IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 4 November 2022, accepted 17 November 2022, date of publication 23 November 2022,
date of current version 30 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3224217

RESEARCH ARTICLE

Mining High Utility Time Interval Sequences Using MapReduce Approach: Multiple Utility Framework

SUMALATHA SALETI[1], T. JAYA LAKSHMI[1], (Member, IEEE), AND MOHD WAZIH AHMAD[2]
[1]Department of Computer Science and Engineering, SRM University AP, Guntur, Amaravati 522502, India
[2]Department of Computer Science and Engineering, Adama Science and Technology University, Adama 1888, Ethiopia

Corresponding author: Mohd Wazih Ahmad (wazih.ahmad@astu.edu.et)

ABSTRACT Mining high utility sequential patterns is observed to be a significant research in data mining. Several methods mine the sequential patterns while taking utility values into consideration. The patterns of this type can determine the order in which items were purchased, but not the time interval between them. The time interval among items is important for predicting the most useful real-world circumstances, including retail market basket data analysis, stock market fluctuations, DNA sequence analysis, and so on. There are a very few algorithms for mining sequential patterns those consider both the utility and time interval. However, they assume the same threshold for each item, maintaining the same unit profit. Moreover, with the rapid growth in data, the traditional algorithms cannot handle the big data and are not scalable. To handle this problem, we propose a distributed three phase MapReduce framework that considers multiple utilities and suitable for handling big data. The time constraints are pushed into the algorithm instead of pre-defined intervals. Also, the proposed upper bound minimizes the number of candidate patterns during the mining process. The approach has been tested and the experimental results show its efficiency in terms of run time, memory utilization, and scalability.

INDEX TERMS Data mining, MapReduce framework, multiple utility thresholds, sequential pattern mining, time interval patterns.

I. INTRODUCTION

Sequential pattern mining (SPM) [1], [2], [3], [4], [5], [6], [7], [8] is a significant research theme in data mining. The primary goal of SPM is to identify frequent sequences that are defined by a minimum support level that is planned by the user. In particular, a customer who purchased ''Television'' would like to return to the store and purchase ''Speakers''. Market analysts can use this information to develop novel marketing tactics such as product cross-selling and advertising activities. The standard SPM techniques employ a frequency-based framework and are regarded as uninformative because they can't mine highly profitable sequences. As a result, high utility sequential pattern mining (HUSPM) [9] has been offered as a solution to this problem, it mines high utility sequences while taking into consideration both the item's profit and quantity.

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Ming Chen.

Although previous HUSPM algorithms [9], [10], [11], [12], [13], [14], [15] produce very profitable sequences, they are unable to estimate the time gap between consumers' subsequent visits to the store. Therefore, high utility time interval sequential pattern mining (HUTISP) [16] was developed to take time periods into account. Its primary objective is to find the patterns that have the time interval between each item. Observe a store that offers groceries such as soaps, cereals, books, and ice creams. Assuming these items in a database, the goal is to determine the time interval between purchases of specific items which are being sold. As a solution, the store keeper may keep track of the quantity of consumed items by time period. For instance, consider an output sequence pattern with time intervals denoted as $\langle x, 4, y, 6, z\rangle$. It signifies that a person who bought item x also bought item y after 4 months and returned to the store after 6 months to purchase item z. It mines extremely profitable sequences as well as the time interval between them. In spite of this, it uses the single

unique threshold for each item in the input database, implying that every item has the same unit profit. This is unsatisfactory because every single item in real world situations is unique and should not be considered equally. For instance, sales of washing machine will make more profits than sales of detergent. The problem of finding sequential patterns considering multiple utility thresholds was first addressed in [17] and later extended to [18]. The authors [18] designed Lexicographic sequence tree and utility array. The tree structure represents the possible HUSPs as nodes and the extension of each node is done using I-concatenation and S-concatenation mechanisms. The former denotes the itemset extension, whereas later denotes the sequence extension. Utility array allows to find the utility values of each node in the tree. However, the authors in [17] and [18] does not deal with the time intervals which generates more significant patterns.

Due to the ever growing database sizes, many researchers of data mining revised the conventional mining algorithms and formulated the distributed algorithms to handle the big data more efficiently. The most efficient big data framework that helps in designing the distributed algorithms is MapReduce [19]. In 2008, Google developed MapReduce [19] distributed programming framework. It can handle the processing of big data by distributing the work into two parallel processes, namely, Mapper and Reducer. Mapper is a process that partitions the input data into multiple chunks and processes them in parallel. The processed output is sent to the reducer for further processing which leads to the final output. The output of the parallel program is always stored inside a distributed storage called Hadoop distributed file system (HDFS). The work flow of MapReduce framework is given in Fig. 1.

HUTISP [16] is a centralized algorithm that is ineffective for managing large amounts of data. In light of this, DHUTISP [20] has been introduced, it is a distributed MapReduce algorithm. Recently, a MapReduce algorithm, namely, DHUTISP-MMU [21] has been proposed to handle the issue of multiple utility of items. But, it considers the predefined time interval set before generating the patterns. In consideration of this, we propose MRHUTSP-MMU in this study. It will handle the issue of setting time constraints.

The work plan of the current paper is to:

1) Investigate HUTISP mining using multiple utility thresholds.
2) Propose the distributed framework that deal with the big data.
3) Provide an upper bound that maintains the downward closure property which can efficiently prune the unpromising candidates.
4) Introduce an efficient mechanism to push the time constraints within the algorithm instead of predefined time intervals.
5) Provide an empirical analysis and comparison between the distributed and non-distributed algorithms and analyze the impact of applying time constraints.

The significant contributions of the current paper include:

1) Defined a novel sequential pattern that includes utility, time intervals and multiple utility thresholds.
2) Proposed MRHUTSP-MMU - a distributed algorithm based on MapReduce framework that mines high utility sequential patterns including time intervals and each item having its own threshold.
3) Introduced Downward closure property that can be tested on the patterns which includes multiple utility thresholds.
4) The correctness of MRHUTSP-MMU is proved theoretically.
5) Compared the distributed version with the non-distributed approach in terms of processing time, usage of memory, and scalability.

The remaining sections include the following. Section II provides a quick overview of the literature. The problem is defined in Section III. In Section IV, the proposed algorithm is described. Section V contains the experimental outcomes. Section VI discusses the conclusions.

II. LITERATURE REVIEW

A. SEQUENTIAL PATTERN MINING AND TIME INTERVAL MINING

Sequential pattern mining [1], [2], [3], [4], [5], [6], [7] is applied in a variety of data mining research disciplines. For mining sequential patterns, there are mainly two kinds of algorithms. They are Apriori [22] and Pattern growth approach [5]. In 1996, Agarwal and Srikant invented the GSP algorithm. Candidate generation and frequency calculation are the two processes of GSP. Zaki invented the SPADE method in [4], it is grounded on the vertical mining procedure. Han developed pattern growth approaches such as Freespan [23] and Prefixspan [5]. Garofalakis et.al., conceived and implemented the SPIRIT algorithm for sequential pattern mining using constraints in [2]. Researchers focused on time interval pattern mining after investigating the problem of mining sequential patterns. Chen et al. [23] presented novel algorithms for time interval mining, notably I-Apriori and I-Prefixspan. These techniques have been extended by Chen et al. by proposing FTI-Apriori and FTI-Prefixspan, which use fuzzy theory to partition temporal periods [24].

B. HIGH UTILITY ITEMSET MINING WITH MULTIPLE THRESHOLDS

The above listed algorithms are concerned with mining of frequent sequences and time interval patterns. They consider the occurrence of each item as binary and miss the number of units purchased and the profit raised by each item. In view of this, [25], [26] introduced a new framework called high utility itemset mining (HUIM) which is an improvement over frequent itemset mining [27], [28], [29], [30]. Considering non binary transactions, utility mining [25], [26], [31] was introduced, and it became an essential research theme in the data mining industry. Initially, HUIM with multiple

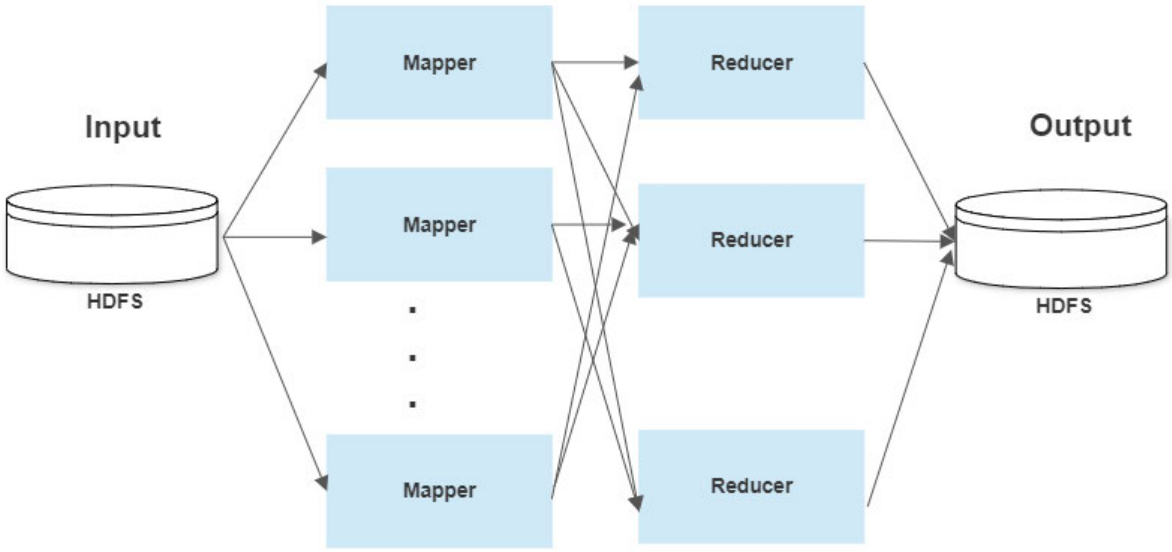

FIGURE 1. MapReduce framework.

thresholds was introduced in [32]. The authors proposed two algorithms, one is a baseline algorithm provided as an initial solution for mining HUIs using more than one threshold. The other is an improved version of the former that makes use of TID-index procedure. Using this index, utility of a candidate can be found by reading the TID-index of that candidate instead of scanning the complete database. Further, the above two algorithms are extended using estimated utility co-occurrence structure (EUCS) and a novel algorithm, namely, *HUI-MMU*$_{TE}$ [33] has been proposed. However, these are Apriori-based and follows candidate generation and level wise approach. These involve multiple database scans. In contrast to Apriori-based algorithm, the authors proposed *HIMU* and *HIMU*$_{EUCP}$ [34] algorithms. These make use of the compact tree structure. Also,the properties introduced in [34] assure the anti-monotonicity inorder to mine the patterns from MIU-tree. However, these algorithms order the items in a transaction with reference to the values of minimum utility threshold. It means that these algorithms are sensitive to specific ordering of items. MHUI algorithm proposed in [35], do not consider any ordering among the items and is superior to all the above mentioned high utility mining algorithms with multiple thresholds. The author proposed suffix minimum utility which is used in developing generic pruning strategies which are independent of item ordering based on minimum utility thresholds.

