Contents lists available at ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

Full length article

Frequent itemset mining using cellular learning automata

Mohammad Karim Sohrabi*, Reza Roshani

Department of Computer Engineering, Semnan Branch, Islamic Azad University, Semnan, Iran

A R T I C L E I N F O

Article history:
Received 21 January 2016
Received in revised form
14 October 2016
Accepted 20 November 2016

Keywords:
Frequent itemset mining
Cellular automata
Data mining
Association rules
Parallel frequent itemset mining

A B S T R A C T

A core issue of the association rule extracting process in the data mining field is to find the frequent patterns in the database of operational transactions. If these patterns discovered, the decision making process and determining strategies in organizations will be accomplished with greater precision. Frequent pattern is a pattern seen in a significant number of transactions. Due to the properties of these data models which are unlimited and high-speed production, these data could not be stored in memory and for this reason it is necessary to develop techniques that enable them to be processed online and find repetitive patterns. Several mining methods have been proposed in the literature which attempt to efficiently extract a complete or a closed set of different types of frequent patterns from a dataset. In this paper, a method underpinned upon Cellular Learning Automata (CLA) is presented for mining frequent itemsets. The proposed method is compared with Apriori, FP-Growth and BitTable methods and it is ultimately concluded that the frequent itemset mining could be achieved in less running time. The experiments are conducted on several experimental data sets with different amounts of minsup for all the algorithms as well as the presented method individually. Eventually the results prod to the effectiveness of the proposed method.

1. Introduction

Frequent patterns are one of the major issues in the field of data analysis. Many books and articles have been published in this regard and significant progresses were made. Frequent patterns are essentially itemsets, sequences or infrastructures which are repeated in a data set with a frequency greater than or equal to a threshold determined by the user. In this paper, frequent patterns and frequent itemsets will be used interchangeably. Since the frequency of customer transactions tend to be enormous in the shops, hence they may not necessarily fit in memory. Besides, the potential number of frequent itemsets can vary from item numbers, Even though the actual number of frequent items could be less than that. Therefore calls for some scalable algorithms which are originally comparable with highly frequent and less frequent itemsets are heard. Many algorithms can be availed for this task, among which some act based on candidate creation and then investigation while others attempt to create a Tree without candidate generation and then find the frequent items by scanning on the Tree. Some other algorithms try to perform this task through the BitTable. The researchers first consider a number of algorithms in this field, and then present a cellular learning automata-based approach for frequent itemsets mining.

1.1. Statement of the problem

Suppose that E is an itemset. A transaction on E is $T = (t_{id}, E)$, where T_{id} is the transaction ID and E is an itemset. Also, a database of DB on E is: consists mainly of transaction sets in a way that each transaction enjoys a unique ID. The researchers claim that a transaction $T = (t_{id}, E)$ supports itemsets A if $A \subseteq E$. The cover of A set in DB is composed of a set of transactions that support A $\mathrm{cover}(A, DB) = \{t_{id} | (t_{id}, E) \in DB, A \subseteq E\}$ Besides, the support of A set in DB denotes the number of transactions already affiliated to A cover in DB $\mathrm{support}(A, DB) = |\mathrm{cover}(A, DB)|$. It is interpreted that an itemset is frequent if its support is more than thresholdsupport$(A, DB) \geq \mathrm{min_sup}$, which min_sup is the minimum threshold defined by the user.

1.2. Cellular learning automata

Cellular Learning Automata (CLA) is better known as a modified model of Cellular Automata. In Cellular Automata, the space is

* Corresponding author.
E-mail addresses: Amir_sohraby@aut.ac.ir (M.K. Sohrabi), r.roshany@gmail.com (R. Roshani).

http://dx.doi.org/10.1016/j.chb.2016.11.036
0747-5632/© 2016 Elsevier Ltd. All rights reserved.

defined as a network in which each part is called a cell. CLA is mostly resorted for the systems which comprise simple components and their behavior is determined and modified based on their neighbors' behavior as well as past experiences. Fig. 1 displays the examples of well-known neighborhood in Cellular automata.

Simple components of this model could exhibit complex behavior by means of interaction with one another. Each CLA interacts with an environment and is composed of a CA as well as Learning automata. Fig. 2 puts the relationship between the automata and the environment on display.

Taking the learning rules in CLA and neighbors' modes into account, each new transaction in dataset will result in either a reward or a penalty. The reward or penalty updates the structure of CLA respectively. The reward and penalty in Cellular Automata are considered in the following 3 modes:

1. Linear Reward Penalty (LRP): the amount of reward and penalty are the same in this mode.
2. Linear Reward Epsilon Penalty (LRεP): the amount of reward is manifold to penalty in this mode.
3. Linear Reward Inaction (LRI): In this mode, rewards are given without any penalties.

Based on their properties, the automata come into different classifications. A Cellular automaton is called regular if the neighborhood in cells already possesses a sorted regularity such as the neighborhood pattern of Von Neumann or Moore. In some applications, the need for an unlimited model like subsequence mining is from one neighborhood is felt, that is, the groups are completely random in these networks consequently it is not possible to come up with a sorted structure for them. This type of automata is labeled as Irregular Cellular Learning Automata (ICLA) (FathiNavid & Aghababa, 2012). Besides, a Cellular Learning Automata is called uniform if all cells in CLA have the same neighborhood function, rules and learning. Otherwise, it is named non-uniform (Esnaashari & Meybodi, 2007).

The rest of paper includes the following sections. In the second section, the researchers describe the main approaches to solve the frequent itemset mining problem. It is then followed by the explanation of the new method. Next comes the experimental results and it is concluded in the fifth section.

2. Related works

Data mining has numerous applications, including analysis of customer purchase patterns (Hu & Yeh, 2014), analysis of web access patterns (Ristoski & Paulheim, 2016), the investigation of scientific or medical processes (Wang, Davis, & Ren, 2016), and several types of prediction I social networks (Sohrabi & Akbari, 2016). There exist several types of data mining techniques. Association rule mining and frequent pattern extraction are two of the most

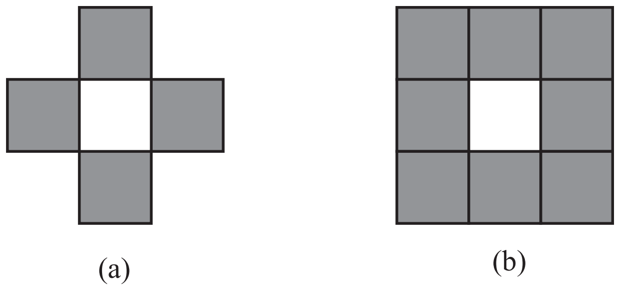

Fig. 1. Neighborhoods (a) Von Neumann, (b) Moore.

