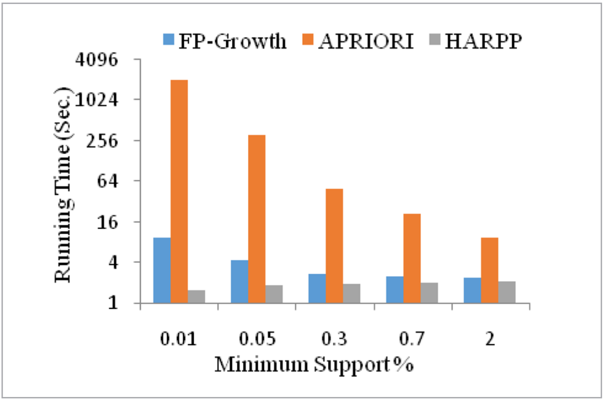
Running time on *Skin* dataset



The efficiency of HARPP is considerably better than the others. At *minsup* 0.01%, HARPP is 7 times faster than FP-Growth and faster than Apriori by a factor of 30.

In Figure 12, HARPP succeeded to achieve the smallest running times at all *minsup* thresholds on *Extended Bakery* dataset on a logarithmic scale. At *minsup* 0.01%, HARPP becomes 3 orders of magnitude faster than Apriori, and about 6 times faster than FP-Growth.

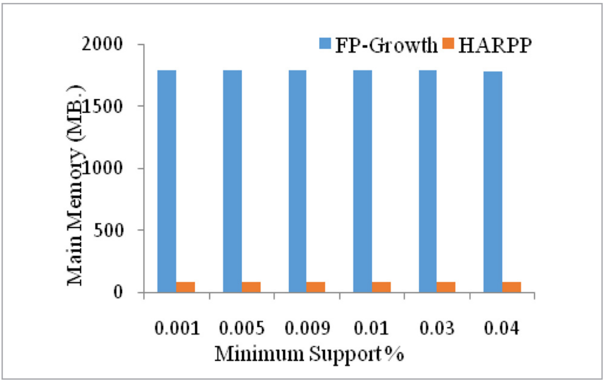
Figure 12
Running time on *Extended Bakery* dataset



6.3. Comparison of Memory Usage

Figures 13-16 show the peak memory consumption of HARPP, Apriori, and FP-Growth on four real datasets. In Figure 13, a significant difference in memory requirements is evident on *PowerC* dataset. As the dataset is large and sparse in nature, FP-Growth takes about 21 times more memory. HARPP gets least memory due to its distinguishing characteristic of not storing the entire dataset in main memory.

Figure 13
Memory consumption on *PowerC* data set



In Figure 14, on *Online Retail* dataset, FP-Growth takes 11 times more memory than taken by HARPP.

In Figure 15, on a dense dataset, HARPP uses the least memory for all *minsup* thresholds. The density of data sets means that there exist a number of itemsets in the transactions and numerous itemsets have *support* higher than *minsup* residing in main memory [6, 28].

In Figure 16, memory consumption for *Extended Bakery* data set is presented.

