Chapter 3

Proposed Representation of Itemset

3.1 Motivation

Most of the researchers have focused their research on mining the itemsets to discover most of the researchers have focused their research on mining the itemsets for discovering the relationship between the itemsets in large datasets[28]. The dataset usually contains a huge number of transactions where each transaction consists of each item. Mining the itemset is challenging when the search space is large. If there are n number of distinct items, the search space will consist of 2^n. Hence, researchers have introduced various data structures and algorithms for optimizing the search space [29]. One solution is to implement the breadth-first search algorithm in the mining algorithm. One such example is the well-known Apriori Algorithm where the frequency of the items is achieved by scanning the dataset every time for different sizes of candidate itemset. Unfortunately, when the number of generated itemsets is large, the memory requirements to handle such itemsets are overwhelming, and infeasible for an Apriori-based algorithm to mine the itemsets on a single machine. Other approaches show that by increasing the minimum threshold support, the count of generated candidate itemsets will automatically reduce [69]. Secondly, current approaches tend to keep the output and runtime under control by increasing the minimum frequency threshold, automatically reducing the importance of the number of candidates and frequent itemsets. However, studies have revealed that the itemsets with low-frequency count are more interesting [57]. the relationship between the itemsets in large datasets[28]. The dataset usually contains a huge number of transactions where

each transaction consists of each item. Mining the itemset is challenging when the search space is large. If there are n number of distinct items, the search space will consist of 2^n. Hence, researchers have introduced various data structures and algorithms for optimizing the search space [29]. One solution is to implement the breadth-first search algorithm in the mining algorithm. One such example is the well-known Apriori Algorithm where the frequency of the items is achieved by scanning the dataset every time for different sizes of candidate itemset. Unfortunately, when the number of generated itemsets is large, the memory requirements to handle such itemsets are overwhelming, and infeasible for an Apriori-based algorithm to mine the itemsets on a single machine. Other approaches show that by increasing the minimum threshold support, the count of generated candidate itemsets will automatically reduce [69]. Secondly, current approaches tend to keep the output and runtime under control by increasing the minimum frequency threshold, automatically reducing the importance of the number of candidates and frequent itemsets. However, studies have revealed that the itemsets with low frequency. Most of the researchers have focused their research on mining the itemsets to discover the relationship between the itemsets in large datasets[28]. The dataset usually contains a huge number of transactions where each transaction consists of each item. Mining the itemset is challenging when the search space is large. If there are n number of distinct items, the search space will consist of 2^n. Hence, researchers have introduced various data structures and algorithms for optimizing the search space [29]. One solution is to implement the breadth-first search algorithm in the mining algorithm. One such example is the well-known Apriori Algorithm where the frequency of the items is achieved by scanning the dataset every time for different sizes of candidate itemset. Unfortunately, when the number of generated itemsets is large, the memory requirements to handle such itemsets are overwhelming, and infeasible for an Apriori-based algorithm to mine the itemsets on a single machine. Other approaches show that by increasing the minimum threshold support, the count of generated candidate itemsets will automatically reduce [69]. Secondly, current approaches tend to keep the output and runtime under control by increasing the minimum frequency threshold, automatically reducing the importance of the number of candidates and frequent itemsets. However, studies have revealed that the itemsets with low-frequency count are more interesting [57]. count are more interesting [57].

So in practice, to represent a set of the size of 200 elements, memory of 1600 bytes is used. However, the same 1600 bytes can be used to represent a set having 200 elements but from a universal set with cardinality 2^{32-1}, by simply assuming every element to be an integer. This representation may be useful for sets with smaller cardinality. Array representation can be used only for the finite set (a universal

set has a fixed number of elements). If the universal set size ≥ is greater than 256 to represent any random set coming from this, a minimum of 256 bytes (1 byte for each element) but this representation will fail to represent sets with a cardinality of more than 256. To handle this situation, every element must be considered an integer. In this case, domain size may be extended up to 2^{32}. If a set of maximum cardinality 300 is to be represented, then 1200 bytes will be used irrespective of the cardinality of the set that is stored at this moment. This representation will use the same amount of effective memory to represent the set irrespective of the elements present in the set.
If the bitmap representation is used to represent any set that is a subset of a universal set with the cardinality 300, only 300 bits are needed. These 38 bytes can be treated as a sequence of 38 bytes longer integers. These 300-bit long integers can be realized as an array of 38 characters or 10 system-given integers resulting in a total memory requirement of 40 bytes. Hence, to represent 300 elements set, the linked list will consume 2400 (i.e., 4+4 * 300) bytes, array representation will need 1200 (i.e., 4 * 300) bytes and bit representation will require 40 bytes. The rule mining algorithm attempts to find out the frequent itemsets that are a subset of all the items present in the dataset. Since the number of items in the dataset is finite, bit map representation can be used to represent any itemset. During the mining process, many set-theoretic operations like checking membership, union, intersection, difference, etc., have to be performed. If this operation can be designed on an integer of 40 bytes long, then the representation will benefit the mining process.

3.2 Description of Proposed Representation

The proposed itemset representation is represented in such a way that if an item is present in the itemset then the item is marked by '1' and if the item is not present in the itemset then the item is marked by '0'. Each item in the itemset takes one bit. If the maximum size of the domain set is 384, then a 384-bit long integer can be visualized as 12 system-given integers with a total memory requirement of 48 bytes. An example of an itemset I ={1, 3, 5, 12, 17, 30, 31} is represented with the help of a diagram in Figure 3-1. The set operations can be performed after transforming the dataset into the proposed itemset representation. The bitwise operators such as AND, NOT, and OR are used for performing the different set operations. The set operations are those operations used in the association rule mining process as follows:

A = {1, 3, 5, 12, 17, 30, 31}

0	1	2	3	4	5	.....	12	13	14	15	16	17	.....	30	31
0	1	0	1	0	1	.....	1	0	0	0	0	1	.....	1	1

Figure 3-1: Proposed itemset representation for the itemset I ={1, 3, 5, 12, 17, 30, 31}

A = {1, 3, 5, 12, 17, 30, 31}

0	1	2	3	4	5	.....	12	13	14	15	16	17	.....	30	31
0	1	0	1	0	1	.....	1	0	0	0	0	1	.....	1	1

B= {1, 3, 5, 13, 18, 29, 31}

0	1	2	3	4	5	.....	12	13	14	15	16	17	18	.....	29		30	31
0	1	0	1	0	1	.....	0	1	0	0	0	0	1	.....	1		0	1

A ∪ B = {1, 3, 5, 12, 13, 17, 18, 29, 30, 31}

0	1	2	3	4	5	.....	12	13	14	15	16	17	18	.....	29	30	31
0	1	0	1	0	1	.....	1	1	0	0	0	1	1	.....	1	1	1

Figure 3-2: Union operation of the itemset A = {1, 3, 5, 12, 17, 30, 31} and B= {1, 3, 5, 13, 18, 29, 31}.

