Caryntia Kharkongor
Bhabesh Nath

SET REPRESENTATION FOR RULE-GENERATION ALGORITHMS

Abstract *The task of mining association rules has become one of the most widely used discovery pattern methods in knowledge discovery in databases (KDD). One such task is to represent an item set in the memory. The representation of the item set largely depends on the type of data structure that is used for storing them. Computing the process of mining an association rule impacts the memory and time requirements of the item set. With the constant increase of the dimensionality of data and data sets, mining such a large volume of data sets will be difficult since all of these item sets cannot be placed in the main memory. As the representation of an item set greatly affects the efficiency of the rule-mining association, a compact and compressed representation of the item set is needed. In this paper, a set representation is introduced that is more memory- and cost-efficient. Bitmap representation takes 1 byte for an element, but a set representation uses 1 bit. The set representation is being incorporated in the Apriori algorithm. Set representation is also being tested for different rule-generation algorithms. The complexities of these different rule-generation algorithms that use set representation are being compared in terms of memory and time of execution.*

Keywords item set, item set representation, Apriori algorithm, rule-generation algorithm, data set, set representation, bitmap, memory, time

Citation Computer Science 23(2) 2022: 205–224

1. Introduction

Association-rule mining is considered to be one of the most important aspects in the data-mining domain. The process of the mining of association rules between items was introduced that generate rules for items in large databases [1]. The main objective is to find the interesting patterns, correlations, and associations between data items in such transactions. These data items are stored in the memory for the process of mining. In order to store the items in the memory, a data structure is needed for representing the items. One such data structure is an array that consumes 4 bytes for an item (if it is an integer) [11]. If the data set size is 100, then the number of consumed bytes will be 100 x 4 = 400 bytes. Depending on the size of the data set, the number of bytes for storing the items in the memory will also vary. When a data set is large, the process of mining the item sets will also consume more time and memory [40] [41]. In this paper, set representation is used for representing item sets in the memory. This representation is more compressed and efficient than the other representation, as it consumes only 1 bit for each item. With a compressed and concise item-set representation, additional storage may not be required. This paper consists of the following sections:

- representation for item set called set representation;
- data is mined using Apriori algorithm;
- different rule-generation algorithms are used for generating rules.

In this paper, the computation operation is accomplished by using Boolean operators; this will speed up the mining process. Hence, the process of mining and generating rules will also be rapid and fast. Then, rule-generation algorithms such as Agarwal's algorithm, Srikant's algorithm, and NBG's algorithm will be tested using set representation with varying item set sizes.

2. Background Concepts

The database (D) consists of a set of transactions where each refers to a collection of items that occur together in the same transaction. Each transaction is unique and is identified by using an identifier (TID).

- Item: refers to an element that occurs in a given data set. For a given data set D, the formal representation of an item set is represented by I $=\{i_1,i_2,i_3,...,i_n\}$, where each $\{i_1,i_2,i_3,...,i_n\}$ element denotes an item.
- Item set: represents a group of items that occur together. An item set is identified as a set of $\{i_1,i_2,i_3,...,i_n\}$ items that occur together. For example, I$=\{i_2,i_4,i_6\}$ is an item set.
- Transactional data set: consists of a set of items that happen together in each transaction; it is represented by a unique identification.
- Support count: defines the frequency of an item set that occurs throughout an entire database.

- Confidence: refers to the number of times an item X occurs (an item T also exists that occurs in the same transaction).
- Frequent item set: an item set whose support count is more than a threshold value [1] [3] [2].
- Association rule: A rule is known as an association rule if it in the form of $X \rightarrow Y$, where X and Y are the set of items, $X \rightarrow Y$ (where X and Y are the subsets of I), and $X \cap Y = 0$ [1] [12].

2.1. Finding association rules

Association-rule mining is probabilistic. Rule $X \rightarrow Y$ holds for a minimum confidence, but rule $X + Z \rightarrow Y$ does not necessarily mean that the rule is valid since it may not have the required minimum threshold. Similarly, the rule is not transitive. This means that rule $X \rightarrow Y$ and $Y \rightarrow Z$ may have held for a minimum threshold, but it does not mean that $X \rightarrow Z$ will hold [1] [3] [2]. The problems with association-rule mining can be categorized into two parts:

- Generating all of the possible combinations of the items that have support that is greater than or equal to the support threshold. These item sets are called frequent item sets.
- Generating rules from a given data set that have support and confidence that are greater than the minimum threshold value [41].

Other additional constraints are needed for checking the generation of rules; these are as follows:

- Syntactic constraints
  These are the constraints that involve restricting the antecedent and the consequent of a rule. A combination of the structures of the rule may require that some predefined X appears in the consequent part, while some of the structures of the rule appear in the antecedent part. In a check-out basket transaction, for example, a user specifies an additional restriction on the rule such that the antecedent part may contain only ‘milk’ and ‘cheese’ [1].
- Support constraints
  The support constraint is of a primary importance for the generation of candidate item sets; this gives the statistical significance and level of interest of a rule. If the support count is not high, then the importance of the rule decreases. This rule is not important enough for consideration, and it is less preferred. If there are many generated rules that satisfy the minimum support count, then the more applicable the rule will be. This is critical from a business point of view as well as statistically [1].
- Confidence constraints
  This is the metric that defines a rule’s strength; it indicates how the degree of the items in an antecedent is related to the items in the consequent part. Using the confidence constraint, an association rule is considered to be either irrelevant or interesting. Rules that are greater than a minimum confidence are considered

to be more relevant than those with a confidence count that is lower than the minimum value [1]. There are other ways of optimizing rule generation, such as reducing the number of checks at run time [30].