Figure 14
Memory consumption on *Online Retail* data set

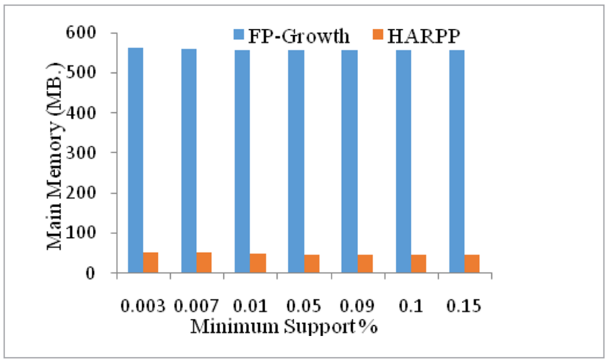


Figure 15
Memory consumption on *Skin* data set

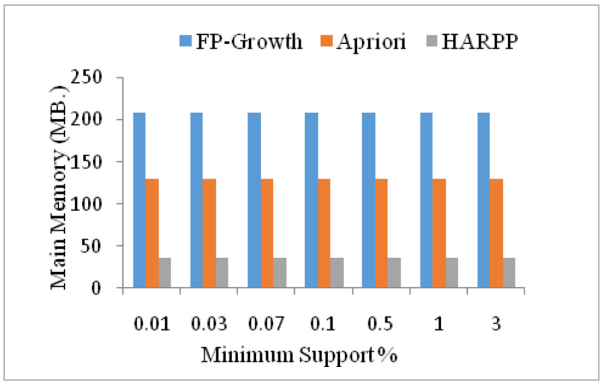
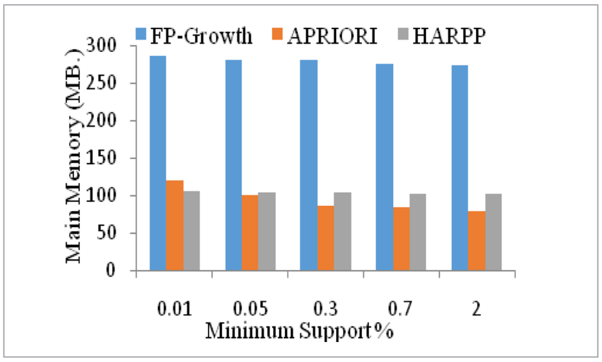


Figure 16
Memory consumption on *Extended Bakery* data set

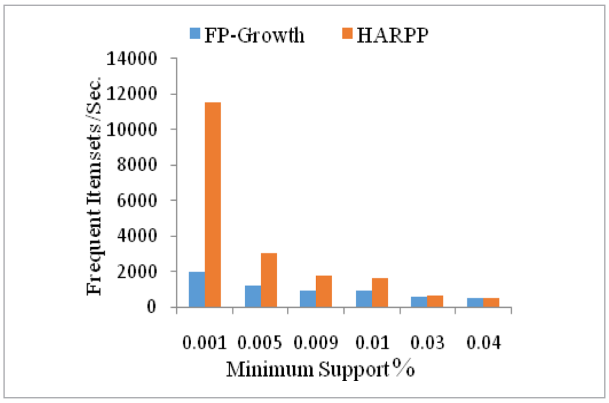


FP-Growth takes the lead as its FP-tree structure gets large. Apriori gets less memory at higher *minsup* but gradually gains larger memory at lower *minsup* due to massive production of candidate itemsets. HARPP gets the least memory at lower *minsup* values.

6.4. Comparison of Agility

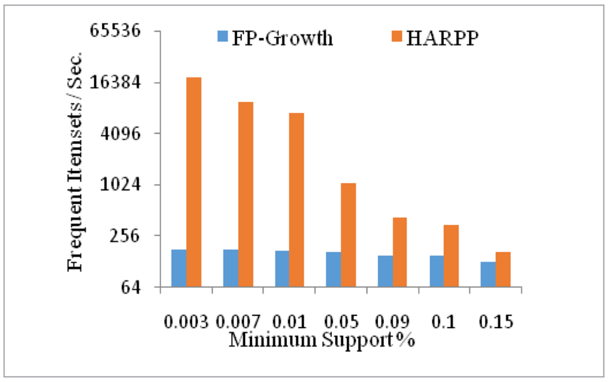
In Figures 17-20, comparison of *agility* of HARPP, Apriori, and FP-Growth is presented. In Figure 17, HARPP is 6 times more agile than FP-Growth on *PowerC* dataset.

Figure 17
Agility Comparison on *PowerC* data set



In Figure 18, FP-Growth shows nearly consistent behavior at all *minsup* thresholds, which means that variation in *minsup* does not affect the speed at which frequent itemsets are generated. But the *agility* of HARPP is extraordinary. At *minsup* 0.003%, the *agility* of HARPP is more than two orders of magnitude higher than FP-Growth.