- Union operation: is performed by using OR operation on the two itemsets. Each item is mapped to bit representation. The union operation is achieved when each item from the one itemset is OR'ed with the item from another itemset that takes only O(c). Suppose the first itemset contains A = {1, 3, 5, 12, 17, 30, 31} and the second set B= {1, 3, 5, 13, 18, 29, 31}. The resultant set now is {1, 3, 5, 12, 13, 17, 18, 29, 30, 31} after performing the union operation is shown in Fig 3-2.

- Superset and Subset operation: To check whether an itemset is a subset or superset of another itemset. An itemset A is the subset of B if the elements in A are elements of itemset B. For example, if itemset A = {1, 2} and itemset B ={1, 3, 4}, then itemset A can be considered as the subset of B since all the elements of A {1, 2} are in itemset B. The operation of checking subset and superset using bitwise operation is implemented.

- Intersection operation: is accomplished by performing AND operation between the two itemsets. The operation takes only O(c) time as it uses only bitwise operation instead of computing the process recursively to find the intersection between the two itemsets. The bitwise operation of computing intersection is shown as follows in Fig 3-3.

A = {1, 3, 5, 12, 17, 30, 31}

0	1	2	3	4	5	.....	12	13	14	15	16	17	.....	30	31
0	1	0	1	0	1	.....	1	0	0	0	0	1	.....	1	1

B= {1, 3, 5, 13, 18, 29, 31}

0	1	2	3	4	5	.....	12	13	14	15	16	17	18	.....	29	30	31
0	1	0	1	0	1	.....	0	1	0	0	0	0	1	.....	1	0	1

A ∩ B = {1, 3, 5, 31}

0	1	2	3	4	5	.....	12	13	14	15	16	17	18	.....	29	30	31
0	1	0	1	0	1	.....	0	0	0	0	0	0	0	.....	0	0	1

Figure 3-3: Intersection operation of the itemset A = {1, 3, 5, 12, 17, 30, 31} and B= {1, 3, 5, 13, 18, 29, 31}

- Membership operation: To find out whether an itemset is a member of another itemset takes at least O(log n) in the worst case using array representation. The proposed itemset representation uses a bitwise operation making the process of searching faster instead of recursively searching for such an itemset.

3.2.1 Dealing Numerical Attributes

In the dataset, each record or observation is expressed in terms of a feature or a characteristic. These features or characteristics are called attributes. This attribute ranges from one set of observations to another. These attributes can be numerical or categorical. A numerical attribute is either an integer or a float data type. This type of attribute contains a range of values.
The attribute can also be continuous. Since most of the data mining algorithms usually work on discrete values, the dataset needs to be converted to discrete values. The continuous attribute can be divided into intervals or ranges. Every interval is considered as a discrete value where the original value is mapped with the discrete value. Suppose the dataset consists of the following items as shown in Figure 3-4. The attribute is divided into intervals such as 1^{st} Attribute which consists of 1.2, 1.3, and 1.9 can be mapped to 2, and 2^{nd} Attribute which contains 23.1, 24.2, and 24.6 can be mapped to 25. The proposed itemset representation stores the dataset in the form of '1' or '0' which takes only one bit per item. Suppose an example of the itemset I_1={1, 2, 25 } is to be stored, a 32-bit integer is used. The items are not stored in the form of item values but in the form of the bit where '1' denotes the presence of the item in the itemset and '0' is if the item is absent from the itemset. An example is shown in Figure 3-5:

Id No	Attribute 1	Attribute 2
1	1.2	23.1
2	3.2	24.2
3	7.1	24.6
4	4.2	28.8
5	7.15	29.55
6	8.44	30.3
7	9.73	31.05
8	2.3	31.8
9	4.3	22.9
10	5.4	21.9
11	1.3	20.9
12	2.85	19.9
13	2.66	18.9
14	2.47	17.9
15	2.28	16.9
16	2.09	15.9
17	1.9	14.9

Figure 3-4: Dataset consisting of continuous values

A = {1, 2, 25}

0	1	2	3	4	5	.....	12	13	....	24	25	26	.....	30	31
0	1	1	0	0	0	.....	0	0	0	0	1	0	.....	0	0

Figure 3-5: Representation of the itemset I={1, 2, 25} using the proposed itemset representation

3.2.2 Dealing Categorical Attributes

Categorical attributes represent a value or a condition of a particular attribute in the instance data. Categorical attributes may consist of a set of discrete values which are more than two values [76]. The data mining field is overwhelmed with a huge amount of categorical attributes. To handle categorical attributes, they are mapped to numerical attributes and the dataset can be applied to the association rule mining algorithms. For example, an itemset that consists of I={'a', 'b', 'j', 'z'} is mapped to numerical values. It can be represented as shown in Figure 3-6.

A = {1, 2, 10, 26}

0	1	2	3	4	5	.....	9	10	....	24	25	26
0	1	1	0	0	0	.....	0	1	0	0	0	1

Figure 3-6: An example of a categorical dataset I={'a', 'b', 'j', 'z'}

3.3 Theoretical Analysis

The numerical and categorical attributes are stored in the memory with the help of a data structure. Depending on the type of data structure used, the amount of memory required for storing these attributes will also vary. This plays an important role because it affects the process of association rule mining. When the dataset is very large, all the itemsets cannot be kept in the main memory. It will require additional storage for storing the itemsets. Moreover, this process will just take away time since time will be spent more on I/O operation.