C. SEQUENTIAL PATTERN MINING INCLUDING UTILITY AND MULTIPLE THRESHOLDS

All the above mentioned algorithms are associated to HUIM with multiple utility thresholds and are unable to address the HUSP issue. A framework for HUSP mining that considers different utility threshold for each item has been introduced in [17]. From the proposed framework, the authors introduced a baseline algorithm and later it was extended by including three pruning strategies. The pruning techniques helps to reduce the upper bound there by the search space is reduced. Later, the framework proposed in [17] was extended to USPT algorithm [18]. The authors made use of compressed utility array structure and this helps in the construction of Lexicographic-sequential tree. To the best of our understanding, these are the only algorithms that discuss the issue of HUSP mining using multiple utility thresholds. Yet, they are unable to mine the time intervals.

D. HIGH UTILITY SEQUENTIAL PATTERN MINING INCLUDING TIME INTERVALS AND SINGLE THRESHOLD

Considering the time intervals, Wang et al. [16] introduced *UTMining_A* algorithm that can mine high utility sequences including the time intervals. Considering the need of processing big data, a distributed approach has been proposed in [20] which uses a single threshold for all items. The authors in [36] proposed an efficient way of imposing the time constraints while generating the sequential patterns with high utility. The above-mentioned techniques in [16], [20], and [36] on the other hand, treat each item as equally essential, considering a single minimal utility criterion. Recently, UIPrefixSpan-MMU [37] algorithm has been proposed which can handle multiple utility threshold problem. But, the algorithm assumes that the data fit into a single centralized system and not suitable to handle the current era of big data. As a result, in the current framework, we present a distributed approach for HUTSP mining with multiple utilities.

E. DIFFERENCE FROM EARLIER WORKS

The existing works in mining HUSP using multiple utility thresholds avoid the time frame amidst the items. Also, time interval HUSP mining algorithms use a single threshold for every item. Furthermore, the research in HUSP mining lacks a distributed algorithm using MapReduce framework that

TABLE 1. Utility dataset.

Sequence id	Sequence
S_1	$\langle 0, a[3]\rangle\langle 1, a[4]b[2]d[1]\rangle\langle 2, a[4]f[3]\rangle\langle 3, d[1]\rangle$
S_2	$\langle 0, e[2]\rangle\langle 1, a[2]b[6]\rangle\langle 2, d[1]\rangle\langle 3, c[2]\rangle$
S_3	$\langle 0, c[3]f[1]\rangle\langle 1, b[2]\rangle\langle 2, d[4]e[3]\rangle$
S_4	$\langle 0, a[2]\rangle\langle 1, b[6]d[1]\rangle\langle 2, a[2]b[4]\rangle\langle 3, e[5]\rangle$
S_5	$\langle 0, a[4]c[1]\rangle\langle 1, e[2]\rangle$

TABLE 2. Profit table.

Item	Profit
a	2
b	3
c	1
d	6
e	2
f	1

deal with utility, time interval sequences and multiple utility thresholds. To the best of our understanding, the current work is the initial study on proposing a MapReduce framework which can generate high utility sequences including time intervals along with a different threshold on each item.

III. PROBLEM DEFINITION

To illustrate the mining process, we consider a sample quantitative sequence dataset as given in Table 1. Let $X = \{i_1, i_2, \ldots, i_m\}$ are the allowed items in the dataset. An itemset is dentoed as $I = \{i_1, i_2, \ldots, i_q\}$, where $I \subseteq X$. If $\mid I \mid = q$, then I is referred to as a *q-itemset*. An input sequence S is denoted as $\langle (t_{1,1}, I_1), (t_{1,2}, I_2), \ldots, (t_{1,n}, I_n)\rangle$, where I_i denotes the itemset and $t_{\alpha,\beta}$ denotes the time gap between the purchase of two itemsets I_β and I_α. A quantitative sequence dataset is denoted as $D = \{S_1, S_2, S_3, \ldots, S_n\}$, where every input sequence has its unique identifier called Sequence id. Each item is allotted a profit value called its external utility, $E(i)$, and its existence in the input sequence is allotted a value called internal utility, $IU(i, I_j, S_n)$, here i is the item in I_j. For instance, $IU(a, I_1, S_1) = 3$ and $E(a) = 2$ according to Table 2.

Definition 1: The utility of each item i in I_j for a given sequence S_a is defined as $u(i, I_j, S_a) = IU(i, I_j, S_a) * E(i)$. The utility of i in S_a is the maximum utility value out of multiple itemsets in S_a where i occurs. It is denoted as $u(i, S_a)$. Let us find the utility of a in S_1, i.e. $u(a, S_1) = \max\{u(a, I_1, S_1), u(a, I_2, S_1), u(a, I_3, S_1)\} = \max\{3 * 2, 4 * 2, 4 * 2\} = 8$.

Definition 2: The utility of a pattern $P = \langle (t_{1,1}, I_1), (t_{1,2}, I_2), \ldots, (t_{1,n}, I_n)\rangle$ of length n in a sequence S_a and $P \subseteq S_a$ is denoted as $u(P, S_a)$ and derived as follows:

$$u(P, S_a) = max\{\sum_{i \in P} u(i, S_a), \forall P \in S_a\} \tag{1}$$

For example, $u(\langle (0, a)(1, a)\rangle, S_1) = \max\{3 \times 2 + 4 \times 2, 4 \times 2 + 4 \times 2\} = \max\{14, 16\} = 16$.

Definition 3: Sequence utility is equal to the sum of the item utilities in a sequence S_a, and derived as follows:

$$SU(S_a) = \sum u(i, I_j, S_a), \forall i \in S_a \tag{2}$$

TABLE 3. Multi-threshold table.

Item	Threshold
a	45%
b	28%
c	50%
d	40%
e	70%
f	40%

For example, $SU(S_1) = u(a, I_1, S_1) + u(a, I_2, S_1) + u(b, I_2, S_1) + u(d, I_2, S_1) + u(a, I_3, S_1) + u(f, I_3, S_1) + u(d, I_4, S_1) = 6 + 8 + 6 + 6 + 8 + 3 + 6 = 43$.

Definition 4: Let D denote the input data located in the Hadoop Distributed File System (HDFS). $T_1, T_2, \ldots, T_n$ denote the nonempty disjoint input partitions of D, where $D = \{T_1 \cup T_2 \cup T_3 \cup \cdots \cup T_n\}$. The input splits for the dataset shown in Table 1 are assumed to be, $T_1 = \{S_1, S_2, S_3\}$ and $T_2 = \{S_4, S_5\}$.

Definition 5: Local utility of a pattern P in a partition T_i is represented as $U_L(P, T_i)$, where

$$U_L(P, T_i) = \sum u(P, S_i), \forall S_i \in T_i \cap P \subseteq S_i \tag{3}$$

For example, $U_L(\langle (0, a)(1, e)\rangle, T_2) = u(\langle (0, a)(1, e)\rangle, S_4) + u(\langle (0, a)(1, e)\rangle, S_5) = 14 + 12 = 26$.

Definition 6: For an input partition T_i, its utility is $U(T_i)$, where

$$U(T_i) = \sum SU(S_a), \forall S_a \in T_i \tag{4}$$

For instance, $U(T_1) = SU(S_1) + SU(S_2) + SU(S_3) = 43 + 34 + 40 = 117$.

Definition 7: Global utility of a pattern P is represented as $U_G(P)$ and derived as follows:

$$U_G(P) = \sum U_L(P, T_i), \forall T_i \in D. \tag{5}$$

For example, $U_G(\langle (0, b)(1, a)\rangle) = U_L(\langle (0, b)(1, a)\rangle, T_1) + U_L(\langle (0, b)(1, a)\rangle, T_2) = 14 + 22 = 36$.

Definition 8: The utility of a given dataset D is defined as the summation of each partition utility.

To illustrate, let us find the utility of the sample dataset D, $U(D) = U(T_1) + U(T_2) = 117 + 67 = 184$.

Definition 9: For a pattern $P = \langle (t_{1,1}, I_1), (t_{1,2}, I_2), \ldots, (t_{1,n}, I_m)\rangle$, the following time constraints C_1, C_2, C_3, C_4 are defined.

Considering the time interval between two consecutive itemsets, C_1 and C_2 denote the minimum and maximum time. Similarly, considering the first and last itemsets, C_3 and C_4 denote the minimum and maximum time.

Definition 10: Multiple Minimum Utility threshold table (MMU table) is used to construct and express utility thresholds for every item, i.e. $mu(i_j)$. Table 3 shows the MMU table for the sample dataset. $mu(i_j)$ is defined as follows [32]:

$$mu(i_j) = \max\{c \times E(i_j), LMU\} \tag{6}$$

LMU is the user defined least minimum utility threshold, $E(i_j)$ is the external utility of i_j, and c is the constant to adjust mu of item.

Definition 11: The minimum utility threshold of a pattern P is expressed as $MIU(P) = \min\{mu(i) \mid i \in P\}$. For example, $MIU(\langle(0, b), (1, d)\rangle) = \min\{mu(b), mu(d)\} = \min\{28, 40\} = 28$.

Definition 12: The potential minimum utility threshold of a pattern P is denoted as $PMIU(P)$,

$$PMIU(P) = \min\{mu(i_j) \mid i_j \in P \vee i_j \in ext(P) \tag{7}$$

where, $ext(P)$ represents the possible extensions of P in the database. For instance, $PMIU(\langle(0, b), (1, d)\rangle) = \min\{\{mu(b), mu(d)\} \vee \{mu(c), mu(e)\}\} = \min\{\{28, 50\} \vee \{50, 70\}\} = 28$.