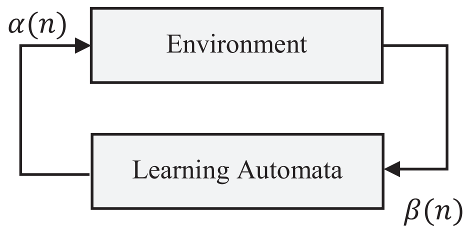

Fig. 2. The interaction of learning automata and the environment.

important data mining techniques which have been introduced in 1993 for the first time. There are two main approaches to find and extract frequent patterns from databases, efficiently. Apriori-like algorithms mine data using the 'candidate generation and test' method to find the frequent patterns in a breadth first search manner. Depth first mining algorithms use the second approach which usually compress the dataset in a tree structure and mine that compressed tree. Itemsets, Sequences, and graphs are three of most using types of patterns which have been mined and extracted from tremendous volumes of data by different pattern mining techniques. Since this work is a bit wise parallel mining algorithm based on cellular learning automata, the literature review is organized as follows in this section. First of all, some of the most important traditional itemset mining algorithms will be discussed. Then, several mining methods will be described which used bitwise approach as their improvement technique. Finally, different existing distributed and parallel mining approaches will be explained.

Agarwal, Imielinski, and Swami (1993) for the first time presented an interesting property called apriori in the form of association rules. Thanks to this property, a k-itemset can only be frequent when all its subsets are frequent. The result that could be retrieved from this property is that the super itemset of a non-frequent itemset are essentially non-frequent. Furthermore, this property will enable the apriori-based algorithm to piece together an itemsets of non-frequent k-items in itemsets mining of a (k+1)-item. Thus, in order to come up with a complete set of all existing frequent itemsets in a transaction database, firstly the transaction database should be scanned once thoroughly to find all the existing frequent 1-item itemsets and then generate all frequent 2-items itemsets (which can be created by these frequent items) namely as a frequent candidate. Since each of these 2-itemsets consists of only frequent items, they are potentially frequent within the transaction base. To ensure the frequency, it is required to scan the transaction database afresh and identify the frequent and non-frequent 2-items candidates. When the frequent 2-items itemsets are already identified, we take them identically to create 3-items candidates and to test their frequency or non-frequency through rescanning the transaction database. This process of using frequent k-item itemsets in generating (k+1)-items candidates and testing their frequency or non-frequency through a complete scan of transaction, continues until all the existing frequent itemsets in transaction database is mined. After the apriori method was presented, extensive studies were carried out to improve its efficiency and application. In 1995, Park attempted to use Hashing Techniques to improve its efficiency (Park, Chen, & Yu, 1995). Savasere, Omiecinski, and Navathe (1995) applied the Partitioning Method to improve the algorithm. In this method, by identifying a specific number of large frequent itemsets through a two-step process of transaction database scan -based on the large itemsets mined- the transaction database will be divided into the overlapping partitions and then the results of each separate partition mining will be subsequently merged. Toivonen (1996) used the Sampling Method to improve the efficiency of apriori method and Brin, Motwani, Ullman, and Tsur (1997)presented a method for Dynamic Itemsets Counting (DIC). Incremental Mining is yet another improvement presented by Cheung, Han, Ng, and Wong (1996). The algorithms presented before it, all assumed the transactions database to be static, and thus, the frequent itemsets used the association rule mining in a non-frequent way. In fact, the recent method was a way to preserve the association rule mining and to update them based upon the changes in transaction database.

Huge number of candidate itemsets, and multiple scans of database were two main drawbacks of all apriori-like methods which led researches to new depth-first search approaches without generating and testing candidate pattern. Han, Pei, and Yin (2000) coined a method called FP-Growth. This algorithm mines all itemsets without candidate creation. This method is based on 'divide and conquer' approach. In this method, once scanned, all the items included in transaction database are mined and based on their frequency in transaction database; they are sorted in descending order. Then, based on these ordered frequent items, the databases are formed in the frame of frequent pattern Tree (FP-Tree). This tree includes all the information in transaction database which is used for frequent itemsets mining. This tree is created as follows: it is created initially with a null root node, then each transaction is read from the database and all items included in the transaction are sorted in a descending order based on their frequency number and finally, they are added to the so-called Tree as a branch. Except the time that the Tree does not have any value, in other cases, in order to add, a common route of transaction items (which are already included in the Tree) is rather used and only one of them is added to the counting items with common aforementioned route. Along with adding a transaction to the FP-Tree, once a new node is created, this node will be added to its merged item list which is held by the header table. Next, the conditional FP-Tree is made out of the above mentioned pattern, and then the mining operation is applied recursively.

Many methods have been proposed for improving FP-Growth; one of which is the method of first-depth frequent itemsets presented by Agarwal, Aggarwal, and Prasad (2001). Another method is H-Mine Algorithm presented by Pei (2002). This method is used as hyper structure mining to find the frequent patterns. Liu, Pan, Wang, and Han (2002)suggested the alternative Tree construction method and the bottom-up and top-down mining of the same Trees created in FP-Growth manner in 2003 (Liu, Lu, Lou, & Yu, 2003). Besides, in 2003, Grahne and Zhu (2003) presented an array-based implementation of prefix table structure which is used to efficiently mine the FP-Growth Method.

Several intelligent methods have been proposed in the literature after FP-Growth which have been attempted to improve efficiency of mining process in different aspects using several intelligent techniques (Sohrabi & Marzooni, 2016), (Sohrabi & Ghods, 2014). Bit-wise representation of dataset and using corresponding techniques was one the most important improvements in frequent pattern mining process which compressed the database as a bit matrix and used bit operators to extract frequent itemsets, efficiently. BitTableFI, is an important attempt has been given to applying bitmap techniques in the frequent patterns mining process (Dong & Han, 2007). BitTableFI has a special data structure which is used horizontally and vertically to compress database for quick candidate itemsets generation and support count, respectively. Since BitTableFI suffer from the high cost of candidate generation and test, Index-BitTableFI has been proposed by Song, Yang, and Xu (2007). Index-BitTableFI computed the subsume index, and so identified itemsets that co-occurrence with representative item quickly by using breadth-first search at one time. Sohrabi and Barforoush (2012) used bie-wise mining approach to represent an efficient vertical bottom up Method to conduct extracting of frequent colossal patterns in high dimensional datasets.