3.3.1 Effect of Attribute Values on the size of a single itemset

The attributes both categorical and numerical affect the size of the itemset. The dataset is stored in the memory based on the number of attributes. The size of the itemset will vary depending on the number of attributes present on the dataset. If the number of attributes is 6, then the size of this itemset is 6. Hence, if the number of attributes is large, the size of the itemset will also increase.
If the attribute is in numerical form, the attribute is stored based on the attribute value. The numerical attribute will be stored as *int* or $float$ value. Each *int* or $float$ takes 4 bytes for storage if an array representation is used. If the size of the itemset is 6, I ={1, 5, 8, 23, 31}, then the amount of memory needed will be 24 bytes(i.e., 4 *6). If the number of elements is n, the memory consumed will be 4*n bytes since each item takes 4 bytes.
If the attribute value is categorical in form, then the attribute value can be mapped to a numerical value that takes 4 bytes if array representation is used. Furthermore, if the attributes are stored in the form of characters then each character takes one byte. If the number of attributes is 6 then the memory consumed is 6 bytes(i.e., 1*6).

3.3.2 Existing Representations

The itemset representation is stored using a data structure. A data structure is a technique used for storing data efficiently in the computer. Different data structures can be used to represent an itemset on the computer. Most of the researchers have employed different types of data structures for representing the

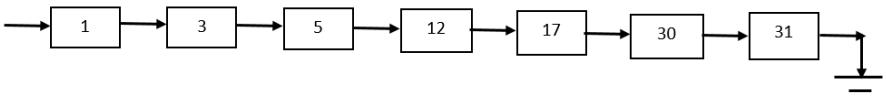

Figure 3-7: Linked list representation of an Itemset I ={1, 3, 5, 12, 17, 30, 31}

A = {1, 3, 5, 12, 17, 30, 31}

0	1	2	3	4	5	6	12	13	14	15	16	17	.....	30	31
1	3	5	12	17	30	31	.....	.....	....	.....	.....	....	.....	.....	....

Figure 3-8: Array representation of an Itemset I ={1, 3, 5, 12, 17, 30, 31}

itemsets. The most common data structures used for storing these itemsets are:

- Linked list: is one of the data structures used that is divided into two parts: the first part contains the element and the second part contains a pointer to the next element. To store an item using a linked list, 8 bytes will be required i.e., 4 bytes for the element and another 4 bytes for the pointer. Let's say the dataset size is 12, and the number of generated candidate itemsets would be $2^{12} \approx 4096$. To store the candidate itemsets, the memory consumption would be 8(4+4) * 4096 = 32768 bytes. The association rule mining algorithms such as FP-growth algorithm [36], Trie-based Apriori [9], Transaction Mapping Algorithm [70]. Each item in the itemset is accessed using a pointer causing a delay in the mining process. Another disadvantage of using a linked list is that it also takes up more memory since for each item an additional 4 bytes are needed for pointer[50]. An example is shown in Fig 3-7 where an itemset I ={1, 3, 5, 12, 17, 30, 31} is represented using a linked list. This scheme may be found useful to represent the set of lower cardinality but if the cardinality of the range is too big the space overhead increases linearly as the cardinality increases.

- Array: contains successive blocks where each block can store only one element. The element in the array is accessed using an index and each block is of the same size [51]. When the cardinality of the set is 12, the number of generated candidate itemsets would be $2^{12} \approx 4096$. The total memory consumption for generated itemsets is 4*4096 = 8192 bytes. H-mine is one such algorithm that uses array representation for itemset representation [64]. Another algorithm implemented by Gosta Grahne et.al also uses an array for the FP tree algorithm to enhance the performance[34].
  An itemset I ={1, 3, 5, 12, 17, 30, 31} can be represented using the array representation shown in Figure 3-8.

A = {1, 4, 5, 13, 16, 17, 31}

0	1	2	3	4	5	.....	12	13	14	15	16	17	.....	30	31
0	1	0	0	1	1	.....	0	1	0	0	1	1	.....	0	1

Figure 3-9: Bitmap representation of an Itemset I ={1, 4, 5, 13, 16, 17, 31}

- Bitmap: is a bit string of bits containing one bit per attribute value [22]. Each bit is represented as '1' if the element is present and '0' if the element is absent [7]. Suppose the number of generated items is 4096, then the amount of memory consumed will be 4096 * 1 bit = 512 bytes. Takeaki Uno et al.[75] have combined the three types of data structures such as prefix tree, array, and bitmap. A bitmap is marked by '1' using a matrix if the element is present. The prefix tree stores the sequences representing the trial starting from the leaf to the root. Another bitmap-based representation is introduced by Chen et al.,[11]. The database is scanned only once and the itemset is represented as a bit string. If the dataset contains the four items {1, 2, 3, 4}, the itemsets are {},{1}, {2}, {3}, {4}, {1, 2}, {2, 3}, {1, 3}, {1,4}, {2,4}, {3, 4},{1, 2, 3},{2, 3, 4}, {1, 2, 4}, {1, 3, 4}, {1, 2, 3, 4}. The itemsets are represented using 3 bitmap representation as {0000}, {0001}, {0010},{0100}, {1000}, {0011}, {0110}, {0101},{1001}, {1010} ,{1100}, {0111}, {1110}, {1011}, {1111}. The memory consumption is O($2n$). However, the representation can only handle 32-bit integer-size itemsets. Zaki et al., use the vertical bitmap representation of itemset where each item in the transaction is represented by the value '1' and '0', if otherwise. By applying the AND operation of the items, the itemset {i, j, k} is mapped to bit representation. However, this representation cannot be used when the size of the itemset is larger than 64 [81]. An example of the itemset I ={1, 4, 5, 13, 16, 17, 31} using bitmap representation is shown in Figure 3-9.

3.3.3 Proposed Representation

In the proposed itemset representation, the itemset can be represented using an integer array. The dataset is mapped to bit representation where an item is marked '1' if it is present and '0' if it is absent. The integer array size will be controlled by the highest cardinality set. Each item on the transaction is stored in the form of bit '1' or '0' depending on the presence and absence of the item. For a set size greater than 256 characters, then the integer bit array can be represented as 10 system-given integers. An example of the representation of the itemset is shown in Figure 3-10.