Figure 18
Agility Comparison on *Online Retail* data set



In Figure 19, HARPP has proven its unprecedented *agility* on *Skin* dataset. It is about 7 times more agile than FP-Growth at 0.01% *minsup*. This is due to its inherent characteristic, which states that more and more *Sub-Sets* become frequent sooner at lower *minsup*.

In Figure 20, HARPP is six times more agile than FP-Growth.

Figure 19
Agility Comparison on *Skin* data set

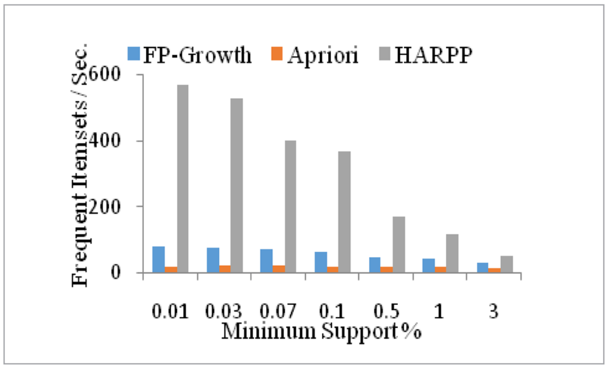
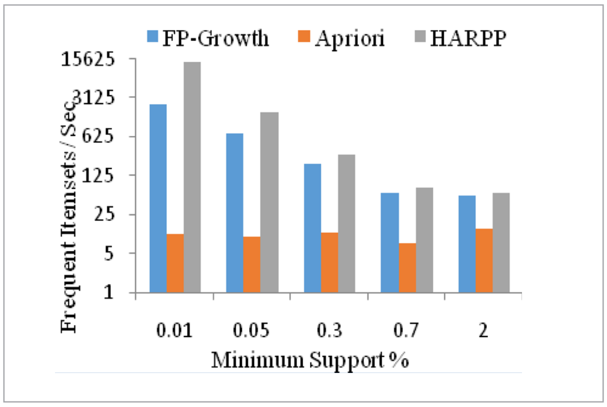


Figure 20
Agility Comparison on *Extended Bakery* data set

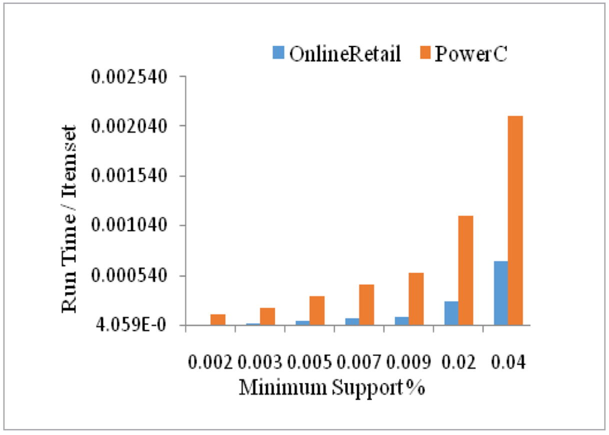


6.5. Scalability of HARPP

Figure 21 shows the runtime per frequent itemset of HARPP. It indicates as the *minsup* goes down, the runtime per frequent itemset decreases dramatically for *PowerC* and *Online Retail* datasets.

It shows that HARPP has good scalability with the reduction of *minsup*. Though the frequent itemsets

Figure 21
Run Time of the HARPP per itemset versus *minsup*



emerge in an exponential manner, the running time of HARPP raises in a much more conservative way.

6.6. Discussion of Results

The factor which affects the running time of a frequent itemset mining algorithm most is *minsup* [34]. In this study, it has been noted that the performance of HARPP is better than Apriori and FP-Growth at lower *minsup* due to the following reasons.