Bit-mapping Table algorithm (Rezaei & Shiri, 2013) is an another bit wise method which works with a BitTable and tries to obtain k-number frequent itemsets without producing useless candidate itemsets and with a scan on transaction database. The input datasets are considered binary for this method. First, this method removes the non-frequent itemsets, then sorts the itemsets in a descending order according to the supports, and compresses each transaction in a numeric value of base 10 in accordance with each transaction available in dataset in an entry of Bit-mapping Table. Since the amounts of support items are highly valued in this method, item orders based on the size of supports not only does impact on transaction compression but also on creation of frequent itemsets. Thus, for the frequent 1- itemsets of base 2, a specific amount is allocated as a priority based on their number. After preparing the mapping table the priority array, the items of the priority array are selected respectively and commensurate with the item priority decrease in BitTable, only the components that are equal to higher item priority are added to that. These components are located in the selection list to improve the production of frequent itemsets in the target item. Then, this is anchored for a combination of items with less priority level than the targeted item. After reducing the priority list, the selection process is performed for any combination of components. If the number of reduced components is greater than the minimum support, then it is considered as the frequent itemset, otherwise, investigating the other combinations of target item will come to its end. If the number of non-zero component in the selection list mounts to greater than the minimum support, it will be preserved for improving the combination of length K+1. Hence, in order to expand the itemsets, the total priority and minimum priority will be retrieved from the items of a given itemset, and those components of BitTable which are excluded from this total priority are then inserted in the select list; once the deduct of total priority of the existing components are inserted in the selected list; those items whose priority are less than the minimum priority in the expanding itemset are opted for generating a combination of expanding itemsetin accordance with the priority array of items. This process continues until the entire frequent itemsets are generated.

Many optimizations were practiced for each of the above method, but the essence of the aforementioned algorithms remains the same. Each analyzed algorithm is characterized with its idiosyncratic weaknesses, as a case in point; apriori algorithm generates a significant number of frequent pattern candidates. Moreover, the frequent scanning of transaction database and investigating a large set of candidates are very time consuming. The FP-Growth method is fallible in that they cannot be fitted to the main memory when the size of a Tree is large; furthermore the Tree construction is not economically timely. The BitTable algorithm does not apply to some database for the table construction is known as binary for all the rows, in addition it faces inadequate memory space. That's why the researchers attempt to present a method based on Cellular Learning Automata which is not afflicted with the above mentioned problems and is able to obtain the frequent itemsets at the more efficient process time.

The explosive data volume in some big datasets causes traditional algorithms do not work efficiently. Using sampling based mining algorithms and distributed or parallel mining methods are two of most important approaches to deal with this challenge. Sampling based methods sample computationally-manageable subsets of data and extract frequent itemsets from them, rather than process the entire data set at once (Wu, Fan, Peng, Zhang, & Yu, 2015). Since the accuracy of sampling based methods is not 100%, when we need full accuracy in frequent itemset mining, we will use distributed mining approaches to extract all frequent itemsets of a dataset, accurately. Several parallel and distributed algorithms have been proposed so far to mine all the frequent itemsets of a dataset. Some of these algorithms partition the database into several parts and then mine each part independently. Other algorithms consider the entire database, globally and parallelize the processing operations.

An efficient parallel algorithm for mining frequent patterns on parallel shared nothing platforms has been represented in [23]. By efficiently partitioning the list of frequent items list over processors, this algorithm (PFP-Tree) tries to introduce minimum communication and synchronization overheads. Load Balancing FP-Tree (LFP-tree) is another parallel and distributed mining algorithm based on FP-tree structure [24]. The algorithm divides the item set for mining by evaluating the tree's width and depth. Moreover, a simple and trusty calculate formulation for loading degree was proposed in [24]. The communication time reduced in LFP-Tree by preserving the heavy loading items in their local computing node and hence LFP-tree reduce the computation time and has less idle time compared with PFP-tree. In addition, it has better speed-up ratio than PFP-tree when number of processors grow. For example, Cheung et al. (2006) proposed a tree partition-based technique that builds only one FP-tree and partitions it into several independent parts to mine frequent patterns by assigning one part to one processor. SABMA is an efficient scalable systolic array based parallel algorithm which uses a bit matrix to compress the dataset and mapping the mining algorithm on the systolic arrays architecture (Sohrabi & Barforoush, 2013). In this paper, we propose a novel distributed frequent itemset algorithm based on cellular learning automata, which extract set of all frequent itemsets. We compare the experimental results of our algorithm with SABMA.

3. The proposed method

In this section, the mining method of this paper is introduced. In this method, first the frequent 1-Itemsets are extracted from the dataset. Then, given a set of 1-itemiv sets, the dataset is compressed and the redundant items are respectively pruned. In other words, the items remain in every row of dataset which are in the frequent 1-Itemsets. Thus, the transactions are compressed with their frequency in the dataset in the consideration. In Data Mining field, this process is known as pre-processing. Thereafter, we have nothing to do with the initial dataset and the entire task is performed according to the frequent 1-Itemset and the compressed and pruned dataset. After pre-processing, the cellular automata environment starts to work and creates the cellular automata cells according to the frequent 1-Itemsets and reads the rows of the obtained dataset one by one and simultaneously transfers them to all the cells, and then each cell operates with the other cells in a parallel manner. The L_{R-I} method takes advantage of the CLA in that includes no penalty. Since there is no regulation for neighborhood among the cells, regular neighborhood structure such as Von Neumann or Moore could not be used. Here, the cells' neighborhood is created according to the dataset transaction which is called Irregular Cellular Learning Automata (ICLA). Besides, the presented Cellular automata is assumed to be uniform, hence the learning process within the cells are the same. The cells have a proximity list for neighbors that update it after receiving the transaction from the environment. Finally, each cell reviews its proximity list and prunes the neighbors that are defined by the user to be less than the minimum threshold in its proximity list. Then, each cell obtains the frequent items according to a survey which is administered according to the proximity list and sends them to the environment.

3.1. Mining of frequent 1-itemsets

The frequent 1-Itemsets are used for pruning and compressing the datasets and creating Cellular automata cells. In order to obtain the frequent 1-Itemsets, first all the 1-Itemsets have to be extracted along with their frequent. Then, the frequent 1-Itemsets are obtained according to the threshold defined by the user. The mining process of 1-Itemsets based on dataset is as follows:

1 The list of 1-Itemsets items is created with an initial value of null.
2 The current row of dataset is read and all its 1-Itemsets are obtained.
3 All the 1-Itemsets in the current row are reviewed in 1-Itemsets' list. If they are already available in this list, one is added to this number. If not, inserts it in this list with number 1. This step is performed for the entire 1-Itemsets of current row.
4 If there is still no unread row within the dataset, step 2 is repeated.