A={1,3,5,12,17,30,31}

0	1	2	3	4	5	.....	12	13	14	15	16	17	.....	30	31
0	1	0	1	0	1	.....	1	0	0	0	0	1	.....	1	1

Figure 3-10: Itemset I= { 1, 5, 10, 19, 23, 28, 31}represented using proposed itemset representation

After mapping the dataset to the proposed itemset representation, the set operations such as subset, intersection, superset, and membership are performed. Theoretically, it is evident from the previously mentioned representation that the array representation consumes less memory since only four bytes are used for each item as compared to the linked list representation that consumes more than 8 bytes(4 bytes for an element + 4 bytes for the pointer). Hence, the array representation is preferable to the linked list representation. For that reason, the comparison of the proposed itemset representation with the array representation of the set will be presented in the thesis. The time complexity of the different set operations using an array and proposed itemset representation is shown in Table 3.1.

Time Complexity	Array representation (if the array is ordered)	Proposed Itemset Representation
Subset	Max(m,n)	O(c)
Superset	Max(m,n)	O(c)
Membership	O(n)	O(c)
Union	O(m+n)	O(c)
Intersection	O(m+n)	O(c)
Set difference	O(m+n)	O(c)

Table 3.1: Time Complexity of Array and proposed itemset representation

As per Table 3.1, for example, the union operation in array representation takes the complexity of Max(m,n) where m,n is the number of elements in the first and second itemset respectively where each element consumes 4 bytes each. The operation in array representation is done based on the value of the attribute. The process of performing the union operation is done by iterating through both arrays starting from the index 0 to maximum(m, n) where the elements in both arrays are compared and then added to the set. The proposed itemset representation takes O(c) where c represents the number of memory words (32 bits, 64 bits) and each element consumes only 1 bit. The union operation using the proposed itemset representation is performing bitwise OR operation between the two arrays which is done at the bit level. Hence, operation using array representation may slow

down the operation as compared to the proposed itemset representation.

3.4 Evaluation of the Performance of Proposed Representation for Apriori Algorithm

Theoretically, the time complexity of the set operations using the proposed itemset representation is less than the array representation as shown in Table 3.1. These set operations are used in the process of mining the itemsets by different algorithms. One such algorithm is the Apriori algorithm.

3.4.1 Apriori Algorithm

The Apriori Algorithm has become the foundation of the association rule mining algorithms. It was introduced by Agarwal. Many researchers have attempted to enhance the performance of the algorithms. The Apriori Algorithm works on the principle that if the "itemset is large then its subset is also large."
The Apriori Algorithm is divided into two parts:-

- To generate the frequent itemsets from the dataset.
- To generate the rules from the generated frequent itemsets.

In the beginning, the dataset is scanned to discover the frequency of the itemset. The itemsets are called Large itemsets L. These are generated starting with the itemset size 1. At level 1, the large itemset is L_1. The large itemset with the same cardinality is joined to generate the next itemsets. These itemsets are now called the candidate itemsets C_2. At the $(k-1)_{th}$ level, the L_{k-1} itemsets are joined to generate the L_k. The union operation of the itemsets that belong to L_{k-1} is considered only when the itemsets share the common first k-2 itemsets. The support count of all the itemsets in C_k is calculated by scanning the dataset. The itemset that has a support count equal to or greater than the minimum threshold support count is called a frequent itemsets. The itemsets available in C_k that have a support count more or equal to minimum support are designated as members of L_k and the remaining are dropped. This process of generation of candidates and finding the frequent itemsets continues till C_k becomes empty.
These frequent itemsets are then used for the generation of rules. The rules are

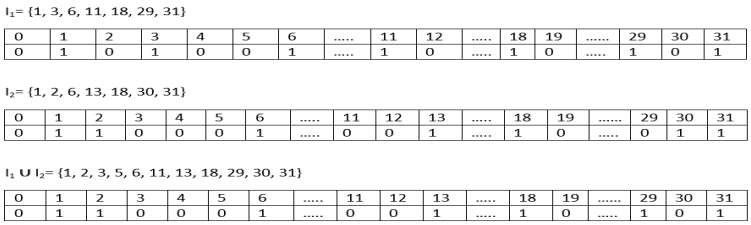

Figure 3-11: Union operation of itemset I_1= {1, 3, 6, 11, 18, 29, 31} and I_2 ={1, 2, 6, 13, 18, 30, 31} generates $I_1 \cup I_2$ ={1, 2, 3, 5, 6, 11, 13, 18, 29, 30, 31}

generated depending on confidence. The frequent itemsets whose confidence count is less than the threshold will not be considered.

3.4.2 Incorporating the Itemset Representation for the Apriori and its variants

The array and proposed itemset representation are incorporated in the Apriori Algorithm.

3.4.2.1 Itemset Generation

According to the Apriori Algorithm, the itemset with the same cardinality is self-joined to generate the next level itemset. Using the proposed itemset representation, the union operation is performed between the two itemsets to generate the next itemset. The union operation is performed using bitwise operators that take O(c) time. For example, union operation between the two itemsets I_1= {1, 3, 6, 11, 18, 29, 31} and I_2 ={1, 2, 6, 13, 18, 30, 31} generates $I_1 \cup I_2$ ={1, 2, 3, 5, 6, 11, 13, 18, 29, 30, 31} as shown in Figure 3-11.

3.4.2.1.1 Performance Analysis of Existing Techniques

The array representation is incorporated for the Apriori Algorithm. The process of generating itemsets is implemented using array representation including the set operations. The set operations such as union, subset, superset, intersection, and membership are tested in the array representation. The operations are done based on the attribute value. Since the operation is done on the value of the item, the set operations take time due to the process of searching, finding duplicates, and merging each item. Therefore, representing the itemsets using the

array representation will slow down the process of generating the frequent itemsets. Furthermore, the size of each item is 4 bytes. If the dataset consists of an itemset whose size is 4096, then the memory consumed is 4 *4096 = 16384 bytes. Now, if the generated itemsets are huge and a single computer is used then these itemsets may not fit in it. An additional memory requirement is needed for such cases. The I/O operations may further delay the process of mining the itemsets.