1 HARPP's intrinsic way of creating *Sub-Sets* is strictly independent of *minsup*. In fact, the number of *Sub-Set*s of currently read transaction *T* remains constant regardless of the value of *minsup*. *Sub-Set*s become frequent early at lower *minsup*, thereby improving efficiency. In contrast, *candidate generation* and *test* mechanism by Apriori, and construction of FP- tree and conditional FP-trees by FP-Growth are strictly dependent on *minsup*.

2 Dual containment check such as checking the existence of each transaction, *T* at Step (2) and its *Sub-Sets* at Step (5) within *F* prohibits their inclusion again into *Dict* if they have become frequent already. This prohibition reduces the running time of the HARPP since numerous overlapping transactions and their *Sub-Set*s arrive every now and then. Therefore the running time of HARPP is significantly reduced at lower *minsup*. HARPP performs better due to its efficient operations (taking constant running time) such as checking containment and insertion of itemsets in both *dictionary* and *set* data structures. On the other hand, the efficiency of FP-Growth decreases for sparse datasets, because there are small repeated patterns. FP-tree built is bigger in size and plenty of time is taken by the algorithm to build and traverse the conditional FP-trees.

3 Due to their inherent characteristics, both Apriori and FP-Growth along with its successors can only perform efficiently when the dataset resides entirely in main memory [6]. There are no repeated patterns in real sparse datasets, therefore FP-Growth builds enormous conditional FP-trees and the FP-tree built is bigger in size. This is why its memory requirement is high. In contrary, HARPP memory consumption is minimal because it does not load the entire dataset in memory. In each iteration, HARPP reads a transaction and before reading next transaction, objects residing in memory are *P* containing the power set of currently read transaction, *Dict* storing *Sub-Set*s, which could not become frequent yet, and *F*.

Applying HARPP on large datasets at various *minsup* thresholds has proven its scalability as shown in Figure 21.

7. Conclusions

Modern algorithms are mostly evaluated on dense datasets but lack adequate investigation on sparse datasets. Performance of these algorithms on sparse datasets is below par as their running times increase rapidly at lower minimum *support* thresholds. Interestingly, the difference in their running times on sparse datasets is almost negligible except for a few cases. This paper presents HARPP, a novel algorithm that mines frequent itemsets efficiently. HARPP performs exceptionally well on lower minimum *support* thresholds. HARPP does not store the database in memory and finds frequent itemsets in a single pass over the database. Since HARPP achieves better running time and unparalleled *agility* at lower minimum *support*, it can be a preferable option for recommendation systems to discover the long-tailed itemsets.

Since HARPP considers every transaction as a set of items for creating its power set, it has to store 2^N *Sub-Sets*, where *N* is the size (number of items) of a transaction. Therefore as future extensions of this work, efforts will be carried out to employ HARPP to perform efficiently on datasets having quite a long average transaction length. Nevertheless, *N* is negligible as compared to |*DB*| (number of items in a dataset), thus makes HARPP a viable solution to mine frequent itemsets. HARPP will also be employed for solving other problems such as mining closed frequent itemsets [38, 60], maximal frequent itemsets [9, 14, 59], high utility itemset mining [37], top-rank-k frequent patterns [18], and frequent weighted itemset mining [13]. Furthermore, the HARPP can be extended for mining data streams [40]. Finally, due to the escalating importance of big data, HARPP would be implemented in a parallel / distributed fashion for discovering frequent itemsets [5, 51].