Fig. 3 shows an example of the dataset wherein the 1-Itemsets along with their number of frequency are presented in the form of Fig. 4 based on the above trend.

By investigating 1-Itemsets and considering minsup, those lower than the minimum threshold defined by the user are removed and the frequent 1-Itemsets are consequently obtained. Assuming that the minimum threshold of minsup is 50%, according to the dataset's row numbers the minimum number of iteration is 5. Therefore, the frequent 1-Itemset is like Fig. 5.

3.2. Compressing and pruning the dataset

When the frequent 1-Itemsets are extracted, in order to increase the operation speed, the dataset needs to be compressed and accordingly the redundant items to be pruned. The purpose of pruning is to avoid useless work that has no effect on output result and only increase the processing time. The redundant items are the items that are not found in the frequent 1-Itemsets. In addition, in order to avoid the redundant tasks for similar transactions; we compress the same transactions together. This operation is as the following:

1 The compressed dataset are formed with a null value.
2 The current row of dataset is read.
3 All the 1-Itemsets of current transaction are extracted.
4 Then, the presence of each obtained item is checked in the frequent 1-Itemsets. If these items are not present in the frequent 1-Itemset, they are called redundant and are therefore removed.

T_{id}	ItemSet
1	b, c, d, h, i, j
2	m, n
3	a, b, d, f, m, n, p
4	a, b, e, c, p
5	a, b, f, p
6	c, d, f, h, i, j, p
7	a, b, d, e, f, h, g, p
8	a, b, c, i, m, p
9	c, f, g, n, p
10	a, b, e, f, h, k, n, p

Fig. 3. An example of dataset.

Item Key	a	b	c	d	e	f	g	h	i	j	k	m	n	p
Count	6	7	5	4	3	6	2	4	3	2	1	3	4	8

Fig. 4. 1-itemsets.

Item Key	a	b	c	f	p
Count	6	7	5	6	8

Fig. 5. The frequent 1-Itemset.

5 When the transaction items were pruned and the number of its items were greater than Zero, its presence in the compressed dataset is reviewed; if already available, one is added to the number of iterations, otherwise it is created as a new transaction with an iteration value of 1 in the compressed dataset. This will prevent the redundant tasks for similar transactions.
6 In case that there are still unread rows in the dataset, the process is repeated from step 2.

Finally, a list of compressed transactions is provided as in Fig. 6 wherein there are no redundant items.

3.3. Cells and proximitylist

The cells of cellular automata are created according to the frequent 1-Itemsets. Therefore, here the cellular automata have 5 cells each of which presents a frequent Item (Fig. 7). Since there are no specific rules in the cells' neighborhoods, the regular neighborhood structure such as Von Neumann or Moore could not be applied among them when it comes to each dataset's rows. This is called the Irregular Cellular Learning Automata (ICLA).

The cellular automata environment reads all the dataset's rows one by one and then transfers them to the cells in every step. After receiving a row of compressed dataset, the cells initiate their operation simultaneously with the other cells. If the cell itself is present among the target itemset row, it continues the operation, otherwise it stops the operation. There is a proximity list in the cells wherein the connection type between the neighbors and their number of iteration is diagnosed. The cells update their proximity list according to the received transaction. Using this list, each cell gets the frequent items and transfers it to the environment. When a transaction's itemset along with the iteration number is sent to a cell, the cell does the following for all the itemsets except for itself.

1 If there is a neighborhood for each of the input items, it is added to that number as many as the received number. In contrast, if there is no neighborhood, that item is created as a new neighbor as many as the input iteration.
2 Then, if the input items are more than 2, their two-member subsets are recorded in the neighbors' proximity list.

Row#	ItemSet	Count
1	b,c	1
2	a,b,f,p	4
3	a,b,c,p	2
4	c,f,p	2

Fig. 6. The compressed and pruned dataset.

For example: suppose that a, b, c, f, p cells which are created based on the frequent 1-itemsets and the compressed dataset of Fig. 6 is available, then the proximity list of cells is updated as following:

The environment reads the {b, c} row which has the iteration value of 1 and transfers it to all cells. Since B and C cells are in this itemset, they process on this itemset, but because the A, F, P cells are not in this itemset, they do no action.

- C cell reviews the {b} item presence among its neighbors and since it has no neighbor, it creates {b} item in its neighbors' list with the iteration value of 1.
- B cell creates {c} item in its neighbors as well.

So far, the neighbors and proximity list of each cell are as in Fig. 8:

Then, the second row of compressed dataset which consists of {a, b, f, p} is read with an iteration value of 4, and is sent to the cells. Because all the A, B, F, P cells are in this itemset, they process on this itemset. The running of these cells is shown in the following discussion.

The presence of {a}, {f}, {p} among the neighbors are reviewed for B cell. Since item {a} is not already available in the neighborhoods, it is created with an iteration value of 4. Should the items remain from that specific item ahead (here from item {a} onward) then they have to be recorded in the proximity list. In this case, we can see that the {p}, {f} items remain, thus they are created in proximity {a} with value 4. If they are already present, their value is increased as many as their iteration number. Note that each cell does not consider its value among the items. Since item {f} is not already present among the neighbors' list, it is created in the neighbors' list of B cell with an iteration value of 4. In order to avoid creation of redundant itemsets, {a} is not recorded in the proximity of {f}. This action prevents the creation of redundant itemsets with different permutations. We know that {a, f} and {f, a} are the same patterns and are not different from one another. Hence, the remained itemset {p} is recorded in its proximity list with an iteration value of 4. In addition, the itemset {p} is also reviewed in the neighbor list of cell B, since it is not already available; it is created with an iteration value of 4. Regarding the explanation on the permutation, itemset {p} has no remaining element. The other cells also execute these steps simultaneously. Fig. 9 shows the created neighbors and the proximity list of the cells after the running of second row of the dataset.

Fig. 10 shows the created neighbors and the proximity list of the cells after the running of third row of the dataset. Because the third row of the dataset consists of items a, b, c, and p, the cells A, B, C, and P should process on this row. Similar to the second row's process, each cell checks the presence of other items among its neighbors. For example, cell A checks the presence of {b}, {c}, and {p} among its neighbors. Since items {b} and {p} are already present, their value is increased as many as their iteration number. Since the third iteration value is 2 and the former value of {b} and {p} are 4 in cell A, their value is modified to 6. Since item {c} is not already available in the neighborhoods, it is created with an iteration value of 2. The other cells also execute these steps simultaneously. The result of this parallel process is shown in Fig. 10.