Definition 13: A pattern P is a local high utility sequential time interval pattern only if the local utility of P is not less than $MIU(P)$ and holds C_1, C_2, C_3, and C_4.

$$U_L(P, T_i) \geq MIU(P) \times U(T_i) \tag{8}$$

Definition 14: A pattern P is a global high utility sequential time interval pattern only if the global utility of P is not less than $MIU(P)$ and holds C_1, C_2, C_3, and C_4.

$$U_G(P) \geq MIU(P) \times U(D) \tag{9}$$

Definition 15: Problem Statement: Mining the HUTISPs considering more than one utility threshold is to extract the possible time interval sequential patterns those satisfy the utility threshold for a given input dataset.

Definition 16: Sequence weighted utility (SWU) of a pattern P is defined as the sum of the sequence utility of each input sequence in which P occur. It is defined as follows:

$$SWU(P) = \sum SU(S_i), \forall S_i \supseteq P \tag{10}$$

Definition 17: Given a pattern P, its upper bound in a sequence S_a is denoted as,

$$\begin{aligned} UB(P, S_a) &= u(P, S_a) + RU(P, S_a), \quad \text{if } RU(P, S_a) > 0, \\ &= 0, \quad otherwise \end{aligned} \tag{11}$$

where $RU(P, S_a)$ is the remaining utility of P in S_a and it is the summation of item utilities that appear after the last item of P in S_a. For example, $RU(\langle(0, a)(1, b)\rangle, S_1) = u(d, I_2, S_1) + u(a, I_3, S_1) + u(f, I_3, S_1) + u(d, I_4, S_1) = 6 + 8+3+6 = 23$. $UB(\langle(0, a)(1, b)\rangle, S_1) = u(\langle(0, a)(1, b)\rangle, S_1) + RU(\langle(0, a)(1, b)\rangle, S_1) = 12+23 = 35$. Multiple occurrences of a pattern consider the maximum value as its upper bound.

Definition 18: For a pattern P, its upper bound in an input partition T_i is,

$$UB(P, T_i) = \sum_{a=1}^{x} UB(P, S_a) \tag{12}$$

where x is the count of sequences present in T_i in which P appear. For example, $UB(\langle(0, a)(1, b)\rangle, T_1) = UB(\langle(0, a)(1, b)\rangle, S_1) + UB(\langle(0, a)(1, b)\rangle, S_2) + UB(\langle(0, a)(1, b)\rangle, S_3) = 35 + 0 + 0 = 35$.

Property 1 (Downward closure property): Given a quantitative sequence dataset D, and sequences A and B, where B is a super-sequence to A, then $UB(A, D) \geq UB(B, D)$.

Algorithm 1 First Phase

Input:
Time interval quantitative sequence dataset
External utility table
MMU-Table
Output:
Promising item and its global utility.
function Mapper
1: Scan the input sequence and calculate LU of each item i and SU of the sequence.
2: Emit the item i and its local utility LU along with the sequence utility SU
3: **end function**
4: **function** Reducer
5: Let item i be the key and LU, SU be the values received from the mapper.
6: Initialize $GU \leftarrow 0$, $SWU \leftarrow 0$
7: **for** each LU, SU pair **do**
8: $GU \leftarrow GU + LU$
9: $SWU \leftarrow SWU + SU$
10: **end for**
11: **if** $SWU \geq MIU(item)$ **then**
12: Emit the item i and global utility GU
13: **end if**
14: **end function**

Proof: According to Definitions 17, and 18, the $UB(A, D) = \sum_{a=1}^{x} UB(A, S_a)$ and $UB(B, D) = \sum_{a=1}^{y} UB(B, S_a)$, where x and y denotes the number of input sequences in which A and B occur respectively. As A is a sub-sequence to B, all the time $x \geq y$. Therefore, $UB(A, D) \geq UB(B, D)$.

Definition 19: Given a sequence $S = \langle(t_{1,1}, I_1), (t_{1,2}, I_2), \ldots, (t_{1,n}, I_n)\rangle$, a sequence $S' = \langle(t'_{1,1}, I'_1), (t'_{1,2}, I'_2), \ldots, (t'_{1,m}, I'_m)\rangle$, where $m \leq n$ is called its prefix iff (1) $t'_{1,i} = t_{1,i}$ and $I'_i = I_i$ for $i \leq m - 1$ (2) $t'_{1,m} = t_{1,m}$ and $I'_m \subseteq I_m$. Similarly, postfix of S with respect to prefix S' is given as $S'' = \langle(t''_{1,m}, I''_m, t''_{1,m+1}, I''_{m+1}, \ldots, t''_{1,n}, I''_n)\rangle$ iff (1) $t''_{1,i} = t_{1,i}$ and $I''_i = I_i$ for $i > m$ (2) $t''_{1,m} = t'_{1,m}$, and $I''_m \subseteq I'_m$ (3) $t_{1,n} = t''_{1,n}$ and $I_n = I''_n$.

Definition 20: The projected database of a sequence S over the database D is expressed as $D\ |_S$ and it is defined as the collection of all the postfix of S in each input sequence of D.

IV. MRHUTSP-MMU

In this section, the proposed system is outlined by listing the details of each MapReduce phase. The major goal of the proposed algorithm is to impose time constraints and consider different utility thresholds while keeping the downward closure property. Using the MapReduce framework, the algorithm is designed in three phases. First of all, the proposed algorithm finds the global utility of every item in its first phase and finds the promising items. The detailed approach for the first phase is stated in Algorithm 1. Inorder

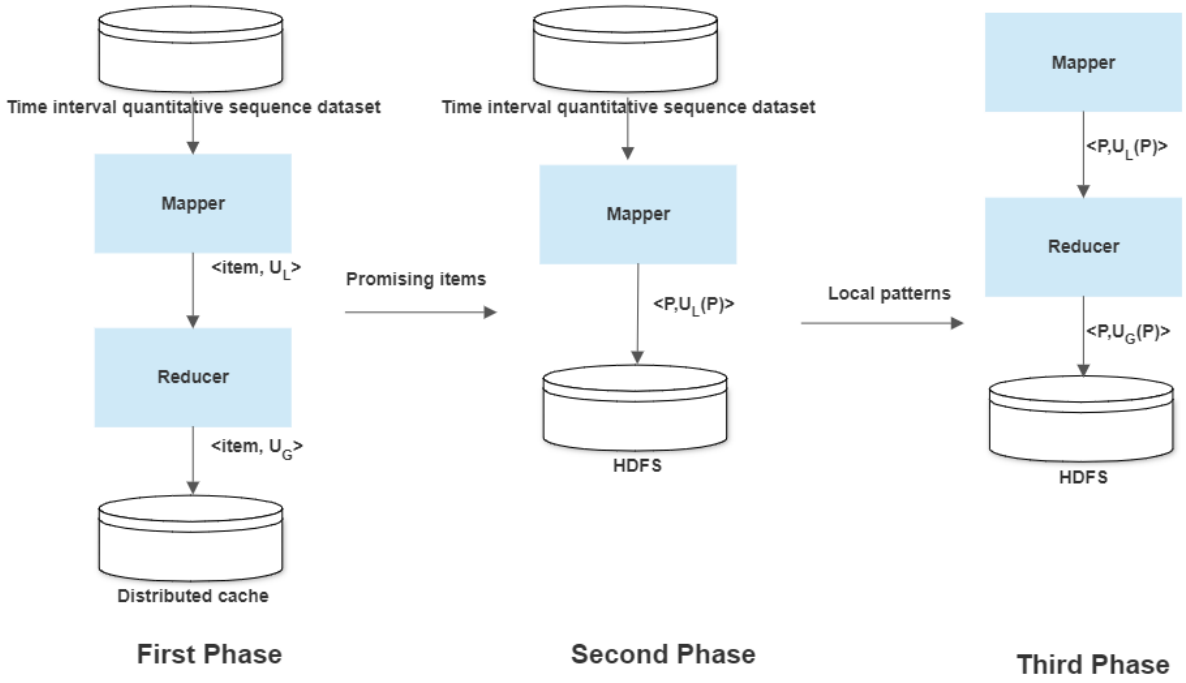

FIGURE 2. Workflow of MRHUTSP-MMU.

to generate all the output sequences, we invoke the second phase. In other words, Algorithm 2 generates entire output sequences that are local to each partition. We conduct the third MapReduce phase to generate the patterns which are global. This is detailed in Algorithm 4. The workflow of MRHUTSP-MMU is given in Fig. 2.

A. FIRST PHASE OF MRHUTSP-MMU

The first phase of MRHUTSP-MMU is explained in Algorithm 1. It includes a mapper and a reducer. Mapper takes time interval quantitative sequence dataset, external utility associated with every item, and MMU-Table as input. Initially, mapper scans the database and extracts the items local utility and sequence utility of each input sequence (Line 1) (refer to Definition 1 and Definition 3). The local and sequence utilities of item are emitted by the mapper (Line 2). This in turn will be the input to the reducer. For each item, reducer is responsible to calculate item's global utility and sequence weighted utility. From Definition 7, global utility of an item is the sum of its local utility (Line 8). Similarly, from Definition 16, *SWU* refers to sum of all the sequence utilities (Line 9). Finally, the items whose sequence weighted utility satisfies the MIU threshold will be emitted as the output (Lines 11-13). They are said to be promising items written to the distributed cache file.

For example, consider the sequence S_1, according to Definition 1, utility of a, b, d, and f are 8, 6, 6, and 3 respectively. Similarly, from Definition 3, sequence utility of a is 43. Hence, the mapper emits the $\langle key, value\rangle$ pairs $\langle a, (8, 43)\rangle$, $\langle b, (6, 43)\rangle$, $\langle d, (6, 43)\rangle$, and $\langle f, (3, 43)\rangle$. In the same way, for every sequence, the output emitted by the mapper is given in Table 4. Now, these $\langle key, value\rangle$ pairs reach the reducer. In our example, the values (8, 43), (4, 34), (4, 54), and (8, 13) are with respect to the *key* a (refer to Table 4). The values associated with the other keys are shown in Table 5. The *value* is a pair of local utility and sequence utility. The sum of local utilities i.e. $8 + 4 + 4 + 8 = 24$ is the global utility of a. Similarly, sum of sequence utilities i.e. $43 + 34 + 54 + 13 = 144$ is *SWU* of a. The global utility and sequence weighted utilities for each item are mentioned in Table 6. As described in Algorithm 1, if the SWU of an item do not satisfy its MIU, then the item cannot generate the high utility patterns. So, the reducer returns the item and its global utility after satisfying the above mentioned condition. MIU of each item is varied (refer to Table 3) in

TABLE 4. <Key,Value> pairs emitted by first phase mapper.