Acknowledgment

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

1. Agrawal, R., Imieliński, T., Swami, A. Mining Association Rules Between Sets of Items in Large Databases. Proceedings of ACM SIGMOD Conference on Management of Data, 1993, 207-216. https://doi.org/10.1145/170036.170072
2. Agrawal, R., Srikant, R. Fast Algorithms for Mining Association Rules. Proceedings of International Conference on Very Large Data Bases, 1994, 487-499.
3. Ali, Z., Khusro, S., Ullah, I. A Hybrid Book Recommender System Based on Table of Contents (ToC) and Association Rule Mining. Proceedings of 10th International Conference on Informatics and Systems, 2016, 68-74. https://doi.org/10.1145/2908446.2908481
4. Anand, R., Jeffrey, D. U. Mining of Massive Datasets. Cambridge, UK, Cambridge University Press, 2014. https://doi.org/10.1017/CBO9781139924801
5. Apiletti, D., Baralis, E., Cerquitelli, T., Garza, P., Pulvirenti, F., Michiardi, P. A Parallel MapReduce Algorithm to Efficiently Support Itemset Mining on High Dimensional Data. Big Data Research, 10, 2017, 53-69. https://doi.org/10.1016/j.bdr.2017.10.004
6. Apiletti, D., Baralis, E., Cerquitelli, T., Garza, P., Pulvirenti, F., Venturini, L. Frequent Itemsets Mining for Big Data: A Comparative Analysis. Big Data Research, 2017, 9, 67-83. https://doi.org/10.1016/j.bdr.2017.06.006
7. Asha, T., Natarajan, S., Murthy, K. B. Associative Classification in the Prediction of Tuberculosis. Proceedings of International Conference and Workshop on Emerging Trends in Technology, Mumbai, Maharashtra, India, 2011, 1327-1330. https://doi.org/10.1145/1980022.1980315
8. Barabási, A. L. Scale-free Networks: A Decade and Beyond. Science, 2009, 325, 412-413. https://doi.org/10.1126/science.1173299
9. Bayardo, R. J. Efficiently Mining Long Patterns from Databases. Proceedings of ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, 1998, 85-93. https://doi.org/10.1145/276305.276313
10. Blake, C. L., Merz, C. J. UCI Repository of Machine Learning Databases. University of California Irvine, 1998. http://archive.ics.uci.edu/ml/
11. Brauckhoff, D., Dimitropoulos, X., Wagner, A., Salamatian, K. Anomaly Extraction in Backbone Networks Using Association Rules. Proceedings of ACM SIGCOMM Internet Measurement Conference, 2009, 28-34. https://doi.org/10.1109/TNET.2012.2187306
12. Brin, S., Motwani, R., Ullman, J. D., Tsur, S. Dynamic Itemset Counting and Implication Rules for Market Basket Data. Proceedings of ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, USA, 1997, 255-264. https://doi.org/10.1145/253262.253325
13. Bui, H., Vo, B., Nguyen, H., Nguyen-Hoang, T. A., Hong, T. P. A Weighted N-list-based Method for Mining Frequent Weighted Itemsets. Expert Systems with Applications, 2018, 96, 388-405. https://doi.org/10.1016/j.eswa.2017.10.039
14. Burdick, D., Calimlim, M., Flannick, J., Gehrke, J., Yiu, T. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. Proceedings of the 17th IEEE International Conference on Data Engineering, 2001, 443-452.
15. Chin, C.Y., Weng, M. Y., Lin, T. C., Cheng, S. Y., Yang, Y. H. K., Tseng, V. S. Mining Disease Risk Patterns From Nationwide Clinical Databases for the Assessment of Early Rheumatoid Arthritis Risk. PLoS One, 2015, 1-20. https://doi.org/10.1371/journal.pone.0122508

16. Dagenais, B. A Few Data Mining Algorithms in Pure Python. https://github.com/bartdag/pymining

17. Dekhtyar, A., Verburg, J. ExtendedBakery Datasets. https://wiki.csc.calpoly.edu/datasets/wiki/ExtendedBakery

18. Deng, Z. H. Fast Mining Top-Rank-k Frequent Patterns by Using Node-lists. Expert Systems with Applications, 2014, 41(4 PART 2), 1763-1768. https://doi.org/10.1016/j.eswa.2013.08.075

19. Deng, Z. H., Lv, S. L. Fast Mining Frequent Itemsets Using Nodesets. Expert Systems with Applications, 2014, 41(10), 4505-4512. https://doi.org/10.1016/j.eswa.2014.01.025