3. Related Work

Over the years, many algorithms have been developed that have enhanced the process of mining in many ways (such as faster execution and less space consumption). The algorithms that have been produced are quantitative association rules like [2], generalized rules such as [1], and sequential patterns such as [10] [41]; big graphs [9] have also been introduced. Improved Apriori algorithms have been introduced (such as [18]). The NOV-CFI algorithm is an algorithm that deals with closed frequent item sets [32]. Mining algorithms have been used in many applications such as providing guidance to students [37] and helping students for selecting courses [26]. In [1], the authors addressed how to mine patterns. An idea called a local dependency item set was discovered; it considers a user's level of interest from a local view on how the mining of patterns can be done. Even when the support count is low, the patterns that are associated with such items are not deleted. However, a disadvantage of this algorithm is apparent when the size of a data set is large. Storing item sets in the memory will not only require more time but also increase the computational cost. An algorithm called CRAR (the concurrent relational association-rule-mining algorithm) is used to discover RAR in large data sets. However, it is a CPU-bound algorithm that requires higher-level processors or multiple processors to increase its computation. Using this algorithm, a lot of resources are needed for computation, which increases computation time and costs [16]. ACAR (atomic class association-rule mining) is applied to prune to specific rules (redundant and inconsistent rules); it solves the problem of rules that are defective that are caused due to an imbalance. This algorithm divides a data set into classes (i.e., true and false sets); then, rules will be generated for each. Although it predicts any defect on an association, it also consumes more time if a data set is very large [35]. To reduce the computational cost and the number of tasks, the following criteria need to be considered:

- To reduce the number of passes over a database.
  An algorithm such as FP is a major improvement as compared to the traditional Apriori algorithm (where the number of passes is only two). Frequent item sets are generated without producing candidate item sets by using only two passes of a database [21].
- To reduce the sample of a database.
  [42] used sampling for representing an association-rule-mining algorithm; this algorithm has two phases. Those rules that are not frequent in the first phase are used for constructing the set in the second phase. These rules may not be frequent in the first phase but are frequent in the entire database. [15] introduced a novel approach called sampling error estimation that identifies the sample size of mining association rules. It gives an appropriate size without generating association

rules. The generation of rules can be efficiently produced by executing a sample size to obtain the result. However, obtaining a particular size is the main issue for data-mining tasks.

- To use parallelization.
  Parallelization has been known to increase the speed of the execution of a process. The association-rule-mining technique has also adapted for this approach. For example, [14] introduced an algorithm called FPM (fast parallel mining) that is used for parallelizing Apriori. Then, other techniques such as DDM (Apriori-based D-ARM) were also introduced [33]; this uses a tree structure and avoids the generation of candidate item sets. It uses constraint during the mining process and generates only those rules that have high levels of interest to a user. PaMPaHD is a map reduce-based algorithm that mines frequently closed item-set mining; it is used for high-dimensionality data sets [6]. [7] has provided a promising direction for research in parallelization for item-set mining problems.
- Adding additional constraints on the methods of generating an algorithm. Some users want to limit the structure of any generated rules. [17] introduced a category-based algorithm; this reduces complexity by passing the subsets of item sets.

4. Apriori algorithm

An algorithm processes data and executes a task according to ideas and methods [44]. One such algorithm is the Apriori algorithm; it forms the foundation of association-rule generation, which was introduced by Agarwal. The algorithm makes multiple scans over a database, and the support count of each item set is calculated. These item sets are called candidate item sets. A count is associated with each item set, which maintains the frequency of the item set. Each item set that has support that is greater than the minimum support is added to the next level, which is then used to find the next candidate item set. An item set that is smaller than the threshold is not extended further, and an item set whose support is greater than the minimum support is considered to be large. Apriori works on the principle that the subsets of any frequent item set will also be frequent. So, if a subset of an item set is not large, then the item set will not be frequent [1] [4] [3]. During a pass, it is possible that a candidate item set can be discarded if it is not large. This is a pruning process that saves both memory and computational tasks. Item sets that are not large are deleted, and only those item sets that have support that is greater than the minimum support are kept. These item sets are used to generate new item sets. The algorithm stops when no item set has support that is greater than the minimum support [1] [4] [3].

5. Rule-generation Algorithm

After generating candidate item sets, rules are produced based on a confidence threshold. These rules are generated from frequent item sets whose confidence values are greater than or equal to the minimum confidence level. Those rules that have a confidence level that is above the threshold level are considered for generating rules.

Some algorithms are used for extracting rules from item sets. The different algorithms that are introduced for rule generation are as follows:-

- Agarwal's Algorithm
  Agarwal introduced the first algorithm for generating rules from frequent item sets. A rule is produced by using one item or element in the consequent part. Rule $X \rightarrow Y$ is defined for a frequent item set of $I = \{I_1, I_2, I_3, \ldots, I_k\}$ $k \geq 2$, where the antecedent of the rule is subset X of I in such a way that X has only k-1 items and the consequent has I - X items. The rule is then generated with a confidence value that is greater than the minimum threshold. The only disadvantage of this rule is that it is not capable of generating all of the rules. A rule generates only n rules for n frequent item sets, although there are 2^n – two rules [1].
- Srikant's Simple Algorithm
  This algorithm is a generalized form of the previous algorithm. In this algorithm, the consequent part is not limited to one item or element. First, all of the non-empty subsets from the frequent-item sets are found first. For each subset (say x), this is checked; then, the rule is generated in the form of $x = (l\text{-}x)$ if the confidence value is greater than the specified threshold. Although the algorithm can generate all of the rules, it also wastes time when checking the rules. For example, if item set $WXYZ$ is used for rule generation, then subsets WXY, WX, and W will lead to check the following rules: $WXY \rightarrow Z$, $WX \rightarrow YZ$, and $W \rightarrow XYZ$. If rule $WXY \rightarrow Z$ has a confidence value that is below the minimum threshold, then the confidence value of $WX \rightarrow YZ$ will not be more than the threshold level. However, this algorithm will check for a second rule; thus, it will waste a lot of time in redundancy checking [2].
- Srikant's Faster Algorithm
  This algorithm is an improvement on the previous algorithm, as it eliminates the unnecessary time for redundancy checking. If c is a subset of a, then the support of c cannot be more than the support count of a. Hence, the confidence of rule $a = (l \rightarrow a)$ cannot be lower than that of rule $c = (c \rightarrow a)$. The algorithm works on the same principle as that of the downward closure property in frequent item sets. This property states that the subsets of a frequent item set will also be frequent. The only drawback of this algorithm is that, for a different antecedent, it generates the same consequent several times and, thus, wastes time. For example, if $A \subseteq B$ generates rules for A, all of the consequent items will be generated. The task will be repeated for B even though many of these have been generated already [2].
- Faster Rule-generation Algorithm
  This algorithm is also known as NBG's algorithm, which overcomes the limitation of Srikant's algorithm. It can generate all of the rules that satisfy the minimum