Sequence	$\langle Key, Value\rangle$ pairs
S_1	$\langle a, (8, 43)\rangle, \langle b, (6, 43)\rangle, \langle d, (6, 43)\rangle, \langle f, (3, 43)\rangle$
S_2	$\langle a, (4, 34)\rangle, \langle b, (18, 34)\rangle, \langle c, (2, 34)\rangle, \langle d, (6, 34)\rangle, \langle e, (4, 34)\rangle$
S_3	$\langle b, (6, 40)\rangle, \langle c, (3, 40)\rangle, \langle d, (24, 40)\rangle, \langle e, (6, 40)\rangle, \langle f, (1, 40)\rangle$
S_4	$\langle a, (4, 54)\rangle, \langle b, (18, 54)\rangle, \langle d, (6, 54)\rangle, \langle e, (10, 54)\rangle$
S_4	$\langle a, (8, 13)\rangle, \langle c, (1, 13)\rangle, \langle e, (4, 13)\rangle$

TABLE 5. <Key,Value> pairs received by the first phase reducer.

Key	Value
a	$(8, 43), (4, 34), (4, 54), (8, 13)$
b	$(6, 43), (18, 34), (6, 40), (18, 54)$
c	$(2, 34), (3, 40), (1, 13)$
d	$(6, 43), (6, 34), (24, 40), (6, 54)$
e	$(4, 34), (6, 40), (10, 54), (4, 13)$
f	$(3, 43), (1, 40)$

TABLE 6. Global utility and sequence weighted utility.

Item	GU	SWU
a	24	144
b	48	171
c	6	87
d	42	171
e	24	141
f	4	83

the current study and it is 45% of the database utility for item a, 28% in case of item b, and so on. As mentioned in Definition 8, database utility is $43+34+40+54+13 = 184$. In the current example, every item satisfies its MIU threshold and emitted as output.

Algorithm 2 Second Phase

Input:
Time interval quantitative sequence dataset
DC_p - Distributed cache file with promising items
MMU-Table
Constraints - C_1,C_2,C_3, and C_4
Output: $\langle LHUTISP, LU\rangle$- Output patterns from each input partition and its utility
function Mapper
1: Let the promising item read from DC_p be i_p.
2: Let C be the candidate pattern set.
3: Let P be the output pattern.
4: Modify the input sequence by removing the unpromising items.
5: **for** each i_p **do**
6: Include $P \leftarrow \langle(0, i_p)\rangle$ into C.
7: **if** utility of P satisfies the MIU threshold **then**
8: Emit P and its utility
9: **end if**
10: Invoke *MRHUTSP-MMU*$(D \mid P, C, MMU, C_1, C_2, C_3, C_4)$
11: **end for**
12: **end function**

B. SECOND PHASE OF MRHUTSP-MMU

The input for the second phase is time interval quantitative sequence dataset, promising items, MMU-Table and time constraints. We should at first make sure that all the candidate patterns may not be output patterns. Hence, we use two structures, one represents the candidate set C and the other represents the output pattern P (Lines 2-3 of Algorithm 2). Initially, the unpromising items will be pruned from each input sequence (Line 4). Now, an initial pattern P is created which includes time 0 and promising item i_p (Line 6). If the utility of P (which is the global utility of i_p) satisfies its MIU then P is emitted as output (Lines 7-8). Inorder to extend the pattern P, we invoke the function *MRHUTSP-MMU*. This is a recursive function that extends the pattern P by scanning the projected database of P.

For example, the promising items received by the second phase mapper are $a, b, c, d, e,$ and f. Hence, the initial patterns generated are (0, promising item). Global utility of a is 24 which is less than its MIU (i.e. 45% of the database utility). That is, $(0, a)$ is not an output pattern. It is known that, super pattern of a non high utility pattern may be of high utility. Hence, we proceed to generate the super patterns of all the initial patterns by invoking Algorithm 3.

Algorithm 3 first reads the projected database of candidate pattern and generates all the possible $\langle time, item\rangle$ pairs. In this process, only the pairs which satisfy the upperbound are considered. Also, the time interval of the pair must obey the constraints C_1 and C_2 (Line 1). Now, the candidate pattern is updated by including all such pairs (Line 3). If the resulting candidate pattern satisfies the constraint C_4, then the function is called with the new candidate pattern as its argument (Lines 4-5). Later, if it satisfies the constraint C_3, we include it in the candidate pattern set (Lines 6-8). Next, if utility of the candidate pattern satisfies the MIU threshold, then it is emitted as output along with its projected database (Lines 10-12).

In the running example, considering the first partition, the $\langle time, item\rangle$ pairs generated from the projected database of $(0, a)$ are $(1, a)$, $(1, b)$, $(1, d)$, $(2, a)$, $(2, f)$, $(3, d)$, $(0, b)$, $(1, d)$, $(2, c)$. All such pairs generated from the initial patterns are given in Table 7. The constraint values are $C_1 = 0, C_2 = 3, C_3 = 1, C_4 = 3$. The $UB(\langle(0, a)(1, a)\rangle, S_1) = u(\langle(0, a)(1, a)\rangle) + RU(\langle(0, a)(1, a)\rangle, S_1) = 14 + 29 = 43 \geq PMIU(\langle(0, a)(1, a)\rangle)$, where $PMIU(\langle(0, a)(1, a)\rangle) = 33$. Similarly, the $\langle time, item\rangle$ pairs from $(0, a)$ satisfying the upper bound threshold are $(1, b)$ and $(1, d)$. The $\langle time, item\rangle$ pair from $(0, b)$ satisfying the upper bound threshold is $(1, d)$. Similarly, the $\langle time, item\rangle$ pairs generated from $(0, c)$ are $(0, f)$, $(1, b)$, and $(2, d)$. The $\langle time, item\rangle$ pair generated from $(0, e)$ is $(1, e)$ and from $(0, f)$ is $(1, b)$. Now, let us consider the $\langle time, item\rangle$ pairs from $(0, b)$. Among them, only the pair $(1, d)$ satisfies the upper bound threshold. $UB(\langle(0, b)(1, d)\rangle, T_1) = UB(\langle(0, b)(1, d)\rangle, S_1) + UB(\langle(0, b)(1, d)\rangle, S_2) + UB(\langle(0, b)(1, d)\rangle, S_3) = 0 + 26 + 36 = 62 \geq PMIU(\langle(0, b)(1, d)\rangle$, where $PMIU(\langle(0, b)(1, d)\rangle = 33$. Now, the pattern $\langle(0, b), (1, d)\rangle$ is extended with the new $\langle time, item\rangle$ pairs, i.e. $(2, c)$ and $(1, e)$. But, the $UB(\langle(0, b), (1, d), (2, c)\rangle) = 26 < PMIU(\langle(0, b), (1, d), (2, c)\rangle$ and $UB(\langle(0, b), (1, d), (1, e)\rangle) = 36 \geq PMIU(\langle(0, b), (1, d), (1, e)\rangle$, where $\langle(0, b), (1, d), (1, e)\rangle$ cannot be extended further with the $\langle time, item\rangle$ pairs. Hence, the recursion ends here and thus generated patterns from $(0, b)$ are $\langle(0, b), (1, d)\rangle$ and $\langle(0, b), (1, d), (1, e)\rangle$ with an utility of 54 and 36 respectively. The same recursive procedure is applied for the remaining initial patterns $(0, a)$, $(0, c)$, $(0, d)$, $(0, e)$, and $(0, f)$. Note that these output patterns are local to first partition. Consider the second partition and the same procedure applies to it and the output patterns generated are $\langle(0, a), (1, bd), (2, b), (3, e)\rangle$, $\langle(0, a), (1, b), (2, ab), (3, e)\rangle$, $\langle(0, a), (1, bd), (2, ab), (3, e)\rangle$, and $\langle(0, bd), (1, ab), (2, e)\rangle$, their local utilities are 50, 48, 54, and 50 respectively.

Algorithm 3 *MRHUTSP-MMU*

function *MRHUTSP-MMU*$(D \mid P, C, MMU, C_1, C_2, C_3, C_4)$
1: Read the projected database $D \mid P$ and generate all the $\langle time, item \rangle$ pairs such that each pair satisfies the constraints C_1 and C_2.
2: **for** each $\langle time, item \rangle$ pair **do**
3: Update $P \leftarrow \langle P, (time, item) \rangle$.
4: **if** P satisfies the constraint C_4 and $UB(P) \geq PMIU(P)$ **then**
5: Invoke *MRHUTSP-MMU*$(D \mid P, C, MMU, C_1, C_2, C_3, C_4)$
6: **if** the pattern P holds the constraint C_3 **then**
7: Include P in list C
8: **end if**
9: **end if**
10: **if** $U_L(P) \geq MIU(P)$ **then**
11: Emit P and $D \mid P$
12: **end if**
13: **end for**
14: **end function**

TABLE 7. (Time,Item) pairs generated from the initial patterns.

Initial pattern	(time,item) pairs
$(0, a)$	$(1, a) : 43, (1, b) : 35, (1, d) : 41, (2, a) : 23, (2, f) : 15, (3, d) : 12, (0, b) : 30, (1, d) : 12, (2, c) : 0$
$(0, b)$	$(0, d) : 29, (1, a) : 23, (1, f) : 15, (2, d) : 0, (1, d) : 62, (2, c) : 0, (1, e) : 0$
$(0, c)$	$(0, f) : 40, (1, b) : 39, (2, d) : 33, (2, e) : 0$
$(0, d)$	$(1, a) : 23, (1, f) : 15, (2, d) : 0, (1, c) : 0, (0, e) : 0$
$(0, e)$	$(1, a) : 34, (1, b) : 30, (2, d) : 12, (3, c) : 0$
$(0, f)$	$(1, d) : 0, (1, b) : 37, (2, d) : 31, (2, e) : 0$

Algorithm 4 Third Phase

Input:
$\langle P, D \mid P \rangle$
MMU-Table
Output:
$\langle P, U_G(P) \rangle$
function Mapper
1: **for** each pattern P_i in P **do**
2: Find the local utility of pattern P_i
3: Emit P_i and $U_L(P_i)$
4: **end for**
end function
function Reducer
5: By adding local utility of P_i calculate the global utility $U_G(P_i)$.
6: **if** $U_G(P_i) \geq MIU(P_i)$ **then**
7: Emit P_i and $U_G(P_i)$
8: **end if**
end function

C. THIRD PHASE OF MRHUTSP-MMU

In the third phase, the output from second phase is received i.e. local output patterns and their projected database. The mapper of third phase scans the projected database of each local pattern and finds the local utility (Line 2). The reducer finds the sum of local utilities, which results in the global utility (Line 5). Finally, only the patterns that satisfy the MIU threshold are emitted as output (Lines 6-8).