20. Deng, Z. H., Lv. S. L. PrePost+: An Efficient N-lists-based Algorithm for Mining Frequent Itemsets via Children-Parent Equivalence Pruning. Expert Systems with Applications, 2015, 42(13), 5424-5432. https://doi.org/10.1016/j.eswa.2015.03.004

21. Deng, Z., Wang, Z. A New Fast Vertical Method for Mining Frequent Patterns. International Journal of Computational Intelligence Systems, 2010, 3(6), 733-744. https://doi.org/10.1080/18756891.2010.9727736

22. Deng, Z., Wang, Z., Jiang, J. A New Algorithm for Fast Mining Frequent Itemsets using N-lists. Science China Information Sciences, 2012, 55(9), 2008-2030. https://doi.org/10.1007/s11432-012-4638-z

23. Duwairi, R., Ammari, H. An Enhanced CBAR Algorithm for Improving Recommendation Systems Accuracy. Simulation Modelling Practice and Theory, 2016, 60, 54-68. https://doi.org/10.1016/j.simpat.2015.10.001

24. Exarchos, T. P., Papaloukas, C., Fotiadis, D. I., Michalis, L. K. An Aassociation Rule Mining-based Methodology for Automated Detection of Ischemic ECG beats. IEEE Transactions on Biomedical Engineering, 2006, 53(8), 1531-1540. https://doi.org/10.1109/TBME.2006.873753

25. Fournier-Viger, P., Lin, C. W., Gomariz, A., Soltani. A., Deng, Z., Lam, H. T. The SPMF Open-Source Data Mining Library Version 2. Proceedings of European Conference on Principles of Data Mining and Knowledge Discovery, 2016, 36-40. https://doi.org/10.1007/978-3-319-46131-1_8

26. Garcia-Molina, H., Ullman, J. D., Widom, J. Database Systems: The Complete Book. 2nd Ed. New Jersey: Pearson; 2009. https://doi.org/10.1145/253262.253287

27. Ghafoor, Y., Huang, Y. P., Liu, S. I. An Intelligent Approach to Discovering Common Symptoms Among Depressed Patients. Soft Computing, 2015, 19(4), 819-827. https://doi.org/10.1007/s00500-014-1408-4

28. Goethals, B. Memory Issues in Frequent Itemset Mining. Proceedings of ACM Symposium on Applied Computing, Nicosia, Cyprus, 2004, 530-534. https://doi.org/10.1145/967900.968012

29. Gopalan, R. P., Sucahyo, Y. G. High Performance Frequent Patterns Extraction Using Compressed FP-tree. Proceedings of SIAM International Workshop on High Performance and Distributed Mining, Orlando, Florida, USA, 2004.

30. Grahne, G., Lakshmanan, L. V. S., Wang, X. Efficient Mining of Constrained Correlated Sets. Proceedings of 16th International Conference on Data Engineering, San Diego, CA, USA, 2000, 512-521. https://doi.org/10.1109/ICDE.2000.839450

31. Han, J., Cheng, H., Xin, D., Yan, X. Frequent Pattern Mining: Current Status and Future Directions. Data Mining and Knowledge Discovery, 2007, 15(1), 55-86. https://doi.org/10.1007/s10618-006-0059-1

32. Han, J., Pei, J., Yin, Y. Mining Frequent Patterns Without Candidate Generation. Proceedings of ACM SIGMOD International Conference on Management of data, Dallas, Texas, USA, 2000, 1-12. https://doi.org/10.1145/342009.335372

33. Harrington, P. Machine Learning in Action. Manning Publications 2012. https://doi.org/10.1007/s10994-011-5249-4

34. Heaton, J. Comparing Dataset Characteristics that Favor the Apriori, Eclat or FP-Growth Frequent Itemset Mining Algorithms. Proceedings of IEEE Southeast Conference, 2016, 1-7. https://doi.org/10.1109/SECON.2016.7506659

35. Jeeva, S. C., Rajsingh, E. B. Intelligent Phishing Url Detection Using Association Rule Mining. Human-centric Computing and Information Sciences, 2016, 6(1), 10. https://doi.org/10.1186/s13673-016-0064-3