A 4GB RAM 64-bit computer is used for experimenting using the synthetic dataset. The three datasets used are as follows:

- First dataset: contains 50 attributes with 1000 transactions. The experiment is run with support count= 1%, 2.5%, and 5%.
- Second dataset: has 50 attributes and a size of 2000 records.
- Third dataset: consists of attributes of 50 with a size 3000 number of transactions.

The experiments are performed on the three datasets. The itemsets are generated using the Apriori Algorithm. The time and memory consumption are recorded with different support count=1%, 2.5%, and 5%.

3.4.2.1.2 Performance Analysis of Proposed Techniques

The proposed itemset representation is integrated into the Apriori Algorithm. The set operations are computed using the bitwise operations that work on the array at the individual bits. Typically, bitwise operations are considerably quicker than division, multiplication, and addition. Using the proposed itemset representation, the process of generating the frequent itemsets will be reduced. Each item in the proposed itemset representation takes only one bit. If the size of the highest cardinality of the itemset is 4096, then the memory consumption is only 512 bytes. The proposed itemset representation is tested on the three datasets based on the different support counts. The time and memory consumption with different-sized support counts and datasets are also measured.

3.4.2.2 Proof of Correctness

The experiments are performed using the existing representation i.e., array representation and the proposed itemset representation for the process of mining the

Sl.No	Support count	Dataset Size	Itemset Representation				Representation Proposed Itemsets Representation			
			Number of generated itemsets							
			2^{nd}	3^{rd}	4^{th}	5^{th}	2^{nd}	3^{rd}	4^{th}	5^{th}
1	1	1000	48	22	7	1	48	22	7	1
2	2.5	1000	38	14	3	-	38	14	3	-
3	5	1000	6	-	-	-	6	-	-	-
4	10	1000	-	-	-	-	-	-	-	-
5	1	5000	48	22	7	1	48	22	7	1
6	2.5	5000	41	21	6	1	41	21	6	1
7	5	5000	1	-	-	-	1	-	-	-
8	10	5000	-	-	-	-	-	-	-	-
9	1	20000	41	21	6	1	41	21	6	1
10	2.5	20000	41	21	6	1	41	21	6	1
11	5	20000	41	21	6	1	41	21	6	1
12	10	20000	1	-	-	-	1	-	-	-

Table 3.2: Number of generated itemsets using Array and Proposed itemset representation

itemsets. The number of generated itemsets produced by the array and the proposed itemset representation are also measured. Depending on the size of the datasets and the support count, the number of generated itemsets is shown in Table 3.2 above: As seen from Table 3.2 above, at each level the number of generated itemsets is the same for the Apriori algorithm using both the array and the proposed itemset representation. Hence, the Apriori algorithm using the proposed itemsets representation is correct.

3.4.2.3 Proof of Completeness

The generated itemsets that are represented using the proposed itemset representation and the array representation are measured at each level. The maximum length of the itemset extracted from the different datasets is shown in Table 3.3. From Table 3.3, it is observed that the maximum length of the generated itemsets represented using the array representation is the same as those represented using the proposed itemset representation. Therefore, the Apriori Algorithm represented using the proposed itemset representation is complete. A compact bitmap representation of a set supported by the corresponding implementation of the different set-theoretic operations are used in the different stages of rule mining. The Apriori algorithm was supported by the proposed itemset representation. The experimen-

Sl.No	Dataset Size	Maximum length of generated frequent itemsets	
		Array Representation	Proposed Representation Itemsets
1	1000	5	5
2	5000	5	5
3	20000	5	5

Table 3.3: Number of itemsets generated with maximum length using Array and Proposed itemset representation from different Dataset

Operations	Complexity	Proposed Itemset Representation			Array Representation		
		1st dataset	2nd dataset	3rd dataset	1st dataset	2nd dataset	3rd dataset
Subset	Time (ms)	61755.54	77927	320510.5	101495	464186	9088453
	Memory(kb)	30389	85893	151565	44369	103138	199043
Superset	Time (ms)	64031.64	77618.75	442944.5	108996.66	418338	9868559
	Memory(kb)	28455.4	63163.75	222729.5	45737	110555	199625
Set Difference	Time (ms)	66515.28	80000.8	419711.15	123024.6	533906	9914321
	Memory(kb)	24133	77457	308018	35975	109112	2200279
Union	Time (ms)	64735.24	84399.66	309622.3	128267.5	592152	12345539
	Memory(kb)	28845	53516.33	334095.6	55742	172399	5464948
Intersection	Time (ms)	64149.32	89061.3	393331.3	132009.39	599124	12139968
	Memory(kb)	28168.8	52190.46	375994.5	45920	107952.5	3163159
Membership	Time (ms)	4715	11215	28404	10145	31645	90212
	Memory(kb)	29564.2	71635	152940.8	49895	103192.5	209348.7

Figure 3-12: The time(millisecond) and memory(kilobits) consumption using the proposed itemset representation and the array representation

tal results are shown in figure 3-12. The proposed representation outperforms the array representation of itemsets significantly when used in the Apriori algorithm concerning time and memory consumption.

3.4.3 Incorporating the Searching Techniques for Itemsets in the Apriori Algorithm

The itemsets are generated and maintained for each level with multiple lists based on the support count. The itemsets whose support count is less than the minimum threshold are dropped, whereas the ones that are greater than the minimum threshold will be added to the next level. Before adding the itemsets to the list, redundant itemsets need to be removed. To remove redundancy, the itemset is

searched in the list whether the itemset is already present on the list or not. For this purpose, a searching technique is used for searching the itemsets. The commonly used search techniques are linear and binary search.

3.4.3.1 Performance Analysis of Linear Search Technique used in Apriori Algorithm

The linear search technique searches for an item consecutively one after another item. It starts its operation by searching for the item from the first position on the list and continues the searching till it finds the searched item or it reaches the last item on the list.

The linear search is incorporated into the Apriori algorithm. The synthetic datasets are used for testing the improvised Apriori Algorithm. These synthetic datasets consist of three different sizes.

- First dataset: The dataset consists of 1000 transactions.
- Second dataset: consists of 5000 transactions.
- Third dataset: consists of 20000 transactions.

The memory and time consumption for the itemsets represented using an array and the proposed itemset represented by the improvised Apriori Algorithm is shown in Figure 3-13, 3-15, 3-16, 3-18.