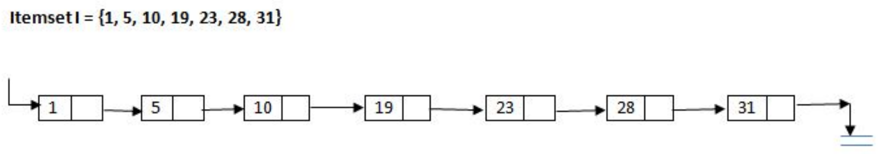


Figure 1. Linked-list representation

confidence and avoids the unnecessary time for checking redundant rules. This algorithm does not generate subsets; thus, the time will be significantly reduced in the rule generation. It will only generate rules for those item sets that are already stored in the memory. Thus, the algorithm not only saves time but also the memory requirements for storing item sets [19].

6. Data Structure Used in Association-rule Mining

An item set in association-rule mining uses different data structures for storing the items in the memory. Some of the data structures that are used for representing the item set are as follows:

- Linked list: used to store data of similar types. The advantage of using a linked list is that the size of the linked list is not fixed. The items in a linked list can be added dynamically. A linked list has two parts: one for storing data, and the other for storing a pointer. Items are added sequentially; so, each element requires 8 bytes (4 + 4) if integers are used in the linked list. Some of the algorithms that use linked-list representation are the transaction mapping algorithm [38], trie-based Apriori [13], and the FP-growth algorithm [22]. Suppose item set I={1,5,10,19,23,28,31} uses a linked list for a 32-bit system; the total amount of memory consumption would total 7 x (2 x Integers) = 56 bytes (as shown in Figure 1).
- Array: a data structure that is used for storing items of the same type. The size is fixed, and the items are indexed. Items can be inserted contiguously, and each item is identified by a unique index. The algorithms that use arrays for representation are the H-Mine algorithm [31] and a variation of the FP-tree algorithm [20]. A more compact representation was also discussed in [11]. To represent item set I={1,5,10,19,23,28,31}, memory that uses array requires 32 x integers for a 32-bit system (which is shown in Figure 2). If integers are used, the total memory requirement would be 32 x 4 bytes = 128 bytes.
- Bitmap: this maps domains to bits where items are represented by using 1 byte [5]. Each item in an item set is marked by '1' if the item occurs in the item set; if not, then it is marked by '0.' Each item takes 1 byte for storing each element in an item set [36]. The *HVSM* algorithm [39] and *LCM* Ver. 3 [43] use bitmaps

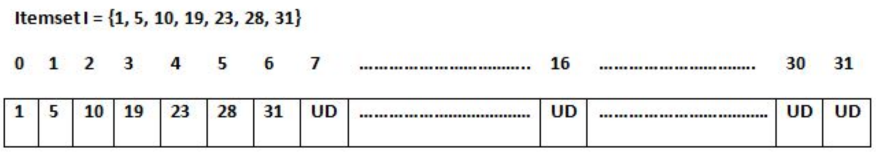

Itemset I = {1, 5, 10, 19, 23, 28, 31}
0 1 2 3 4 5 6 7 .... 16 .... 30 31
1 5 10 19 23 28 31 UD .... UD .... UD UD

Figure 2. Array representation

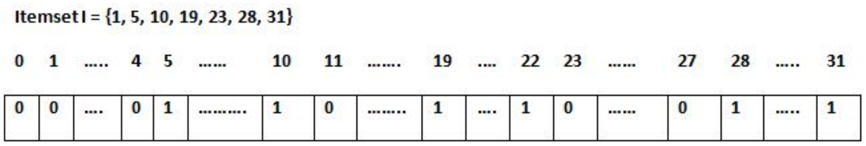


Figure 3. Bitmap representation

to represent an item set. Another mining algorithm (known as sequential pattern mining) utilizes a bitmap representation [43]. $VIPER$ is another algorithm that employs bit vectors [33]. Another bit-wise parallel algorithm was also introduced for representing item sets in [29]; in this, items are stored contiguously in the memory. Using bitmap representation, item set I ={1, 5, 10, 19, 23, 28, 31} requires 32 x 1 byte = 32 bytes for a 32-bit system (shown in Figure 3).