In our running example, for each local pattern, its utility in both the partitions is calculated. For instance, the utility of $\langle (0, b), (1, d) \rangle$ from partition 1 is 54 and from partition 2 is 0. Hence, the global utility of $\langle (0, b), (1, d) \rangle$ is 54 which satisfies its *MIU*, i.e. $U_G(\langle (0, b), (1, d) \rangle) \geq 28\%$ of 184. Similarly, the global utility of $\langle (0, a), (1, bd), (2, b), (3, e) \rangle$, $\langle (0, a), (1, b), (2, ab), (3, e) \rangle$, $\langle (0, a), (1, bd), (2, ab), (3, e) \rangle$, and $\langle (0, bd), (1, ab), (2, e) \rangle$ are 50, 48, 54, and 50 respectively. Among these patterns, $\langle (0, a), (1, bd), (2, ab), (3, e) \rangle$ satisfies its *MIU*. Finally, the output patterns and their utilities are $\langle (0, b), (1, d) \rangle : 54$ and $\langle (0, a), (1, bd), (2, ab), (3, e) \rangle : 54$

D. CORRECTNESS OF MRHUTSP-MMU

Theorem 1: Given a time interval quantitative sequence dataset, external utility of each item, time constraints and MMU-Table, MRHUTSP-MMU will generate all possible high utility time interval sequential patterns

Proof: We prove the theorem by stating that MRHUTSP-MMU will not lose any of the pattern in every phase.

1) Prune the unpromising items in the first MapReduce phase: According to Algorithm 1, the mapper generates all the items and their local utility and sequence utility. Mapper will not miss any item present in the dataset. The reducer aggregates the local utilities to form the global utility of item and aggregates the sequence utility to form the sequence weighted utility. Reducer will

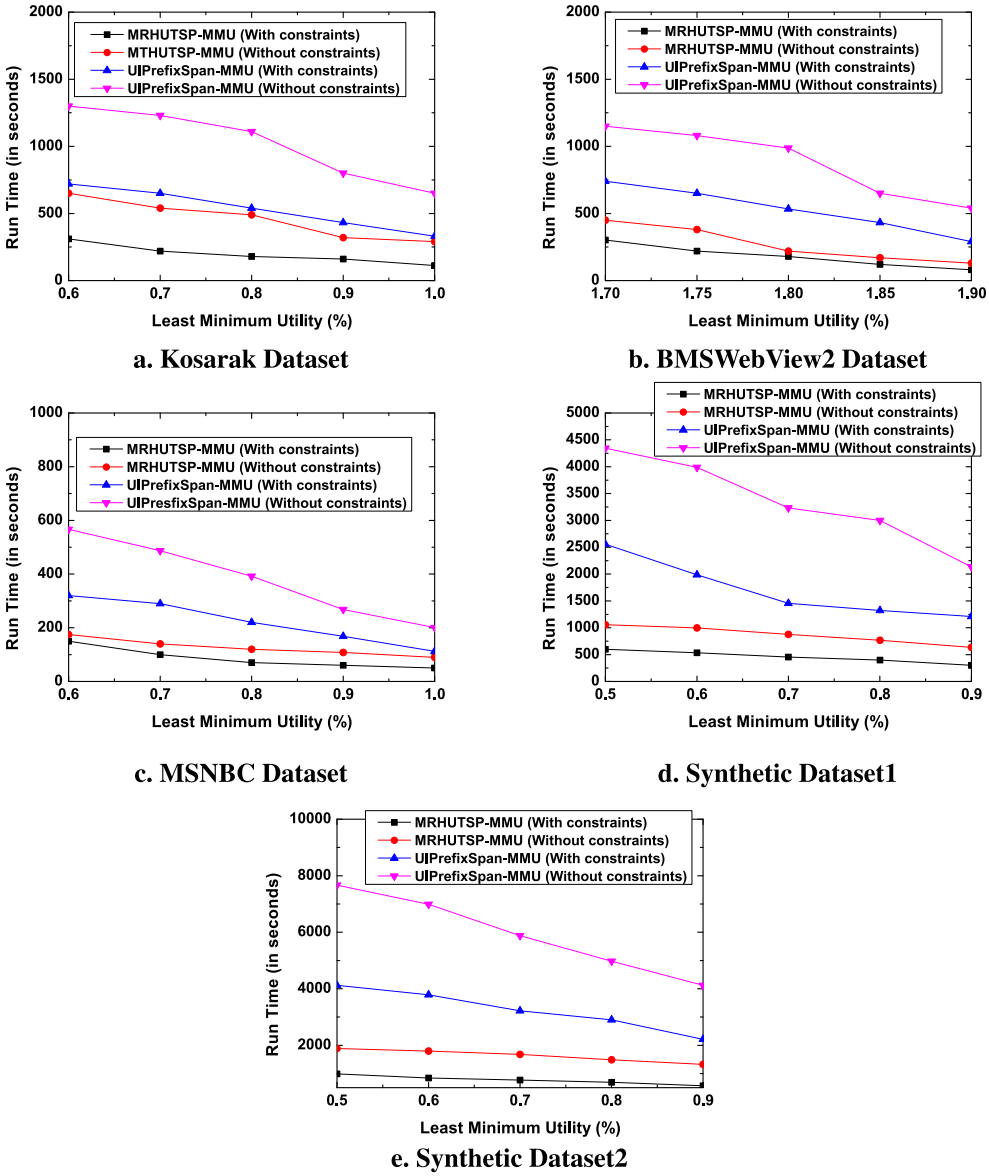

FIGURE 3. Performance of MRHUTSP-MMU.

output the items whose *SWU* satisfies its *MIU*. Thus, we do not miss any of the promising item.

2) Generate the candidate patterns whose length exceeds 2: In the second phase, each mapper generates entire local high utility sequential patterns following the time constraints. During this process, each candidate pattern is extended to its super pattern following the Property 1. Hence, the pattern is not extended if its upper bound does not satisfy the *PMIU* threshold. From Definition 13, whenever the local utility of a pattern is less than its *MIU*, then we can prune the candidate pattern. This pruning followed in the second phase will not lose any of the local candidate pattern.
3) Prune the patterns whose global utility do not satisfies its *MIU*: This is done in the third MapReduce phase. According to Definition 14, if the global utility of a pattern is less than its *MIU*, then it is not a high utility sequential pattern. Therefore, the third phase will not lose any of the output pattern.

Hence, each phase in MRHUTSP-MMU will not lose any of the pattern whose utility satisfies the threshold.

V. RESULTS

We conducted several experiments on three real datasets and two synthetic datasets to assess the performance of MRHUTSP-MMU. Kosarak,[1] BMSWebview2,[2] MSNBC[3] are the three real datasets. Kosarak consists of 990,002 sequences and it is obtained from FIMI repository. It stores

[1] *http : //fimi.ua.ac.be/data/*
[2] https://www.philippe-fournier-viger.com/spmf/datasets/BMS2.txt
[3] https://www.philippe-fournier-viger.com/spmf/datasets/MSNBC.txt

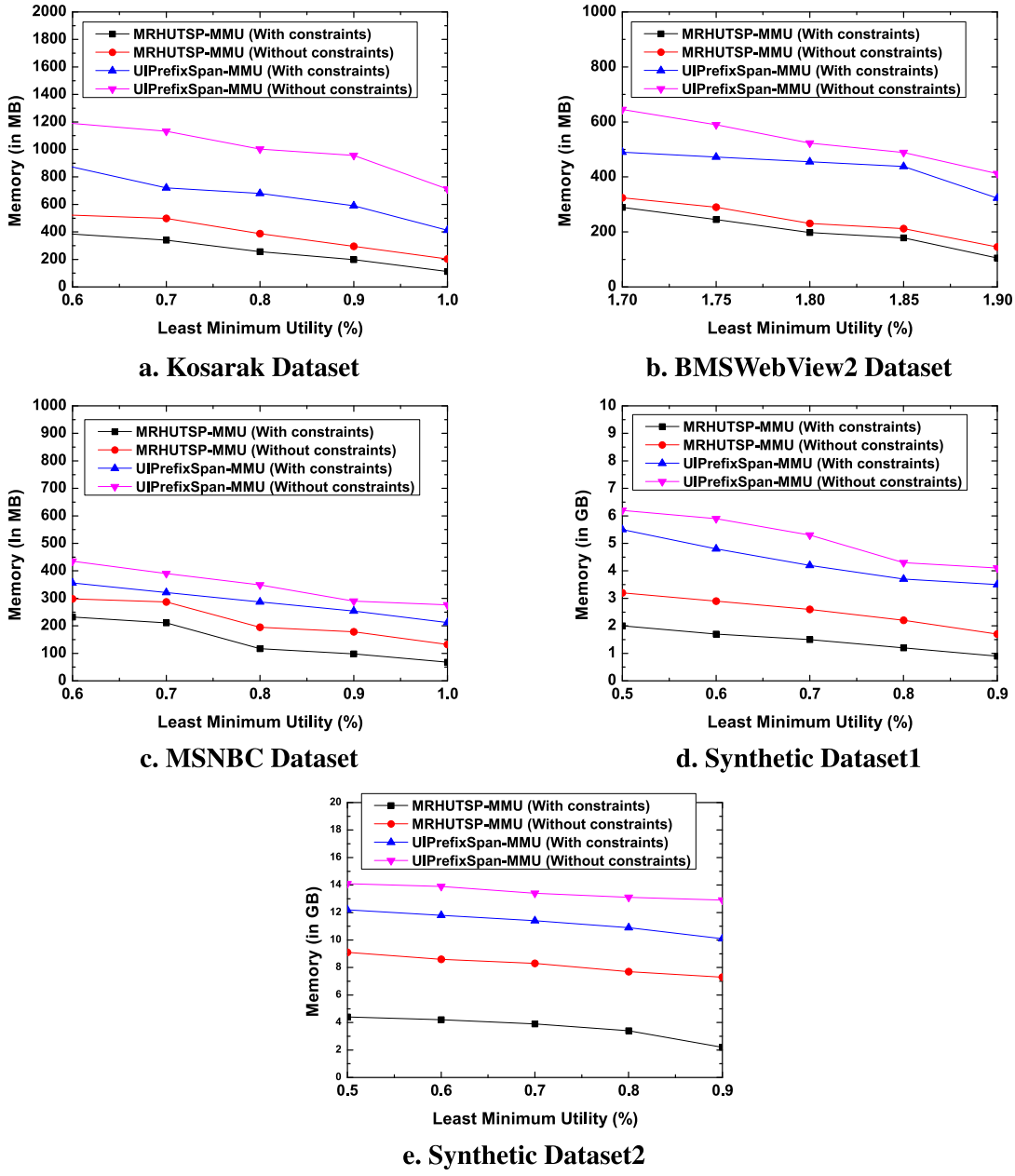

FIGURE 4. Memory usage of MRHUTSP-MMU.