As seen from the results, it is evident that the linear search incorporated in the Apriori Algorithm using the proposed itemset representation exploits that of the array representation.

3.4.3.2 Performance Analysis of Binary Search Techniques used in Apriori Algorithm

The binary search technique is used for locating a value in a sorted list[60]. Using this searching technique in Multi-View Stereo methods on competitive benchmarks shows that this method consumes much less memory[55]. The binary search technique is used only for a sorted list. It divides the list into two parts:- lower, middle, and upper. The searched element is compared with the middle element and if it is a match, then the element is found. If the searched element is greater than the middle element, then searching is done on the upper part of the list. If the

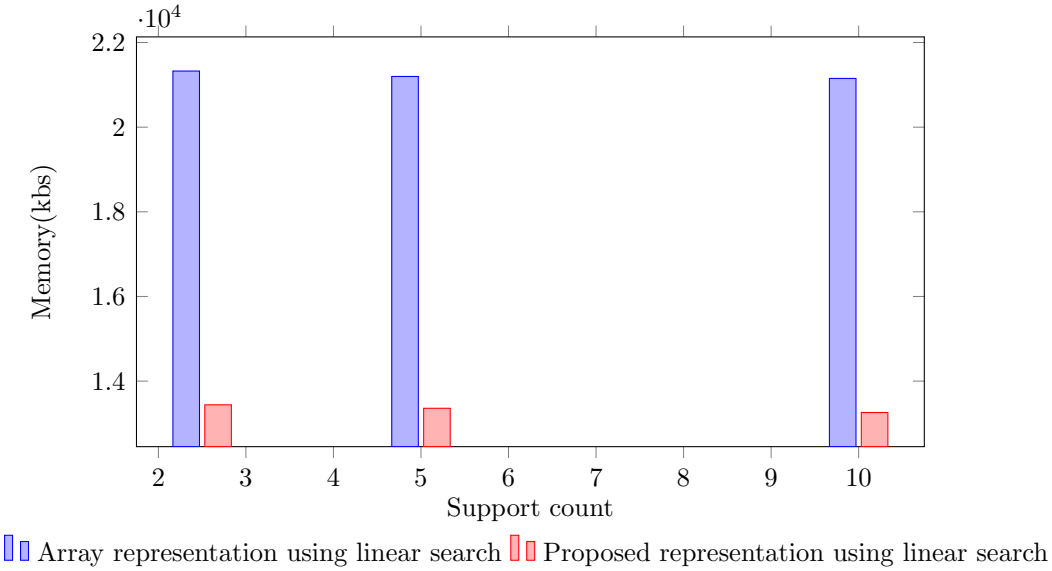

Figure 3-13: Memory consumption of candidate itemset generation represented by array and proposed itemset representation using linear search for the first dataset with support count =2.5%, 5% and 10%

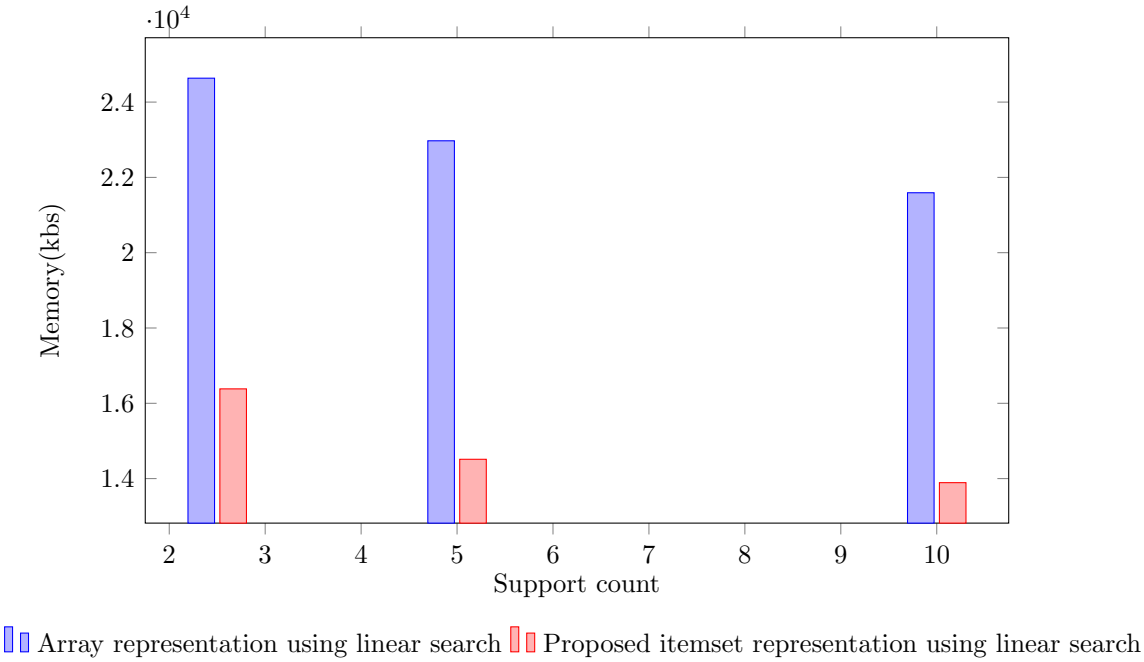

Figure 3-14: Memory consumption of candidate itemset generation represented by array and proposed itemset representation using linear search for the second dataset with support count =2.5%, 5% and 10%