7. Motivation

Data keeps on growing exponentially with increases in the rates of data in streaming, the stock market, social media, etc. There is a need to understand the complexity of this data and signify the opportunities that the data brings [23]. There are many algorithms that have been introduced for mining data in order to cope with the expensive computational task. There have also been many discussions on how to handle the memory management of item sets [34] [8]. One of the main challenges is to store item sets in the memory [28]. A database that contains large transactions will eventually generate more candidate item sets. Sometimes, storing these item sets in the memory becomes infeasible [45]. Representing this item set in the memory is the crucial part. Item sets can be represented by using the three data structures (as discussed in Section 6). The three data structures that are discussed are linked lists, arrays, and bitmaps. An array data structure needs 4 bytes to store each item, while a linked list consumes 8 bytes. In comparison with all three of the data structures, bitmap is better since it consumes only 1 byte for each item (as discussed in [29] and [33]). The main disadvantage of the representation is that it can store up items to a maximum size of 32-bit item sets. It performs efficiently for small databases but not for large ones. In this paper, a set representation is represented for an item set in a more concise representation. In this representation, the size of an item set is

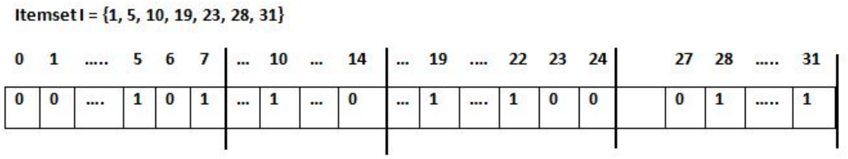


Figure 4. Set representation

the maximum size of the domain. An item set can be represented in the form of a bit, where the presence of an element is represented by '1' and the absence of one by '0.' In this way, only 1 bit is used for representing an element in an item set. This will eventually reduce memory consumption and increase the efficiency of the mining algorithm.

8. Set Representation

Set representation takes 1 bit for each item. The items are stored successively one item after another. In [27], the authors used a model called a machine-oriented data model for representing subsets. It uses Boolean operations for speeding up the data-mining process. However, the size of any item set that is taken under consideration is only 32 bits. Moreover, the size of an item set may never be less than 32 bits. Another representation of an alternative way of representing data is by using strings of bits. The authors proposed an algorithm that used BitSet computations between item sets. They applied fuzzy rules in real and synthetic databases that used restriction for obtaining a summarized result. Suppose a set representation takes only 32 x 1 bit = 32 bits or 4 bytes in item set $I = \{1, 5, 10, 19, 23, 28, 31\}$ (which is given in Figure 4). In set representation, each item in a transaction is represented by a bit (unlike bitmap, which requires 1 byte for each element). If an item is present in the transaction, then '1' is marked; otherwise, '0' is indicated. The maximum size of the item set will be the maximum size of the attributes or the length of the transaction. The size of the item set does not have a limit. For a 32-bit system, an item set whose size is greater than 32 bits cannot be represented. An item set whose size is greater than 32 bits can be represented by using set representation. If the size of an item set is more than 32 bits, the first item set's size is 32 bits, and the next one will again be 32 bits for a 32-bit system. A data set is first pre-processed and then represented using set representation. The maximum size of an item set will be the size of the attributes. If the maximum size of an item set is 64, then two item sets are needed (where the first one contains the first 32 bits and the second comprises the next 32). An item is represented by marking '1' on an item set if the item is present and '0' if it is absent in a transaction. Each position on a item set represents an item where the first item set contains the first 32 items, the second represents the next 33 items, and so on. Suppose the 35^{th} item is present in a transaction; then, a second item set is used where the 3^{rd} position is marked as 1.

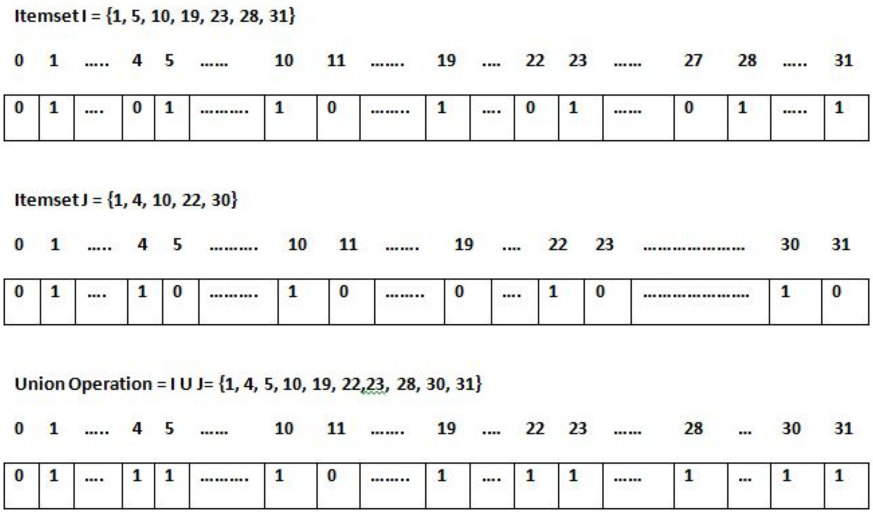


Figure 5. Union Operation

8.1. Set operations

In association-rule mining, operations are performed using item sets. How an item set is stored is crucial, as it affects the way an operation is performed. Set representation uses Boolean operators for its computations. This process speeds up computational tasks (which is needed during the mining process). Although bitmap uses the same Boolean operations for their computations, there is a difference between a set and a bitmap. Each item consumes 1 byte for storage using bitmap representation, while set representation consumes only 1 bit. The operations that are needed to be performed are as follows:

- Union operation: combines those items that are present in both first and second item sets. An OR operation is needed to find the union operation between two item sets; the complexity of such an operation is 0(1). An example of a union operation is shown in Figure 5.
- Intersection operation: contains the common items that are present in both first and second item sets. An AND operation is used to find the intersection between the two item sets and takes an order of 0(1) complexity (which is shown in Figure 6).
- Set difference: consist of the item/s that is/are present in the first item set but not in a second one. A bit-wise operation is used to find the set difference of the two item sets, which takes only a 0(1) complexity (shown in Figure 7).
- Superset: checks whether an item set is a a superset of another item set. This operation is performed using a bit-wise operation.