36. Klemettinen, M., Heikki, M., Ronkainen, P., Toivonen, H., Verkamo, I. Finding Interesting Rules From Large Sets of Discovered Association Rules. Proceedings of 3rd International Conference on Information and Knowledge Management, Gaithersburg, MD, USA, 1994, 401-407. https://doi.org/10.1145/191246.191314

37. Krishnamoorthy, S. HMiner: Efficiently Mining High Utility Itemsets. Expert Systems with Applications, 2017, 90, 168-183. https://doi.org/10.1016/j.eswa.2017.08.028

38. Lee. A. J. T., Wang, C. S., Weng, W. Y., Chen, Y. A., Wu, H. W. An Efficient Algorithm for Mining Closed Inter-Transaction Itemsets. Data and Knowledge Engi-

neering, 2008, 66(1), 68-91. https://doi.org/10.1016/j.datak.2008.02.001

39. Lent, B., Swami, A., Widom, J. Clustering Association Rules. Proceedings of 13th International Conference on Data Engineering, Birmingham, UK, 1997, 220-231. https://doi.org/10.1109/ICDE.1997.581756

40. Li, X., Deng, Z. H. Mining Frequent Itemsets from Network Flows for Monitoring Network. Expert Systems with Applications, 2010, 37(12), 8850-8860. https://doi.org/10.1016/j.eswa.2010.06.012

41. Makhtar, M., Harun, N. A., Aziz, A. A., Zakaria, Z. A., Abdullah, F. S., Jusoh, J. A. An Association Rule Mining Approach in Predicting Flood Areas. Proceedings of International Conference on Soft Computing and Data Mining, 2016, 437-446. https://doi.org/10.1007/978-3-319-51281-5

42. Mobasher, B., Dai, H., Luo, T., Nakagawa, M. Effective Personalization Based on Association Rule Discovery from Web Usage Data. Proceedings of 3rd ACM International Workshop on Web Information and Data Management, Atlanta, Georgia, 2001, 9-15. https://doi.org/10.1145/502932.502935

43. Nahar, J., Imam, T., Tickle, K. S., Chen, Y. P. P. Association Rule Mining to Detect Factors Which Contribute to Heart Disease in Males and Females. Expert Systems with Applications, 2013, 40(4), 1086-1093. https://doi.org/10.1016/j.eswa.2012.08.028

44. Najafabadi, M. K., Mahrin, M. N., Chuprat, S., Sarkan, H. M. Improving the Accuracy of Collaborative Filtering Recommendations Using Clustering and Association Rules Mining on Implicit Data. Computers and Human Behavior, 2017, 67(2), 113-128. https://doi.org/10.1016/j.chb.2016.11.010

45. Ng, R. T., Lakshmanan, L. V. S., Han, J., Pang, A. Exploratory Mining and Pruning Optimizations of Constrained Associations Rules. Proceedings of ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, 1998, 13-24. https://doi.org/10.1145/276305.276307

46. Ozel, S. A., Guvenir, H. A. An Algorithm for Mining Association Rules Using Perfect Hashing and Database Pruning. Proceedings of 10th Turkish Symposium on Artificial Intelligence and Neural Networks, 2001, 257-264.

47. Park, J., Chen, M., Yu, P. An Effective Hash-based Algorithm for Mining Association Rules. Proceedings of ACM SIGMOD International Conference on Management of Data, SanJose, California, USA, 1995, 175-186. https://doi.org/10.1145/568271.223813

48. Pera, M. S., Ng, Y. Analyzing Book-Related Features to Recommend Books for Emergent Readers. Proceedings of 26th ACM Conference on Hypertext and Social Media, Guzelyurt, Northern Cyprus, 2015, 221-230. https://doi.org/10.1145/2700171.2791037

49. Sarawagi, S., Thomas, S., Agrawal, R. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. Proceedings of ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, 1998, 343-354. https://doi.org/10.1145/276304.276335