Finally, Fig. 11 the result of the execution of the last row of

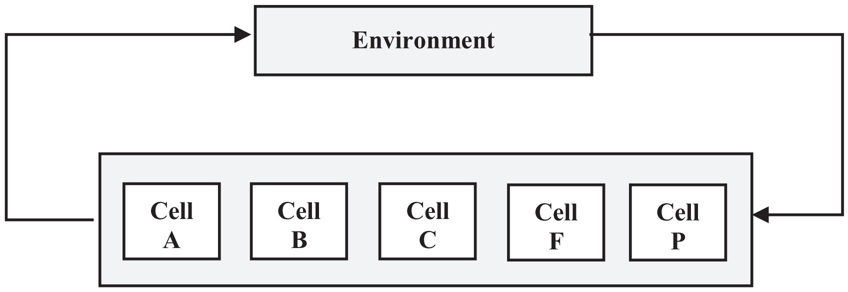

Fig. 7. Cells based on frequent 1-Itemsets.

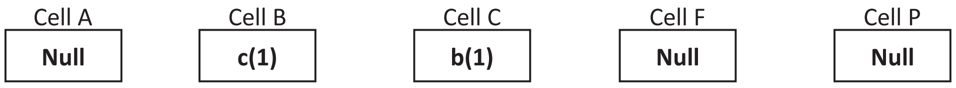

Fig. 8. The proximity list of cells after the first row running.

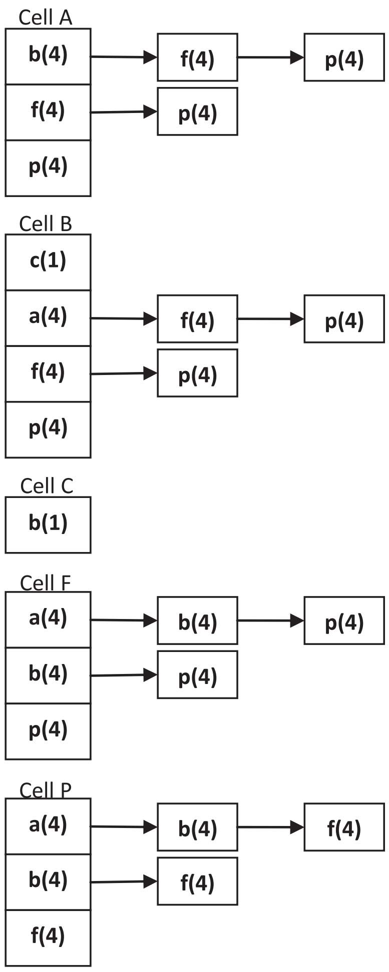

Fig. 9. The proximity list of cells after the second row running.

dataset. C, F, and P are active cells in this step, because the last row of dataset consists of {c, f, p} itemset. For example, cell C updates its 'p' value from 2 to 4, and creates a new item 'f' in its neighborhood with an initial value of 2.

After completing all cells' process on all rows of dataset, the automata is ready to prune and scan the proximity lists for extracting the list of all frequent itemsets.

3.4. Pruning and scanning the proximity list

When all the transactions are sent to the cells by the environment, each cell removes the neighbors and proximities which are less than the minimum threshold already defined by the user from its proximity list. In our example, because the dataset contains 10 rows and the minimum support is supposed as 50%, only the items with value no less than 5 is remaining and other will be pruned. The result of pruning on the example automata is shown in Fig. 12. The refined proximity list is used for scanning and eventually the k-Itemset is achieved.

When the environment sends all the rows available in the compressed dataset and once the cells manage to complete the related tasks, the environment tends to collect and extract the frequent items. The 1-itemsets consisting of cells are all presented in the output. The 2-itemsets also could be created out of the cells and their neighbors are also provided in the output. Considering minsup equals to 50%, each cell acts as following:

First, items {a}, {b}, {c}, {f}, {p} are frequent 1-item elements and are placed in the frequent item list. Then, each cell acts as following regarding Fig. 12:

- Cell A places its {b}, {p} neighbors in the frequent item lists, with a combination of itself mostly known as {a, b}, {a, p} as the frequent 2-items which are in direct connection to cell A. The {b, p} items are the frequent 2-items in cell A which are to be combined with cell A. In other word, the {a, b, p} items are placed in the frequent item lists as the frequent 3-itemsets.
- Cell B places its {a}, {b} neighbors in the frequent item lists, with a combination of itself better known as{a, b}, {b, p} as the frequent 2-items which are in direct connection to cell B. The {a, p} items are the frequent 2-items in cell B which are to be combined with cell B. In other word, the {a, b, p} items are placed in the frequent item lists as the frequent 3-itemsets.
- Cell C has no neighbors, thus, no frequent item is created out of its combination.
- Cell F has only one neighbor which recurs it with its combination that is {f, p} as the frequent 2-item.
- Cell P places its {a}, {b}, {f} neighbors in the frequent item lists, with a combination of itself which is {f, p}, {b, p}, {a, p} as the frequent 2-items which are in direct connection to cell P. The {a, b} items are the frequent 2-items in cell P which are to be combined with cell P. In other word, the {a, b, p} items are placed in the frequent item lists as the frequent 3-itemsets.