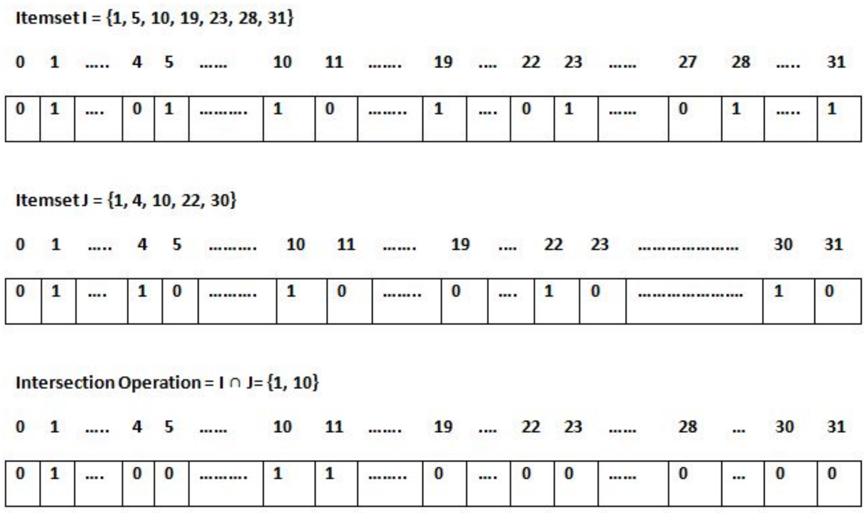


Figure 6. Intersection Operation

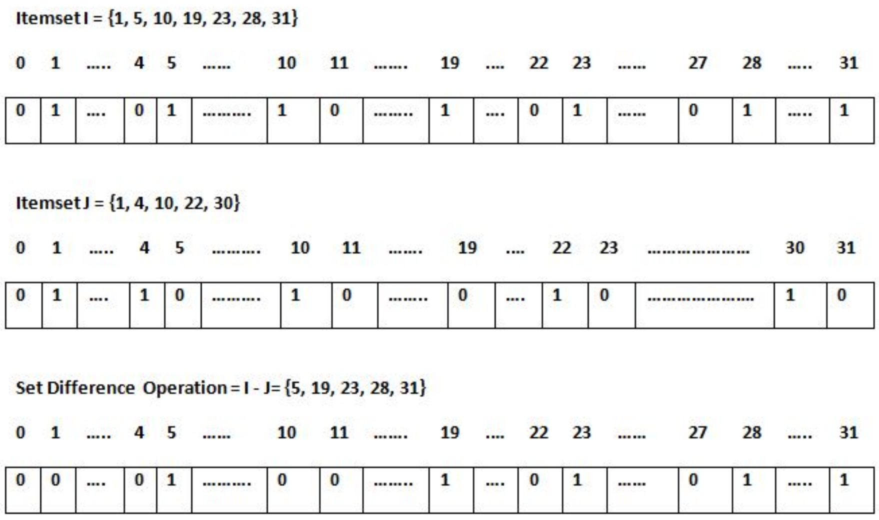


Figure 7. Set Difference Operation

- Subset: similar to a superset operation that checks whether a particular item set is a subset of another item set.
- Membership: checks whether an item is a member of an item set or not.

9. Apriori Algorithm using Set Representation

In this paper, the item-set representation called "set representation" has been incorporated in rule-mining algorithms. In set representation, items are represented by using a horizontal layout where the items' IDs are stored. The size of an item set in this representation will be the maximum size of the item. Each item is labeled '1' if the item is present in an item set and '0' if the item is absent from the transaction. Each item consumes only 1 bit for each item. For example, item set I={1, 10, 19, 23, 31} is represented by set representation. The maximum size of the item set will be 32 because the maximum item number is 32. If each item takes 1 bit for storing, then the total memory consumption for storing will be 32 x 1 bit = 32 bits or 4 bytes. The basic rule generation that is called for in the Apriori algorithm is implemented by using set representation. The generated item sets are stored in the memory using only 1 bit per element instead of consuming 4 bytes per element. Thus, this saves memory storage for representing an item set in the memory. Rule-generation algorithms have different operations, such as membership, superset, subset, union, set difference, and intersection operations. All of these operations take only O(1), which improves the time complexity. Hence, mining algorithms will improve their efficiency in regards to time and memory when using set representation [25].

10. Experimental Evaluation

The memory consumption of using different data sets has been discussed. It can be seen that the memory consumption is better in an array than it is in a linked list. Array representation has been used for a comparison with set representation. The rule-generation algorithms that have been discussed (such as Agarwal's algorithm, Srikant's simple algorithm, Srikant's faster algorithm, and NBG's algorithm) have been implemented using both array and set representation. These algorithms have been tested against three data sets that have 50 attributes with 1000, 5000, and 20,000 transactions, respectively. With varying support counts = 1, 2.5, 5, and 10 %, the two representations showed varying performance levels. The following tables show the execution times and space requirements by the two different item-set representations. The three data sets were used for testing the performance of both the array and set representations. These data sets can be openly accessed through the following link: [24]. The three data sets were generated randomly to be used for these experiments. The number of attributes remained the same for all three data sets, but the sizes of the data sets varied. The data sets that were used in this experiment are as follows:

- First data set: 1000 transactions with 50 attributes. The results of the time and memory requirements for the four different mining algorithms using this data set

Table 1
Algorithms using array and set representation for 1000-transaction data set with 1% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	45,327.7	22,835	41,127.5	14,148
Srikant's 2nd Algorithm	40,855.6	21,848	38,547.9	13,774
Agarwal Algorithm	38,383.6	21,484	36,660.7	13,577
NBG's Algorithm	28,227.6	21,105	27,511.3	13,416

Table 2
Algorithms using array and set representation for 1000-transaction data set with 2.5% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	35,362.7	21,845	33,366.3	15,746
Srikant's 2nd Algorithm	37,854.5	21,559	32,547.9	14,223
Agarwal Algorithm	34,150.5	21,244	31,392.3	13,896
NBG's Algorithm	28,210.6	21,044	26,166.4	13519