50. Savasere, A., Omiecinski, E. R., Navathe, S. An Efficient Algorithm for Mining Association Rules in Large Databases. Proceedings of International Conference on Very Large Data Bases, 1995, 432-444. https://doi.org/10.1109/WKDD.2008.33

51. Sethi, K. K., Ramesh, D. HFIM: A Spark-based Hybrid Frequent Itemset Mining Algorithm for Big Data Processing. Journal of Supercomputing, 2017, 73(8), 3652-3668. https://doi.org/10.1007/s11227-017-1963-4

52. Shenoy, P., Haritsa, J. R., Sundarshan, S., Bhalotia, G., Bawa, M., Shah, D. Turbo-Charging Vertical Mining of Large Databases. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA, 2000, 22-33. https://doi.org/10.1145/342009.335376

53. Song, W., Yang, B., Xu, Z. Index-BitTableFI: An Improved Algorithm for Mining Frequent Itemsets. Knowledge-Based Systems, 2008, 21(6), 507-513. https://doi.org/10.1016/j.knosys.2008.03.011

54. Srikant, R., Vu, Q., Agrawal, R. Mining Association Rules with Item Constraints. Proceedings of 3rd International Conference on Knowledge Discovery and Data Mining, Newport Beach, USA, 1997, 67-73. https://doi.org/10.1016/j.ijar.2004.11.006

55. Toivonen, H. Sampling Large Databases for Association Rules. Proceedings of 22nd International Conference on Very Large Data Bases, Mumbai (Bombay), India, 1996, 134-145.

56. Tsay, Y. J., Chiang, J. Y. CBAR: An Efficient Method for Mining Association Rules. Knowledge-Based Systems, 2005, 18(2-3), 99-105. https://doi.org/10.1016/j.knosys.2004.04.010

57. Vathsala, H., Koolagudi, S. G. Prediction model for peninsular Indian Summer Monsoon Rainfall using Data Mining and Statistical Approaches. Computers and Geosciences, 2017, 98, 55-63. https://doi.org/10.1016/j.cageo.2016.10.003

58. Vo, B., Le, T., Coenen, F., Hong, T. P. Mining Frequent Itemsets Using the N-list and Subsume Concepts. International Journal of Machine Learning and Cybernetics, 2016, 7(2), 253-265. https://doi.org/10.1007/s13042-014-0252-2

59. Vo, B., Pham, S., Le, T., Deng, Z. H. A Novel Approach for Mining Maximal Frequent Patterns. Expert Systems with Applications. 2017, 73, 178-186. https://doi.org/10.1016/j.eswa.2016.12.023

60. Wang, J., Han, J., Pei, J. Closet+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. Proceedings of Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, 236-245. https://doi.org/10.1145/956750.956779

61. Yagci, A. M., Aytekin, T., Gurgen, F. S. Scalable and Adaptive Collaborative Filtering by Mining Frequent Item Co-occurrences in a User Feedback Stream. Engineering Applications of Artificial Intelligence, 2017, 58, 171-184. https://doi.org/10.1016/j.engappai.2016.10.011

62. Yoshida, K., Shomura, Y., Watanabe, Y. Visualizing Network Status. Proceedings of 6th International Conference on Machine Learning and Cybernetics, 2007, 4, 2094-2099. https://doi.org/10.1109/ICMLC.2007.4370490

63. Zaki, M. J. Scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering, 2000, 372-390. https://doi.org/10.1109/69.846291

64. Zaki, M. J., Gouda, K. Fast Vertical Mining Using Diffsets. Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, 326-335. https://doi.org/10.1145/956755.956788

65. Zhengbing, H., Zhitang, L., Jumgi, W. A Novel Network Intrusion Detection System (NIDS) Based on Signatures Search of Data Mining. Proceedings of 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia and Workshop, Adelaide, Australia, 2008, 10-16. https://doi.org/10.1109/WKDD.2008.48