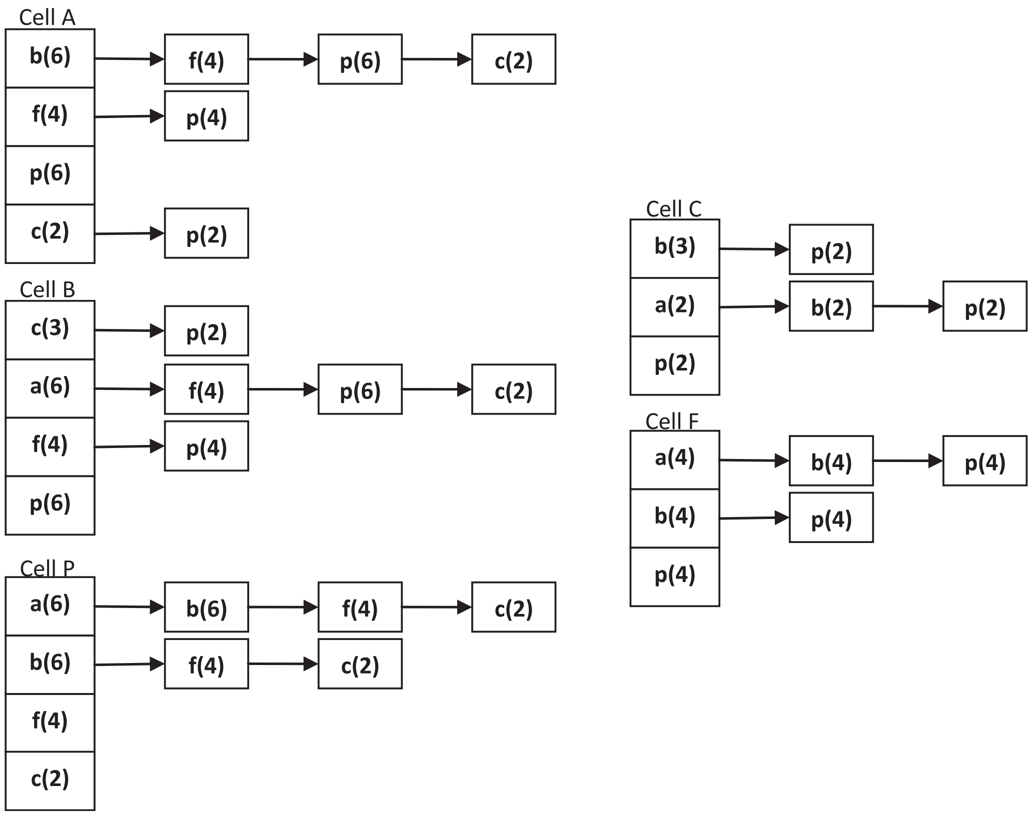

Fig. 10. The proximity list of cells after the third row running.

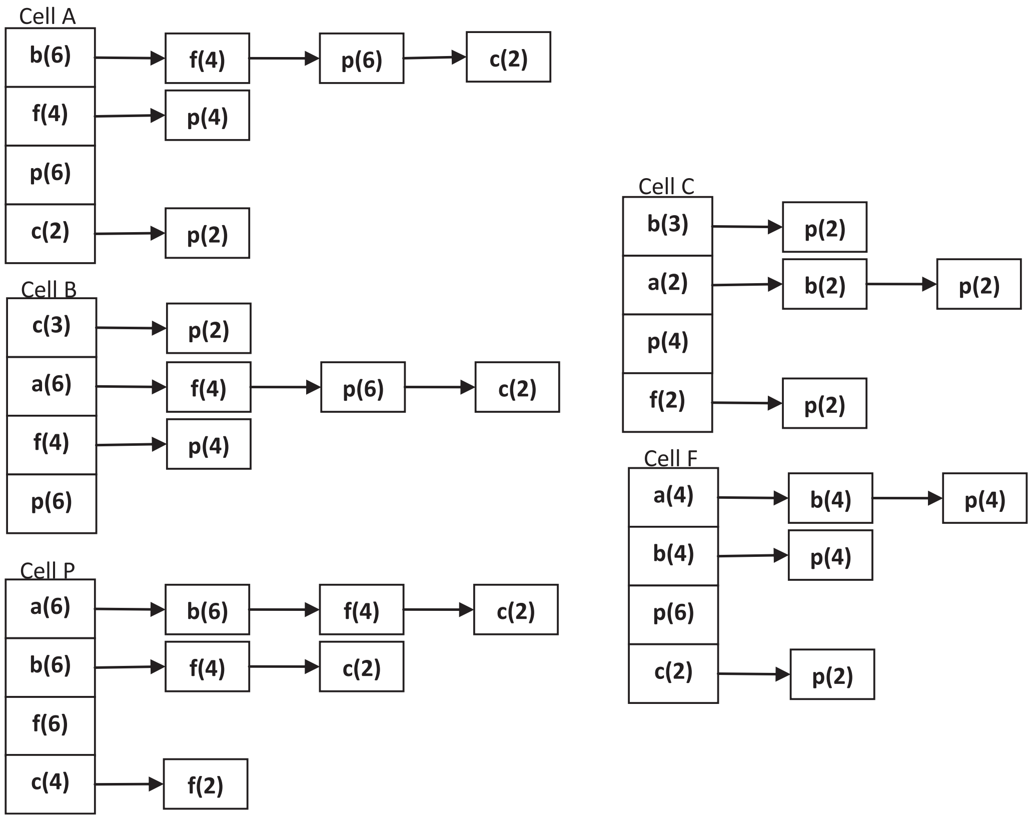

Fig. 11. The proximity list of cells after the fourth row running.

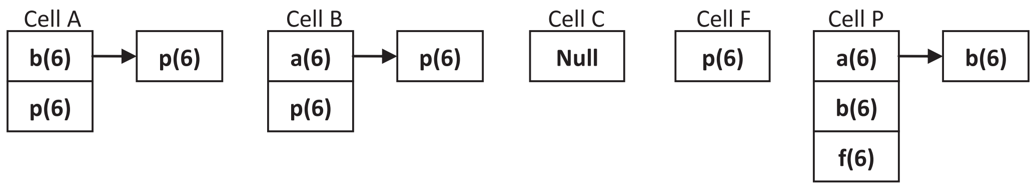

Fig. 12. The proximity list of cells after pruning with minsup = 50%.

As seen, each cell obtains a frequent itemset. By gathering the results of each cell, all the frequent items could be obtained. Note that each cell may create similar items. To avoid the presentation of redundant items, their presence could be investigated before they are sent to the comprehensive list of the frequent items. If they are already present in this list, they ignore that item; otherwise they place it in the comprehensive list of the frequent items.

In order to get 3-items after the presence of the frequent 2-items elements in the cells, and to be frequent 3-items, their non-shared items must be present in cell as a 2-itemsets. If so, 2-items combination could bealike the frequent 3-items, otherwise this is not the frequent 3-items. This relation also makes sense for the k-items that must have (k-1)-items.

For instance, suppose that the F cell present in Fig. 11 is the final state of the cell and min_sup is equal to 2, hence in order to get the 3-items {a, b, p} in cell F, first the 2-items of {a, b} and {a, p} -with their the default minsup as the frequent 2-items in cell F in mind- are reviewed. Now, if the non-shared elements of these two sets are present in the proximity list of cell as the frequent 2-items i.e. {b, p}, which is so, the {a, b, p} is the frequent 3-items.

3.5. The proposed algorithm pseudo-code

Pseudo-code of the proposed algorithm is as Fig. 13 depicts.

In this algorithm, first, the frequent items are extracted, and the compressed dataset is constructed in lines 2 and 3. In line 4, the environment creates the Cells of CLA according to the frequent items. In the first lopp of the algorithm, the environment reads the transaction rows and sends them to the created cells to be processed. When the entire rows of dataset which are read by the environment are sent to the cells and the cells' operation is over, the final list of total extracted frequent itemsets are created and initiated with a null value in line 6. All extracted frequent itemsets from all cells are stored in this list which is names as TFIL. In the second loop of algorithm, the obtained proximity and neighborhood list of cells are pruned and then the frequent itemsets of each cell will be mined.