TABLE 8. Synthetic datasets.

Dataset	D	C	T	N
Synthetic Dataset1	1	20	10	100
Synthetic Dataset2	10	20	10	1000

TABLE 9. Synthetic dataset generation parameters.

Parameter	Meaning
D	Number of sequences in million
C	Average number of itemsets per sequence
T	Average number of items per itemset
N	Number of unique items in the dataset

the click-stream data of an online Hungarian news portal. BMSWebView2 keeps track of an e-commerce website's click stream data. MSNBC dataset maintains the page visits of 989,818 users for a period of one day. BMSWebView2 and MSNBC are obtained from SPMF [38] data mining library. Inorder to evaluate MRHUTSP-MMU on big datasets, we generated two synthetic datasets following the procedure described in [1] and shown in Table 8. The parameters passed to the synthetic data generator are mentioned in Table 9. However, the above mentioned datasets do not include any internal/external utility information. So, we used a random number generator to provide the internal/external utilities from 1 to 10. The time information is included in the dataset based on the occurrence order of the itemset in a transaction. We employed a Hadoop cluster

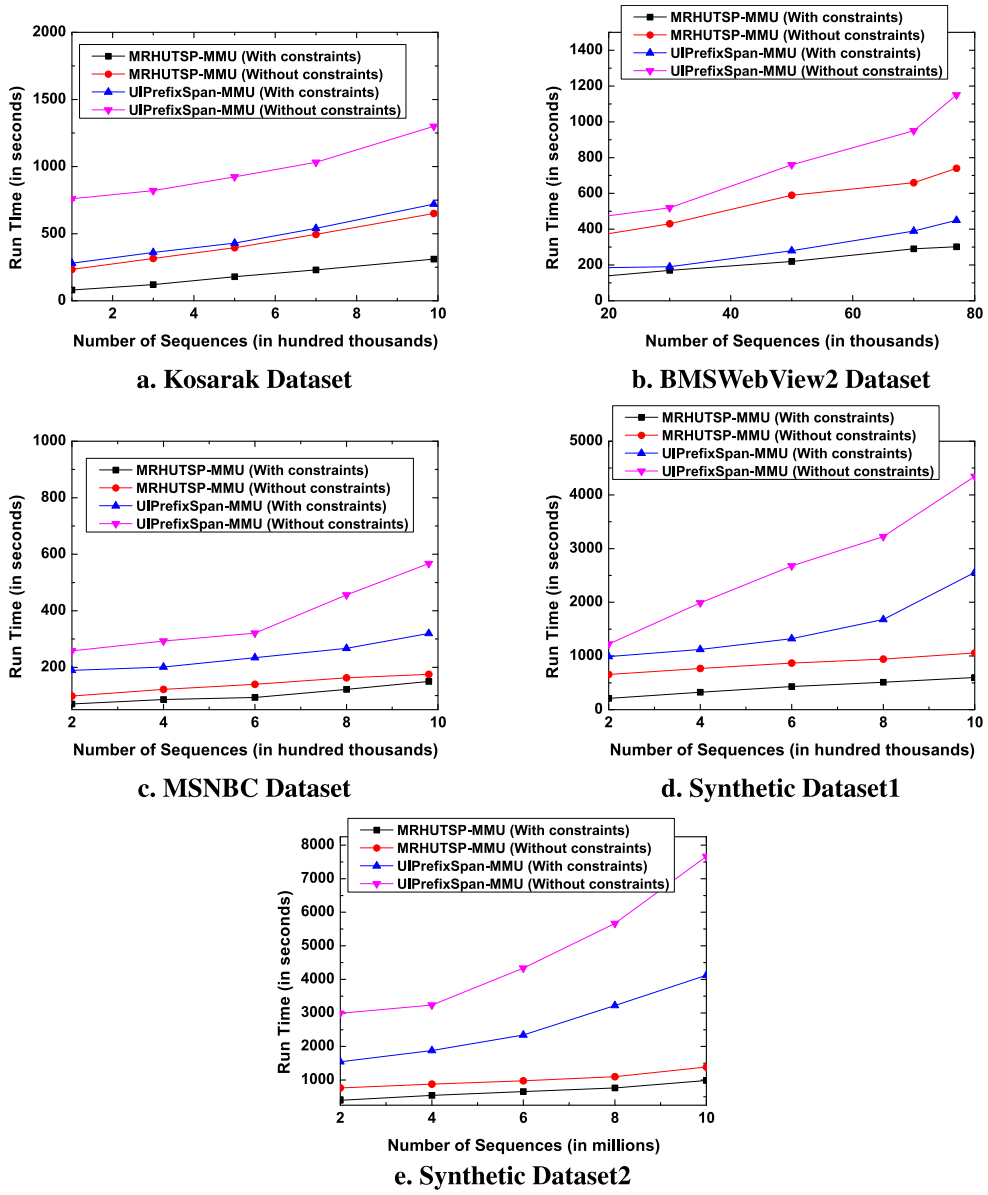

FIGURE 5. Scalability of MRHUTSP-MMU with respect to number of sequences.

including one node as a master and eight as data nodes. Each node has a 2.5 GHz Intel Xeon CPU with 16 GB RAM and Hadoop 2.9.1 installed. Java is used to implement all of the algorithms.

A. RUN TIME COMPARISON WITH RESPECT TO CONSTRAINTS AND LEAST MINIMUM UTILITY

The constraints used in the performance assessment are $C_1 = 0$, $C_2 = 2$, $C_3 = 0$, and $C_4 = 4$ on the real datasets and $C_1 = 0$, $C_2 = 5$, $C_3 = 0$, and $C_4 = 15$ on the synthetic datasets. MRHUTSP-MMU is compared with UIPrefixSpan-MMU in terms of run time and the findings are shown in Fig. 3. Both the algorithms are executed with and without constraints. As a result, it is found that, the algorithms perform better with time constraints. Because the time constraints induce a smaller number of candidates to be generated, which lowers the search space and increases the performance. Furthermore, for lower values of LMU, the run time will increase. This is because of huge candidates for lower LMU values. Additionally, more effort is spent during candidate evaluation. It is also observed that MRHUTSP-MMU is more efficient than UIPrefixSpan-MMU on all the five datasets. This is because of the MapReduce Algorithm employed in MRHUTSP-MMU which distributed the execution process among multiple nodes in the Hadoop cluster.

On Kosarak dataset, MRHUTSP-MMU with constraints is about 2.4, 2.8, and 5.4 times faster than MRHUTSP-MMU without constraints, UIPrefixSPan-MMU with constraints, and UIPrefixSPan-MMU without constraints. On BMSWebView2 dataset, MRHUTSP-MMU with constraints is about

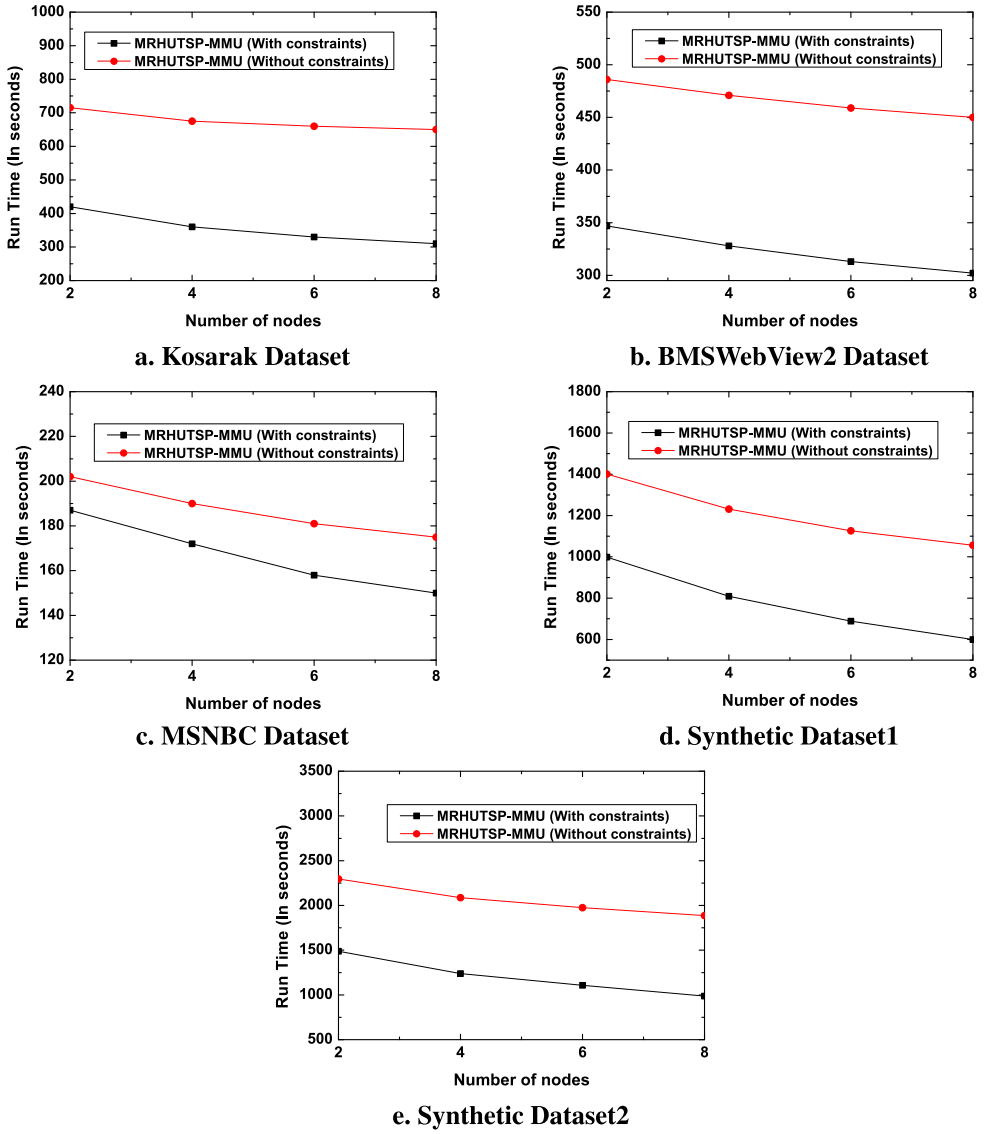

FIGURE 6. Scalability of MRHUTSP-MMU with respect to node count.