Table 3
Algorithms using array and set representation for 1000-transaction data set with 5% support

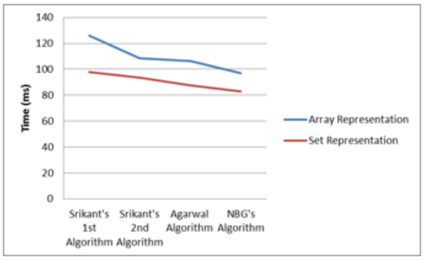
Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	21,745.1	21,575	15,578.1	14,746
Srikant's 2nd Algorithm	19,311	21,248	15,423.33	13,823
Agarwal Algorithm	17,330.8	21,192	15,352.5	13,296
NBG's Algorithm	17,077.5	21,140	15,226.4	13,360

are shown in Table 1 (for 1% support), Table 2 (2.5%), Table 3 (5%), and Table 4 (10%).

- Second data set: 5000 transactions with 50 attributes. The four algorithms are implemented using both set and array representations by measuring the memory and time consumption. Using this data set, the results are represented in Table 5 for 1% support, Table 6 (2.5%), Table 7 (5%), and Table 8 (10%).
- Third data set: 20,000 transactions with 50 attributes. The performance of the four algorithms can be observed by measuring the space and time that are required for their execution. The data set is tested against the algorithms using set and array representations. The results are shown in Table 9 for 1% support, Table 10 (2.5%), Table 11 (5%), and Table 12 (10%).

Table 4
Algorithms using array and set representation for 1000-transaction data set with 10% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	125.92	21,245	98	13,374
Srikant's 2nd Algorithm	108.674	21,259	93.736	13,310
Agarwal Algorithm	106.449	21,136	87.788	13,290
NBG's Algorithm	97	21,024	82.903	13,284



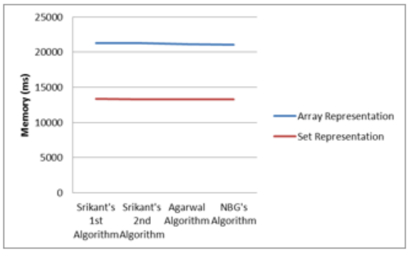


Figure 8. Time and memory requirements for 1000-transaction data set with support count = 10%

Table 5
Algorithms using array and set representation for 5000-transaction data set with 1% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	911,524	26,004	825,072	19,876
Srikant's 2nd Algorithm	853,748	24,652	786,289	18,620
Agarwal Algorithm	817,218	23,233	748,117	16,363
NBG's Algorithm	750,427	22,900	588,567	15,485

Table 6
Algorithms using array and set representation for 5000-transaction data set with 2.5% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	909,012	25,374	828,970	18,276
Srikant's 2nd Algorithm	838,804	23,468	788,277	17,078
Agarwal Algorithm	716,551	22,901	680,261	16,163
NBG's Algorithm	624,141	22,134	526,363	14,985

Table 7
Algorithms using array and set representation for 5000-transaction data set with 5% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	367,680	23,645	330,607	15,390
Srikant's 2nd Algorithm	334,243	22,759	326,194	14,389
Agarwal Algorithm	330,514	21,844	323,074	14,171
NBG's Algorithm	323,921	21,086	317,121	13,411

Table 8
Algorithms using array and set representation for 5000-transaction data set with 10% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	1001.98	21,845	888.83	14,646
Srikant's 2nd Algorithm	995.78	21,244	700.98	14,123
Agarwal Algorithm	713.755	21,099	408.9	13,596
NBG's Algorithm	625	20,286	300.385	13,299

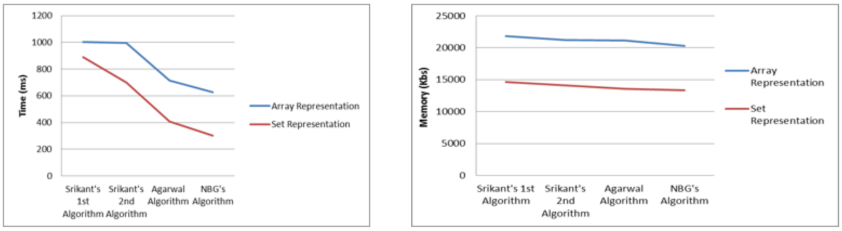


Figure 9. Time and memory requirements for 5000-transaction data set with support count = 10%

Table 9
Algorithms using array and set representation for 20,000-transaction data set with 1% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	951,524	29,996	885,072	20,756
Srikant's 2nd Algorithm	863,791	25,947	752,006	19,096
Agarwal Algorithm	745,721	22,380	620,122	17,951
NBG's Algorithm	617,842	21,758	588,567	16,973

Table 10
Algorithms using array and set representation for 20,000-transaction data set with 2.5% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	877,161	26,374	828,970	19,876
Srikant's 2nd Algorithm	798,804	23,568	758,277	18,620
Agarwal Algorithm	696,651	23,001	680,261	16,363
NBG's Algorithm	654,141	22,647	526,363	14,485

Table 11
Algorithms using array and set representation for 20,000-transaction data set with 5% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	394,676	24,085	320,446	15,991
Srikant's 2nd Algorithm	354,243	23,159	319,806	15,125
Agarwal Algorithm	348,560	21,929	312,374	14,548
NBG's Algorithm	316,085	21,690	302,370	13,903

Table 12
Algorithms using array and set representation for 20,000-transaction data set with 10% support

Algorithms	Array Representation		Set Representation	
	Time (ms)	Memory (kb)	Time (ms)	Memory (kb)
Srikant's 1st Algorithm	1088.86	22,845	923.3	15,646
Srikant's 2nd Algorithm	1005.8	21,244	801.075	14,823
Agarwal Algorithm	944.876	21,132	501.09	14,096
NBG's Algorithm	863.61	21,044	415.385	13,419