4. Experimental result

In this section, the proposed algorithm was tested and the results were compared to SABMA (Sohrabi & Barforoush, 2013) and PP-Tree (Javed & Khokhar, 2004) algorithms results. It was demonstrated that using CLA, the frequent itemsets could be mined at the more efficient process time. Several current datasets in Table 1 were used.

The proposed algorithm and all the compared ones in this study were run and tested in the Visual studio. NET 2013 environment and in C# Programing Language on a computer of features Windows 7 x64 equipped with the processor Intel 2.8 GHz core2 Quad

```
Algorithm CLA-Mining
Input
    TDB: Transaction Database
    minsup: Minimum support threshold
Output
    TFIL: list of all extracted frequent itemset
Begin
    1.  Start
    2.  FI = Extract frequent items            //set of frequent items
    3.  CDS = Preprocess (Dataset, FI)        // Compressed dataset
    4.  Create cells by frequent items of FI
    5.  For i = 1 to number of CDS's rows
                5.1. Read a transaction from CDS
                5.2. Pass transaction to cells in parallel
                5.3. Update cells' neighbors and proximity list by new transaction
    6.  Initialize TFIL
    7.   For j = 1 to number of cells of automata
                7.1. Run PruneNeighbors() for cell[j]
                7.2. Run DFS() for cell[j]
                7.3. Foreach (anItemset on cell[j].FrequentItemset)
                7.4. IF (anItemset not exists in TFIL) Then
                            TFIL.add (anItemset)
                     Else nothing
    8. Return (TFIL)
End
```

Fig. 13. Proposed algorithm.

Table 1
Characteristics of test datasets.

Dataset name	Transaction#	Items#	Size
Mushroom	8124	119	0.56 M
Retail	88162	16469	3.97 M
Pumsb	49046	2113	16.3 M
Accidents	340183	468	33.8 M
Kosarak	990002	41270	30.5 M

and RAM 8 GB. The results of algorithms comparison could be seen in Figs. 14–18.

Figs. 14–18 show the result of running two algorithms SABMA and the newly proposed algorithm on real standard datasets. It is clearly observed that proportionate to the increase in minimum support, the algorithm's performance in all dataset decreases. Since CLA-Mining method uses redundant transaction elimination for compression of datasets, its functionality is better than former parallel approaches. On the other hands, the improvement of the method functionality is directly related to the amount of redundant transaction production after pruning the dataset using the given minsup. For example, in dataset of Fig. 3, when we supposed minsup = 50% and pruned the dataset, 4 transactions 3, 5, 7, and 10 were transformed to {a, b, f, p} and compressed as one transaction with count 4 in our compressed dataset. Meanwhile, if we use minsup = 40%, then 4 transactions 3, 5, 7, and 10 will be transformed to {a, b, d, f, n, p}, {a, b, f. p}, {a, b, d, f, h, p}, and {a, b, f, h, n, p}, respectively, which cannot merge to each other and compress the dataset.

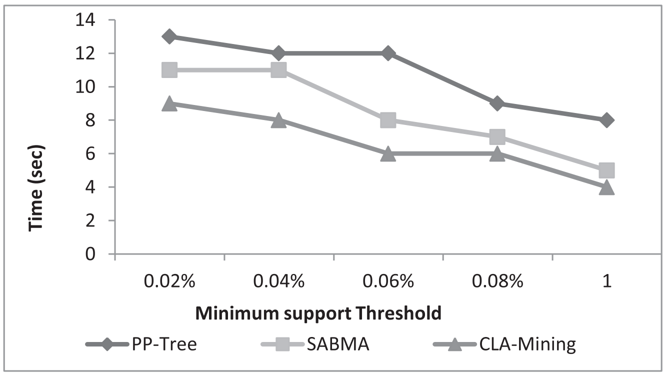

Fig. 14. Dataset mushroom.

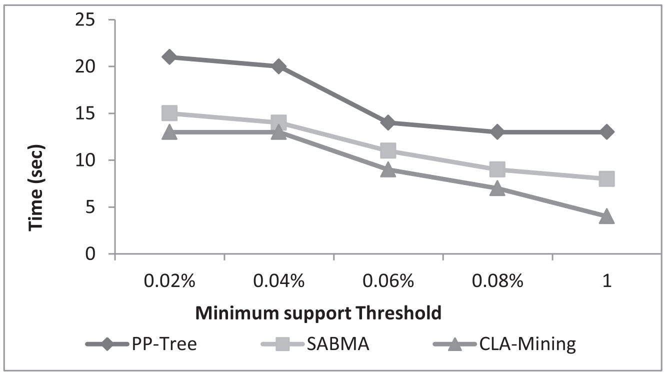

Fig. 15. Dataset retail.

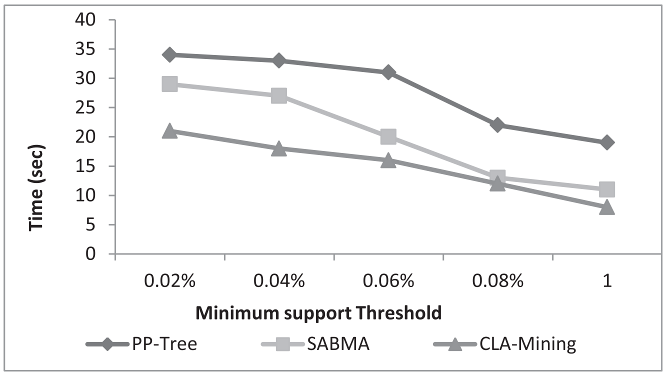

Fig. 16. Dataset pumsb.

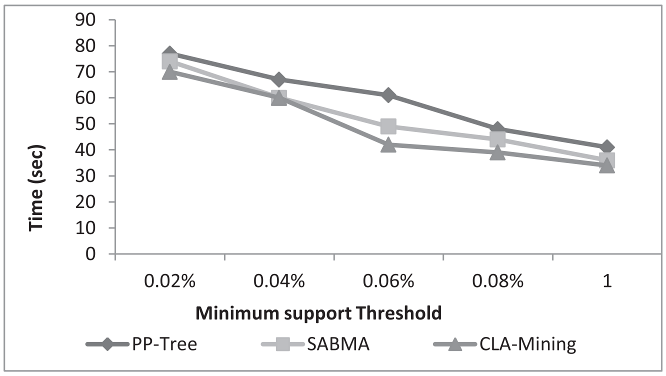

Fig. 17. Dataset Accidents.