1.4, 3.1, and 5.3 times faster than MRHUTSP-MMU without constraints, UIPrefixSPan-MMU with constraints, and UIPrefixSPan-MMU without constraints. On MSNBC dataset, MRHUTSP-MMU with constraints is about 1.5, 2.7, and 4.5 times faster than MRHUTSP-MMU without constraints, UIPrefixSPan-MMU with constraints, and UIPrefixSPan-MMU without constraints On Synthetic Dataset1, MRHUTSP-MMU with constraints is about 1.9, 3.7, and 7.3 times faster than MRHUTSP-MMU without constraints, UIPrefixSPan-MMU with constraints, and UIPrefixSPan-MMU without constraints. On Synthetic Dataset2, MRHUTSP-MMU with constraints is about 2.1, 4.2, and 7.6 times faster than MRHUTSP-MMU without constraints, UIPrefixSPan-MMU with constraints, and UIPrefixSPan-MMU without constraints.

UIPrefixSpan-MMU requires more time because it needs to read the database for three times, and the time for scanning the database is directly proportional to the database size. Whereas, MRHUTSP-MMU scans the database only for two times. Also, the parallel execution of multiple map and reduce functions leads to reduced processing time. Hence, the distributed version MRHUTSP-MMU is more efficient than the original UIPrefixSPan-MMU.

B. MEMORY CONSUMPTION OF THE ALGORITHMS

The memory consumed by MRHUTSP-MMU is lesser than the other three approaches. The test reports of both the algorithms with and without constraints has been presented in Fig. 4. It is observed that both of them consume lesser memory when constraints have been applied. This is due to the fewer candidate sequences generated by applying the constraints. It is also noticed that the memory requirement tends to decrease with an increase in the threshold. This is mainly due to the generation of more number of patterns for lower values of threshold. On Kosarak dataset, the memory consumption of MRHUTSP-MMU

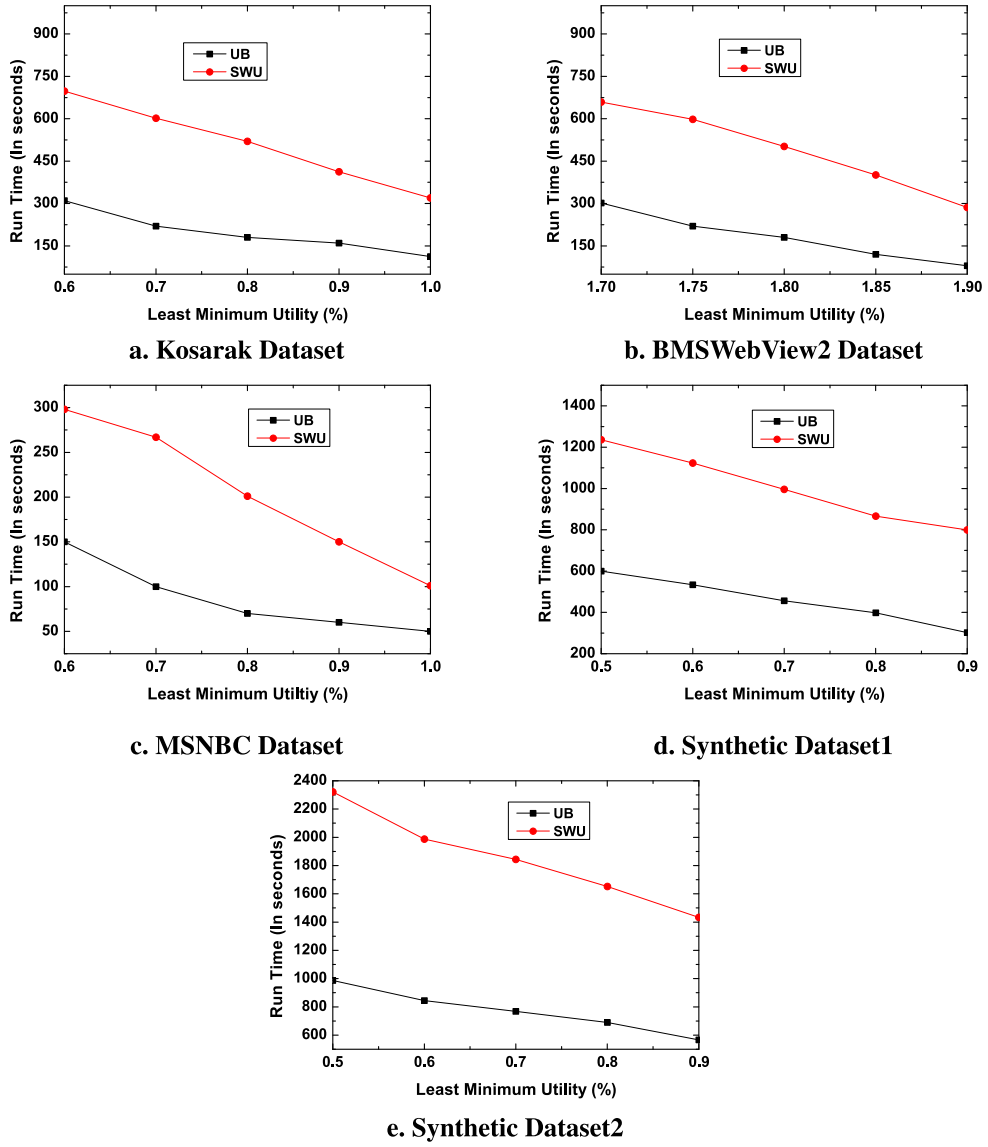

FIGURE 7. Upper bound evaluation.

with constraints is nearly 1.5, 2.8, and 4.2 times less than MRHUTSP-MMU without constraints, UIPrefixSpan-MMU with constraints and UIPrefixSpan-MMU without constraints. On BMSWebView2 dataset, the memory consumption of MRHUTSP-MMU with constraints is nearly 1.2, 2.3, and 2.7 times less than MRHUTSP-MMU without constraints, UIPrefixSpan-MMU with constraints and UIPrefixSpan-MMU without constraints. On MSNBC dataset, the memory consumption of MRHUTSP-MMU with constraints is nearly 1.6, 2.2, and 2.7 times less than MRHUTSP-MMU without constraints, UIPrefixSpan-MMU with constraints and UIPrefixSpan-MMU without constraints.On Synthetic Dataset1, the memory consumption of MRHUTSP-MMU with constraints is nearly 1.7, 3.1, and 3.6 times less than MRHUTSP-MMU without constraints, UIPrefixSpan-MMU with constraints and UIPrefixSpan-MMU without constraints. On Synthetic Dataset2, the memory consumption of MRHUTSP-MMU with constraints is nearly 2.3, 3.3, and 3.9 times less than MRHUTSP-MMU without constraints, UIPrefixSpan-MMU with constraints and UIPrefixSpan-MMU without constraints.

C. SCALABILITY TEST

To test the scalability of MRHUTSP-MMU algorithm, we carried out two experiments - scalability with respect to dataset capacity, scalability with respect to node count in the cluster. The results of former experiment are given in Fig. 5. The number of sequences used in the experiment on the Kosarak dataset ranged from 100,000 to the entire dataset (990,002 sequences). The size has been changed from 200,000 to full dataset size. For BMSWebView2 dataset, sequence count ranged from 10,000 to the full dataset size (77,512). The dataset size has been increased by 20 thousand

every time. For MSNBC dataset, the sequence count ranged from 200,000 to the full dataset size (989,818). The size is increased by 200,000 every time. The size of Synthetic Dataset1 is varied from 200,000 to 1,000,000. The size is increased by 200,000 for every experiment. The number of sequences considered for Synthetic Dataset2 is from 2,000,000 to 10,000,000. The size is increased by 2,000,000 each time. The least minimum utility used in the above experiment is 0.6, 1.7, and 0.6 for Kosarak, BMSWebview2, and MSNBC datasets, and it is 0.5 for Synthetic Dataset1 and Synthetic Dataset2. It is visible that the performance gradually decrease with the rise in dataset size. It is also noticed that the MRHUTSP-MMU scales well when executed on Synthetic Dataset1 and Synthetic Dataset2 compared to real datasets. This shows that MRHUTSP-MMU is more scalable than UIPrefixSPan-MMU especially on large datasets. The reasons are two fold. Firstly, MRHUTSP-MMU scans the dataset only for two times, whereas UIPrefixSpan-MMU scans the dataset for three times. Secondly, the distributed nature of the proposed algorithm parallelize the processing of sequences thereby reducing the execution time.

Figure 6 demonstrates the scalability of MRHUTSP-MMU regarding the updation of node count. For this experiment, the run time of MRHUTSP-MMU is noted with and without constraints. The algorithms are executed using 2, 4, 6, and 8 nodes. The centralized approaches are not used as they exhibited more running time especially for the synthetic datasets using 2 nodes. It is noticed that, the reduction in run time is more when we scaled from 2 to 4 nodes compared to 4 to 6 and 6 to 8 nodes.