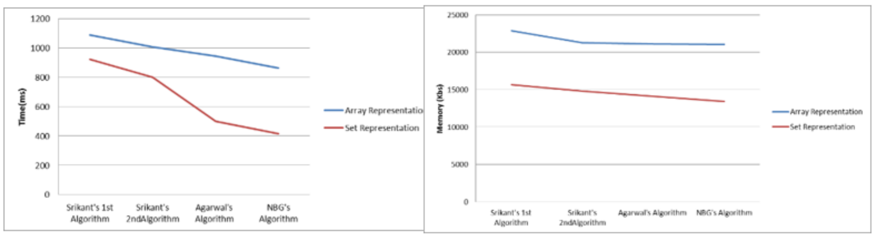


Figure 10. Time and memory requirements for 20,000-transaction data set with support count = 10%

As seen from the tables above, there were many candidate item sets that were generated with low support counts. So, the memory consumption was higher with the candidate item sets that had low support counts as compared to those candidate item sets with high support counts. The execution times of the algorithms were greater for those item sets with low support counts, whereas the execution times were lower for those with higher support counts. These tables also show that the representation of the item sets also affected the performances of these algorithms. Using array representation, the memory consumption was nearly double that of set representation. The execution time was better in set representation than it was in array representation. The reason for this was that set representation only stored 1 bit for each element whereas array representation needed 4 bytes. The different set operations (membership, subset, superset, union, and set difference) take only O(1) in BitSet representation. In this experiment, the system that was used was an Intel(R) Core(TM) i5 with 4 GB RAM, a 64-bit operating system. The experiments wer performed using both array and set representation. The execution times and memory consumption were calculated from the program itself. The memory consumption was calculated by taking the peak value of the memory at any instance of time.

11. Conclusion

This paper represents a more compact representation of item sets as compared to array representation. As seen from the results, the set representation provided better performance as compared to array representation. While increasing the sizes of the data sets, the number of generated candidate item sets will grow; the memory consumption for the item sets will also increase. The mining process for generating candidate item sets will also increase since the size of the data set will be large. Using set representation, only 1 bit is required for storing an item as opposed to array (which requires 4 bytes). From the tables, it can be observed that, for the same data-set size, the memory and time consumption were greater for array representation than they were for set representation. Using set representation, the efficiency of the mining algorithm was enhanced since the amount of memory that was required to store and perform the computation process was less. The time duration for the generation of candidates was also reduced; hence, the time for the generation of rules was also minimized. The mining algorithms will be more cost- and time-effective – especially when a data set is large. Thus, set representation gives a more compact representation that further increases the efficiency of rule-mining algorithms and their complexity in terms of memory and time.

References

[1] Agrawal R., Imieliński T., Swami A.: Mining association rules between sets of items in large databases. In: *Acm sigmod record*, vol. 22, pp. 207–216, ACM, 1993.

[2] Agrawal R., Mannila H., Srikant R., Toivonen H., Verkamo A.I., *et al.*: Fast discovery of association rules., *Advances in knowledge discovery and data mining*, vol. 12(1), pp. 307–328, 1996.

[3] Agrawal R., Srikant R., *et al.*: Fast algorithms for mining association rules. In: *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, pp. 487–499, 1994.

[4] Al-Maolegi M., Arkok B.: An improved Apriori algorithm for association rules, *arXiv preprint arXiv:14033948*, 2014.

[5] Antoshenkov G.: Byte-aligned bitmap compression. In: *Proceedings DCC'95 Data Compression Conference*, p. 476, IEEE, 1995.

[6] Apiletti D., Baralis E., Cerquitelli T., Garza P., Pulvirenti F., Michiardi P.: A parallel mapreduce algorithm to efficiently support itemset mining on high dimensional data, *Big Data Research*, vol. 10, pp. 53–69, 2017.

[7] Apiletti D., Baralis E., Cerquitelli T., Garza P., Pulvirenti F., Venturini L.: Frequent itemsets mining for big data: a comparative analysis, *Big Data Research*, vol. 9, pp. 67–83, 2017.

[8] Apiletti D., Baralis E., Cerquitelli T., Garza P., Pulvirenti F., Venturini L.: Frequent itemsets mining for big data: a comparative analysis, *Big Data Research*, vol. 9, pp. 67–83, 2017.

[9] Aridhi S., Nguifo E.M.: Big graph mining: Frameworks and techniques, *Big Data Research*, vol. 6, pp. 1–10, 2016.

[10] Ayres J., Flannick J., Gehrke J., Yiu T.: Sequential pattern mining using a bitmap representation. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 429–435, ACM, 2002.

[11] Barati M., Bai Q., Liu Q.: Mining semantic association rules from RDF data, *Knowledge-Based Systems*, vol. 133, pp. 183–196, 2017.

[12] Bittmann R.M., Nemery P., Shi X., Kemelmakher M., Wang M.: Frequent Itemset Mining without Ubiquitous Items, *arXiv preprint arXiv:180311105*, 2018.

[13] Bodon F.: A trie-based APRIORI implementation for mining frequent item sequences. In: *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pp. 56–65, ACM, 2005.

[14] Cheung D.W., Xiao Y.: Effect of data skewness in parallel mining of association rules. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 48–60, Springer, 1998.

[15] Chuang K.T., Chen M.S., Yang W.C.: Progressive sampling for association rules based on sampling error estimation. In: *Pacific-Asia conference on knowledge discovery and data mining*, pp. 505–515, Springer, 2005.

[16] Czibula G., Czibula I.G., Miholca D.L., Crivei L.M.: A novel concurrent relational association rule mining approach, *Expert Systems with Applications*, vol. 125, pp. 142–156, 2019.