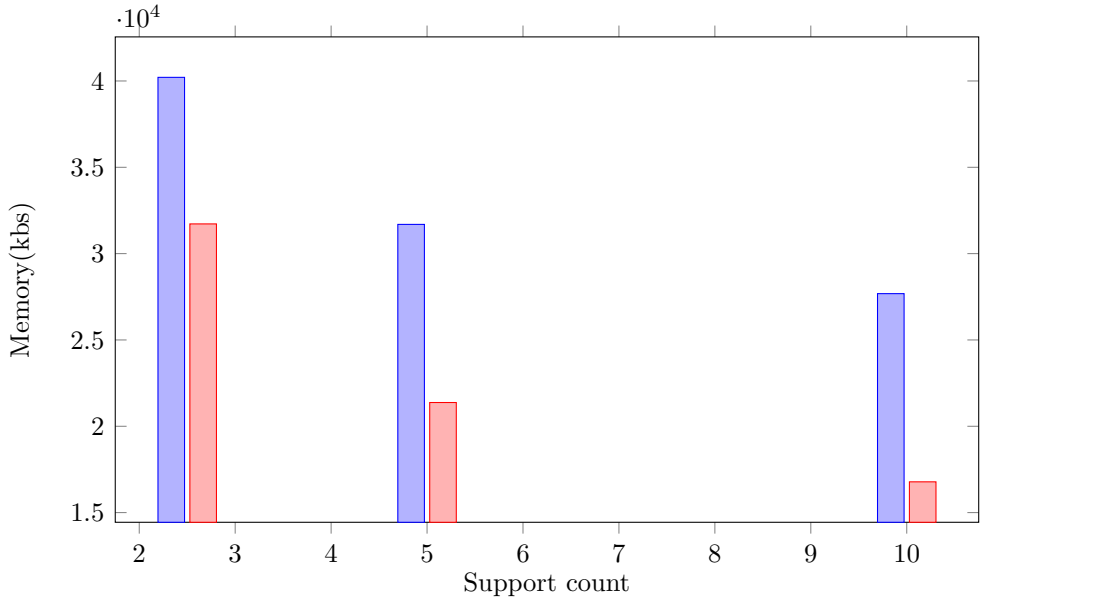

Figure 3-15: Memory consumption of candidate itemset generation represented by array and proposed itemset representation using linear search for the third dataset with support count =2.5%, 5% and 10%

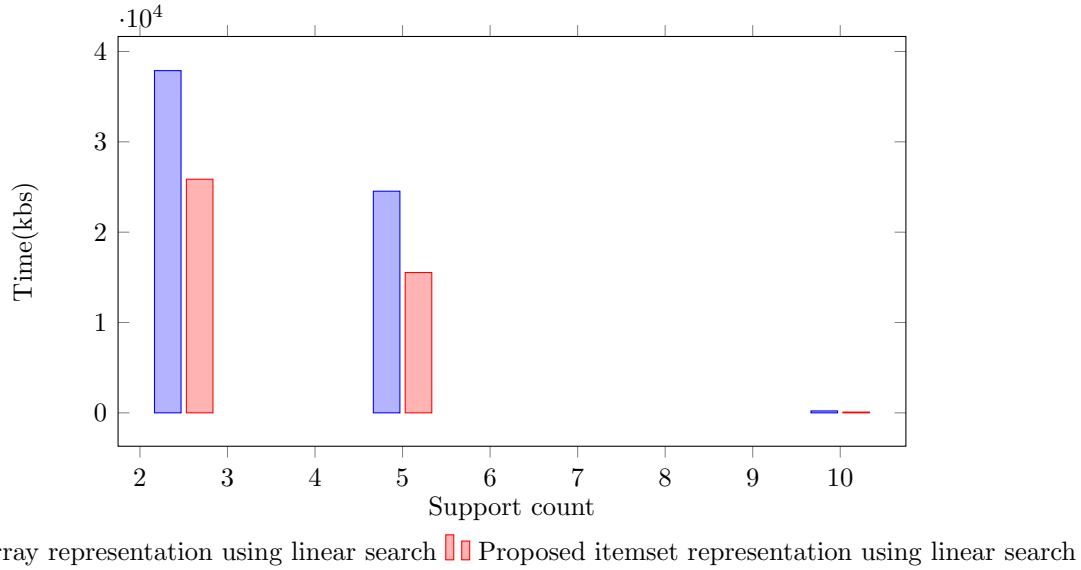

Figure 3-16: Time consumption of candidate itemset generation represented by array and proposed itemset representation using linear search for the first dataset with support count =2.5%, 5% and 10%

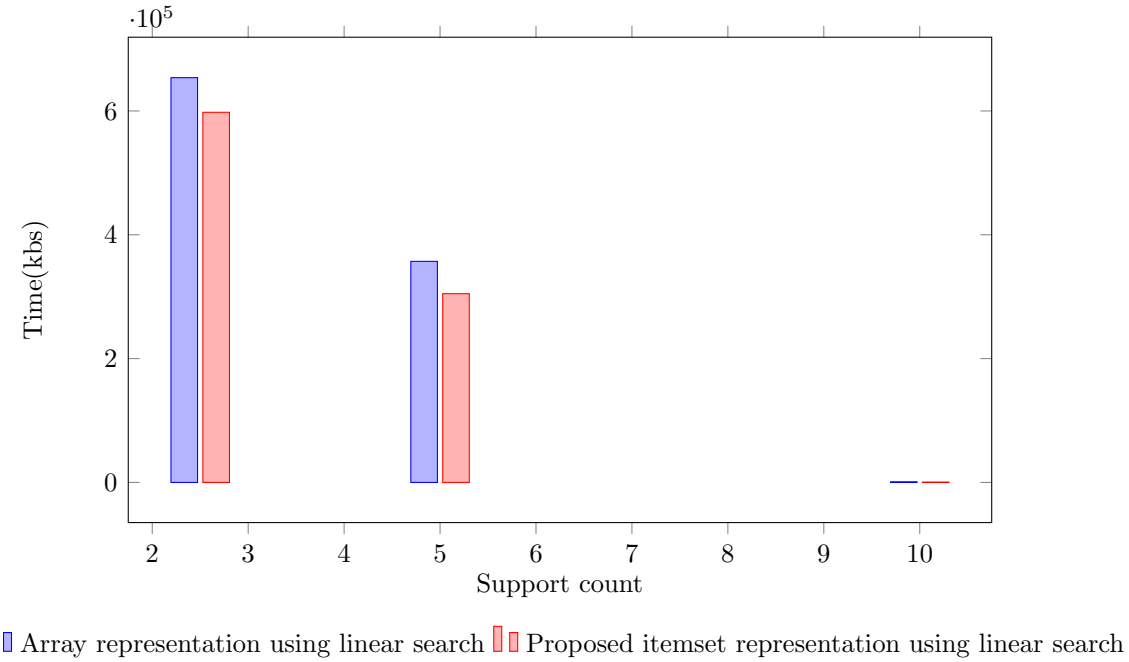

Figure 3-17: Time consumption of candidate itemset generation represented by array and proposed itemset representation using linear search for the second dataset with support count =2.5%, 5% and 10%