D. UPPER BOUND EVALUATION

Two upper bounds *UB* and *SWU* are evaluated and the results are shown in Fig. 7. *UB* is a tighter than *SWU* and this is evident from Definition 16 and Definition 18. A tighter upper bound always results in less number of promising sequences there by the candidates for evaluation are reduced. This effects the algorithm's run time. MRHUTSP-MMU using *UB* as the upper bound executes faster compared to *SWU* as the upper bound. Especially, for lower thresholds, the *UB* approach outperforms the *SWU*.

VI. CONCLUSION

Conventional algorithms for finding sequential patterns with high utility excludes the time factor and consider that all items in a transaction have the same utility. The prime contribution of the current paper is to study the problem of multiple utilities with respect to time interval mining of sequences and propose a distributed solution that deals with the big data. Taking into consideration, we contributed MRHUTSP-MMU for finding the sequential patterns which include separate utility for each item and generates the time between each itemset of the pattern. The experimental work was carried out to know the efficiency of the algorithm with respect to time constraints. Also, MRHUTSP-MMU outperforms the non-distributed algorithms in terms of run time, memory, and scalability. As an extension to the current work, MRHUTSP-MMU can be more investigated to introduce most efficient pruning strategies.

REFERENCES

[1] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. 11th Int. Conf. Data Eng.*, Mar. 1995, pp. 3–14.
[2] N. M. Garofalakis, R. Rastogi, and K. Shim, "SPIRIT: Sequential pattern mining with regular expression constraints," in *Proc. 25th Int. Conf. Very Large Data Bases*, 1999, 223–234.
[3] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "FreeSpan: Frequent pattern-projected sequential pattern mining," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2000, pp. 355–359.
[4] M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences," *Mach. Learn.*, vol. 42, nos. 1–2, pp. 31–60, Jan. 2001.
[5] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining sequential patterns by pattern-growth: The prefixspan approach," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 11, pp. 1424–1440, Nov. 2004.
[6] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of sequential pattern mining," *Data Sci. Pattern Recognit.*, vol. 1, no. 1, pp. 54–77, 2017.
[7] S. Saleti and R. B. V. Subramanyam, "A novel mapreduce algorithm for distributed mining of sequential patterns using co-occurrence information," *Int. J. Speech Technol.*, vol. 49, no. 1, pp. 150–171, Jan. 2019.
[8] S. Saleti, P. RadhaKrishna, and D. JaswanthReddy, "Mining spatio-temporal sequential patterns using mapreduce approach," in *Soft Computing and its Engineering Applications*, K. K. Patel, G. Doctor, A. Patel, and P. Lingras, Eds. Cham, Switzerland: Springer, 2022, pp. 153–166.
[9] C. F. Ahmed, "A novel approach for mining high-utility sequential patterns in sequence databases," *ETRI J.*, vol. 32, no. 5, pp. 676–686, Oct. 2010.
[10] J. Yin, Z. Zheng, and L. Cao, "USpan: An efficient algorithm for mining high utility sequential patterns," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2012, pp. 660–668.
[11] G.-C. Lan, T.-P. Hong, V. S. Tseng, and S.-L. Wang, "Applying the maximum utility measure in high utility sequential pattern mining," *Exp. Syst. Appl.*, vol. 41, no. 11, pp. 5071–5081, Sep. 2014.
[12] O. K. Alkan and P. Karagoz, "CRoM and HuspExt: Improving efficiency of high utility sequential pattern extraction," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 10, pp. 2645–2657, Oct. 2015.
[13] J.-Z. Wang, J.-L. Huang, and Y.-C. Chen, "On efficiently mining high utility sequential patterns," *Knowl. Inf. Syst.*, vol. 49, no. 2, pp. 597–627, Nov. 2016.
[14] J. C. Lin, Y. Li, P. Fournier-Viger, Y. Djenouri, and J. Zhang, "Efficient chain structure for high-utility sequential pattern mining," *IEEE Access*, vol. 8, pp. 40714–40722, 2020.
[15] S. Saleti, "Incremental mining of high utility sequential patterns using MapReduce paradigm," *Cluster Comput.*, vol. 25, no. 2, pp. 805–825, Apr. 2022.
[16] W.-Y. Wang and A. Y.-Q. Huang, "Mining time-interval sequential patterns with high utility from transaction databases," *J. Adv. Comput. Intell. Intell. Informat.*, vol. 20, no. 6, pp. 1018–1026, Nov. 2016.
[17] C.-W. Lin, J. Zhang, and P. Fournier-Viger, "High-utility sequential pattern mining with multiple minimum utility thresholds," in *Proc. Asia–Pacific Web (AP Web) Web-Age Inf. Manag. (WAIM) Joint Conf. Web Big Data*, 2017, pp. 215–229.
[18] W. Gan, J. C.-W. Lin, J. Zhang, and P. S. Yu, "Utility mining across multi-sequences with individualized thresholds," *ACM/IMS Trans. Data Sci.*, vol. 1, no. 2, pp. 1–29, Jul. 2020.
[19] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[20] S. Sumalatha and R. B. V. Subramanyam, "Distributed mining of high utility time interval sequential patterns using mapreduce approach," *Exp. Syst. Appl.*, vol. 141, Mar. 2020, Art. no. 112967.
[21] S. Saleti, J. L. Tangirala, and R. Thirumalaisamy, "Distributed mining of high utility time interval sequential patterns with multiple minimum utility thresholds," in *Advances and Trends in Artificial Intelligence. Artificial Intelligence Practices*, H. Fujita, A. Selamat, J. C.-W. Lin, and M. Ali, Eds. Cham, Switzerland: Springer, 2021, pp. 86–97.

[22] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Proc. 5th Int. Conf. Extending Database Technol.*, vol. 1057, 1996, pp. 3–17.
[23] Y. Chen, "Discovering time-interval sequential patterns in sequence databases," *Exp. Syst. Appl.*, vol. 25, no. 3, pp. 343–354, Oct. 2003.
[24] Y.-L. Chen and T. C.-K. Huang, "Discovering fuzzy time-interval sequential patterns in sequence databases," *IEEE Trans. Syst., Man Cybern., B (Cybernetics)*, vol. 35, no. 5, pp. 959–972, Oct. 2005.
[25] H. Yao and H. J. Hamilton, "Mining itemset utilities from transaction databases," *Data Knowl. Eng.*, vol. 59, no. 3, pp. 603–626, 2006.
[26] R. Setiawan, D.-N. Le, R. Rajan, T. Subramani, D. K. Sharma, V. S. Ponnam, K. Kumar, S. M. A. Batcha, P. Dadheech, and S. Sengan, "Utilizing index-based periodic high utility mining to study frequent itemsets," *Arabian J. Sci. Eng.*, pp. 1–9, Jul. 2021.
[27] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1994, pp. 1–32.
[28] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Rec.*, vol. 29, pp. 1–12, Jun. 2000.
[29] J. M. Zaki and K. Gouda, "Fast vertical mining using diffsets," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 326–335.
[30] O. Jamsheela and G. Raju, "SR-mine: Adaptive transaction compression method for frequent itemsets mining," *Arabian J. Sci. Eng.*, vol. 47, no. 8, pp. 9641–9657, Aug. 2022.
[31] C. Zhang, M. Han, R. Sun, S. Du, and M. Shen, "A survey of key technologies for high utility patterns mining," *IEEE Access*, vol. 8, pp. 55798–55814, 2020.
[32] J. C.-W. Lin, W. Gan, P. Fournier-Viger, and T.-P. Hong, "Mining high-utility itemsets with multiple minimum utility thresholds," in *Proc. 8th Int. Conf. Comput. Sci. Softw. Eng.*, 2015, pp. 9–17.
[33] J. C.-W. Lin, W. Gan, P. Fournier-Viger, T.-P. Hong, and J. Zhan, "Efficient mining of high-utility itemsets using multiple minimum utility thresholds," *Knowl.-Based Syst.*, vol. 113, pp. 100–115, Dec. 2016.
[34] W. Gan, J. C.-W. Lin, P. Fournier-Viger, and H.-C. Chao, "More efficient algorithms for mining high-utility itemsets with multiple minimum utility thresholds," in *Proc. 27th Int. Conf. Database Expert Syst. Appl.*, vol. 9827, 2016, pp. 71–87.
[35] S. Krishnamoorthy, "Efficient mining of high utility itemsets with multiple minimum utility thresholds," *Eng. Appl. Artif. Intell.*, vol. 69, pp. 112–126, Mar. 2018.
[36] T. H. Duong, D. Janos, V. D. Thi, N. T. Thang, and T. T. Anh, "An algorithm for mining high utility sequential patterns with time interval," *Cybern. Inf. Technol.*, vol. 19, no. 4, pp. 3–16, Nov. 2019.
[37] S. Saleti, N. N. Sahithya, K. Rasagna, K. Hemalatha, B. S. Charan, and P. V. K. Upendra, "Constraint pushing multi-threshold framework for high utility time interval sequential pattern mining," in *Soft Computing and its Engineering Applications*, K. K. Patel, Gayatri, A. Patel, and P. Lingras, Eds. Cham, Switzerland: Springer, 2022, pp. 264–273.
[38] P. Fournier-Viger, C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The SPMF open-source data mining library version 2," in *Proc. 19th Eur. Conf. Princ. Data Mining Knowl. Discovery*, in Lecture Notes in Computer Science, vol. 9853. Cham, Switzerland: Springer, 2016, pp. 36–40.

SUMALATHA SALETI received the Ph.D. degree from the National Institute of Technology, Warangal, India, in 2020. She is currently working as an Assistant Professor with the Department of Computer Science and Engineering, SRM University, Amaravati, Andhra Pradesh, India. Her research interests include big data, data mining, and pattern discovery.

T. JAYA LAKSHMI (Member, IEEE) received the Ph.D. degree in link prediction (heterogeneous social networks) from the School of Computer and Information Sciences, University of Hyderabad, India, in 2019. She is currently working as an Assistant Professor with the Department of Computer Science and Engineering, SRM University, Amaravati, Andhra Pradesh, India. She is also a reviewer of reputed international journals. She has an overall teaching experience of 22 years. Her research interests include graph mining, recommender systems, natural language processing, and security analytics.

MOHD WAZIH AHMAD is currently working as an Assistant Professor with the Department of Computer Science and Engineering, Adama Science and Technology University, Adama, Ethiopia. He has supervised more than 20 postgraduate thesis and working as the Leader of Intelligent Systems SIG with ASTU Campus. His current research interests include machine learning, the Internet of Things, information retrieval, and soft computing applications in agriculture and health.