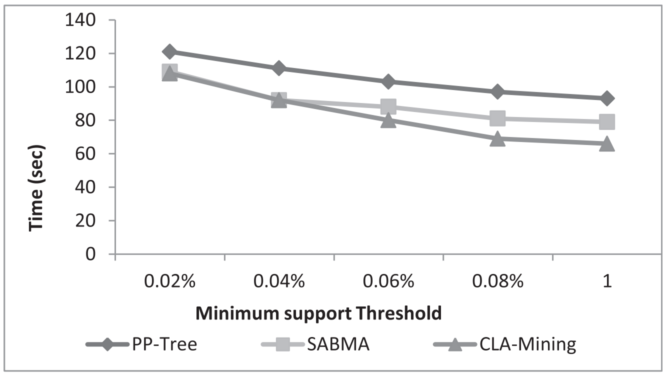

Fig. 18. Dataset kosarak.

5. Conclusion

One of the important issues in data mining is mining the frequent itemsets of transaction databases. Regarding the algorithms and techniques presented in this context, the researchers presented an algorithm which uses cellular automata to mine the frequent itemset at the more efficient process time. In this paper a novel CLA based distributed frequent itemset mining was represented which performed on the constructed proximity list for the neighbors' iteration for each cell of CLA and then applied this list for mining the frequent itemsets. Then, the presented method was compared to one of the best previously constructed methods (SABMA) which is a systolic array based parallel method for mining frequent itemsets. The experimental results revealed that our CLA-based mining method performs much faster than SABMA on several standard dataset. Combining CLA based mining techniques with the efficient structures like FP-Tree and using them for mining other types of patterns, such as sequential patterns, can be studied as future works. CLA based mining methods can also be used to extract interesting patterns from uncertain databases.

References

Agarwal, R., Aggarwal, C. C., & Prasad, V. V. V. (2001). Tree projection algorithm for generation of frequent itemsets. *Parallel and Distributed Computing, 61*(3), 350–371.

Agarwal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the ACM-SIGMOD international conference on management of data (SIGMOD'93), Washington DC*.

Brin, S., Motwani, R., Ullman, J. D., & Tsur, S. (1997). Dynamic itemset counting and implication rules for market basket analysis. In *ACMSIGMOD international conference on management of data, Tucson*.

Cheung, D. W., Han, J., Ng, V., & Wong, C. Y. (1996). Maintenance of discovered association rules in large an incremental updating technique. In *Proceeding of the international conference on data engineering, New Orleans*.

Dong, J., & Han, J. (2007). BitTableFI: An efficient mining frequent itemsets algorithm. *Knowledge Based Systems, 20*(4), 329–335.

Esnaashari, M., & Meybodi, M. R. (2007). Irregular cellular learning automata and its application to clustering in sensor networks. In *Proceedings of 15th conference on electrical engineering (15th ICEE), Tehran, Iran*.

FathiNavid, A. H., & Aghababa, A. A. (2012). Irregular cellular learning automata-based method for intrusion detection in mobile ad hoc networks. In *Proceeding of the 51st international FITCE (Federation of Telecommunications engineers of the european community) Congress*.

Grahne, G., & Zhu, J. (2003). Efficiently using prefix-trees in mining frequent itemsets. In *Proceeding of the ICDM'03 international workshop on frequent itemset mining implementations, Melbourne*.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In *ACM-SIGMOD international conference on management of data (SIGMOD'00), 1-12, Dallas*.

Hu, T. W., & Yeh, Y. H. (2014). Discovering valuable frequent patterns based on RFM analysis without customer identification information. *Knowledge Based Systems, 61*, 76–88.

Javed, A., & Khokhar, A. (2004). Frequent pattern mining on message passing multiprocessor systems. *Distributed and Parallel Databases, 16*, 321–334.

Liu, G., Lu, H., Lou, W., & Yu, J. X. (2003). On computing, storing and querying frequent patterns. In *Proceeding of the ACM SIGKDD international conference on knowledge discovery and data mining, Washington DC*.

Liu, J., Pan, Y., Wang, K., & Han, J. (2002). Mining frequent item sets by opportunistic projection. In *Proceeding of the ACM SIGKDD international conference on knowledge discovery in databases, Edmonton*.

Park, J. S., Chen, M. S., & Yu, P. S. (1995). An effective hash-based algorithm for mining association rules. In *Proceedings of the ACM SIGMOD international conference on Management of data 175–186*.

Pei, j (2002). *Pattern-growth methods for frequent pattern mining*. PhD Thesis.

Rezaei, N., & Shiri, M. E. (2013). An efficient algorithm for frequent items based on Bit Table mapping. In *1st National innovation conference on computer engineering and information Technology, Mazandaran*.

Ristoski, P., & Paulheim, H. (2016). Semantic web in data mining and knowledge discovery: A comprehensive survey. *Web Semantics: Science, Services and Agents on the World Wide Web, 36*, 1–22.

Savasere, A., Omiecinski, E., & Navathe, S. (1995). An efficient algorithm for mining association rule in large databases. In *Proceeding of the 21st international VLDB conference, Zurich, Switzerland*.

Sohrabi, M. K., & Akbari, S. (2016). A comprehensive study on the effects of using data mining techniques to predict tie strength. *Computers in Human behavior, 60*, 534–541.

Sohrabi, M. K., & Barforoush, A. A. (2012). Efficient colossal pattern mining in high dimensional datasets. *Knowledge Based Systems, 33*, 41–52.

Sohrabi, M. K., & Barforoush, A. A. (2013). Parallel frequent itemset mining using systolic arrays. *Knowledge Based Systems, 37*, 462–471.

Sohrabi, M. K., & Ghods, V. (2014). Top-down vertical itemset mining. In *Proceedings of the SPIE 9443 sixth international conference on graphic and image processing*.

Sohrabi, M. K., & Marzooni, H. H. (2016). Association rule mining using new FP-linked list algorithm. *Journal of Advances in Computer Research, 7*(01), 23–34.

Song, W., Yang, Y., & Xu, Z. (2007). Index-BitTableFI: An improved algorithm for mining frequent itemsets. *Knowledge Based Systems, 20*(4), 329–335.

Toivonen, H. (1996). Sampling large databases for association rules. In *Proceeding of the international conference on very large data bases, Bombay*.

Wang, Q., Davis, D. N., & Ren, J. (2016). Mining frequent biological sequences based on bitmap without candidate sequence generation. *Computers in Biology and Medicine, 69*, 152–157.

Wu, X., Fan, W., Peng, J., Zhang, K., & Yu, Y. (2015). Iterative sampling based frequent itemset mining for big data. *International Journal of Machine Learning and Cybernetics, 6*(6), 875–882.