[17] Do T.D., Hui S.C., Fong A.: Mining frequent itemsets with category-based constraints. In: *International conference on discovery science*, pp. 76–86, Springer, 2003.

[18] Gao F., Khandelwal A., Liu J.: Mining Frequent Itemsets Using Improved Apriori on Spark. In: *Proceedings of the 2019 3rd International Conference on Information System and Data Mining*, pp. 87–91, 2019.

[19] Ghosh A., Nath B.: Multi-objective rule mining using genetic algorithms, *Information Sciences*, vol. 163(1-3), pp. 123–133, 2004.

[20] Grahne G., Zhu J.: Efficiently using prefix-trees in mining frequent itemsets. In: *FIMI*, vol. 90, 2003.

[21] Han J., Pei J., Yin Y.: Mining frequent patterns without candidate generation. In: *ACM sigmod record*, vol. 29, pp. 1–12, ACM, 2000.

[22] Han J., Pei J., Yin Y.: Mining frequent patterns without candidate generation. In: *ACM sigmod record*, vol. 29, pp. 1–12, ACM, 2000.

[23] Jin X., Wah B.W., Cheng X., Wang Y.: Significance and challenges of big data research, *Big Data Research*, vol. 2(2), pp. 59–64, 2015.

[24] Kharkongor C.: Google Drive, 2021. https://drive.google.com/drive/folders/11iOaDy5UYgiQJOQe6acSDNrpz1Og-Lgt?usp=sharing.

[25] Kharkongor C., Nath B.: Set Representation for Itemsets in Association Rule Mining. In: *2018 IEEE International Conference on Intelligent Computing and Control Systems (ICICCS)*, IEEE, 2018.

[26] Liang Y., Duan X., Ding Y., Kou X., Huang J.: Data Mining of Students' Course Selection Based on Currency Rules and Decision Tree. In: *Proceedings of the 2019 4th International Conference on Big Data and Computing*, pp. 247–252, 2019.

[27] Louie E., Young T.: Finding association rules using fast bit computation: Machine-oriented modeling. In: *International Symposium on Methodologies for Intelligent Systems*, pp. 486–494, Springer, 2000.

[28] Mehlhorn K., Sanders P.: *Algorithms and data structures: The basic toolbox*, Springer Science & Business Media, 2008.

[29] Nguyen T.T.: Mining incrementally closed item sets with constructive pattern set, *Expert Systems with Applications*, vol. 100, pp. 41–67, 2018.

[30] Ortin F., Garcia M., McSweeney S.: Rule-based program specialization to optimize gradually typed code, *Knowledge-Based Systems*, vol. 179, pp. 145–173, 2019.

[31] Pei J., Han J., Lu H., Nishio S., Tang S., Yang D.: H-mine: Hyper-structure mining of frequent patterns in large databases. In: *Proceedings 2001 IEEE International Conference on Data Mining*, pp. 441–448, IEEE, 2001.

[32] Phan H.: NOV-CFI: a novel algorithm for closed frequent itemsets mining in transactional databases. In: *Proceedings of the 2018 VII International Conference on Network, Communication and Computing*, pp. 58–63, 2018.

[33] Schuster A., Wolff R.: Communication-efficient distributed mining of association rules. In: *ACM SIGMOD Record*, vol. 30, pp. 473–484, ACM, 2001.

[34] Serrano D., Antunes C.: Condensed representation of frequent itemsets. In: *Proceedings of the 18th International Database Engineering & Applications Symposium*, pp. 168–175, ACM, 2014.

[35] Shao Y., Liu B., Wang S., Li G.: A novel software defect prediction based on atomic class-association rule mining, *Expert Systems with Applications*, vol. 114, pp. 237–254, 2018.

[36] Shenoy P., Haritsa J.R., Sudarshan S., Bhalotia G., Bawa M., Shah D.: Turbo-charging vertical mining of large databases. In: *ACM Sigmod Record*, vol. 29, pp. 22–33, ACM, 2000.

[37] Sodanil M., Chotirat S., Poomhiran L., Viriyapant K.: Guideline for Academic Support of Student Career Path Using Mining Algorithm. In: *Proceedings of the 2019 3rd International Conference on Natural Language Processing and Information Retrieval*, pp. 133–137, 2019.

[38] Song M., Rajasekaran S.: A transaction mapping algorithm for frequent itemsets mining, *IEEE transactions on knowledge and data engineering*, vol. 18(4), pp. 472–481, 2006.

[39] Song S., Hu H., Jin S.: HVSM: a new sequential pattern mining algorithm using bitmap representation. In: *International conference on advanced data mining and applications*, pp. 455–463, Springer, 2005.

[40] Srikant R., Agrawal R.: Mining generalized association rules, 1995.

[41] Srikant R., Agrawal R.: Mining sequential patterns: Generalizations and performance improvements. In: *International Conference on Extending Database Technology*, pp. 1–17, Springer, 1996.

[42] Toivonen H., *et al.*: Sampling large databases for association rules. In: *VLDB*, vol. 96, pp. 134–145, 1996.

[43] Uno T., Kiyomi M., Arimura H.: LCM ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In: *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pp. 77–86, ACM, 2005.

[44] Vöcking B., Alt H., Dietzfelbinger M., Reischuk R., Scheideler C., Vollmer H., Wagner D.: *Algorithms unplugged*, Springer Science & Business Media, 2010.

[45] Wu K., Otoo E.J., Shoshani A.: Optimizing bitmap indices with efficient compression, *ACM Transactions on Database Systems (TODS)*, vol. 31(1), pp. 1–38, 2006.

Affiliations

Carynthia Kharkongor
Tezpur University, Assam, India, caryn@tezu.ernet.in

Bhabesh Nath
Tezpur University, Assam, India, bnath@tezu.ernet.in

Received: 20.12.2020
Revised: 28.06.2021
Accepted: 13.09.2021