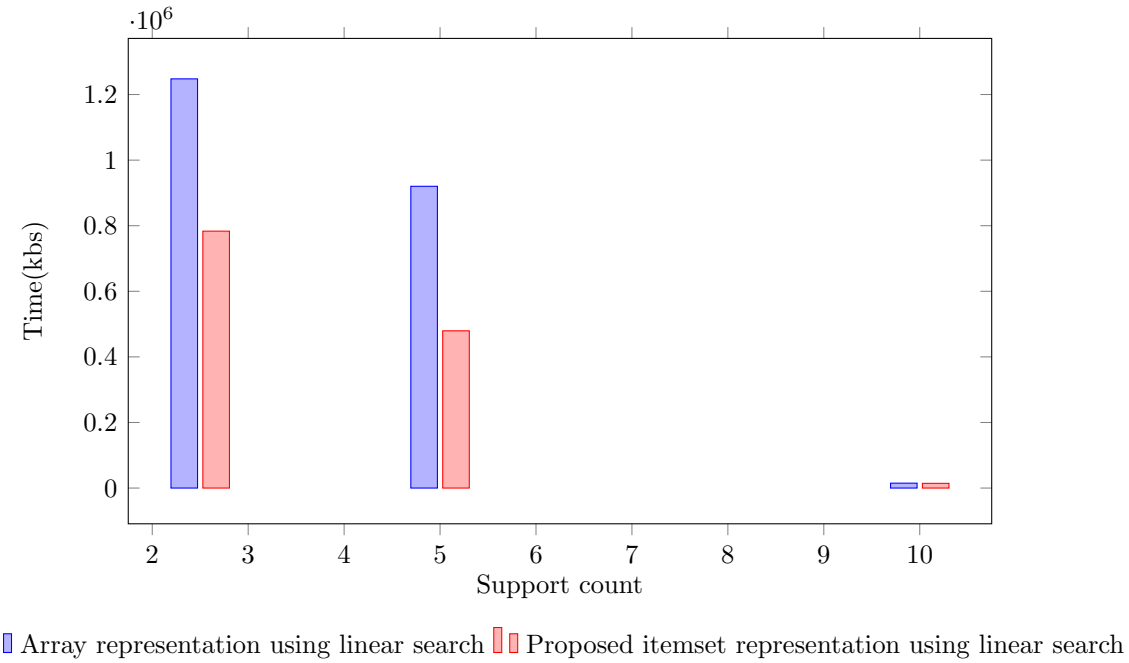

Figure 3-18: Time consumption of candidate itemset generation represented by array and proposed itemset representation using linear search for the third dataset with support count =2.5%, 5% and 10%

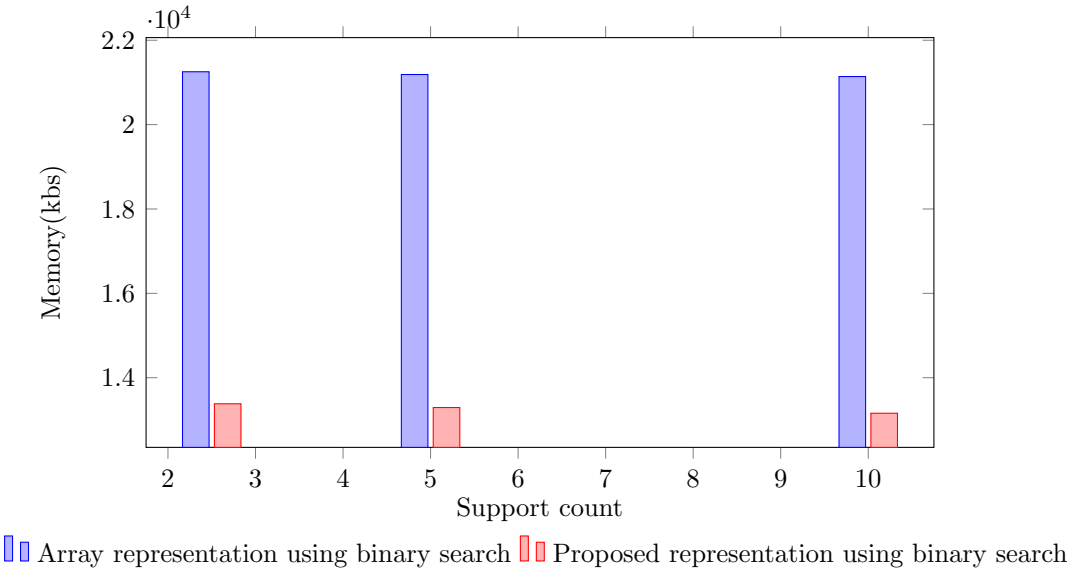

Figure 3-19: Memory consumption of candidate itemset generation represented by array and proposed itemset representation using binary search for the first dataset with support count =2.5%, 5% and 10%

searched element is lesser than the middle element, then searching is done on the lower part of the list.

This process continues until the list is empty. The memory and time consumption of the itemsets are shown in Figure 3-19, 3-20, 3-21, 3-22, 3-23, 3-24 for the Apriori Algorithm that incorporates the binary search technique. They show the performance of this improved Apriori Algorithm which is supported by the array representation and the proposed itemset representation.

3.4.3.3 Conclusion

The proposed bitmap representation is implemented in the set and the operations were used in the different phases of rule mining. The Apriori Algorithm avoids the duplicate itemset generation for level k+1 by checking first k-1 items are common. For example, an itemset ABC and ABD will generate ABCD but ABC and BCD will not generate ABCD. We could not design an efficient method for checking "first k-1 common items" in the proposed bitmap representation. Hence, we have eliminated the duplicate items from the candidate set based on an efficient searching technique on a sorted list of itemset. By following the candidate generation procedure suggested by the Apriori algorithm, it can be ensured that the newly generated candidate will be bigger than the currently available candidates. This, in turn, ensures that the list of candidates generated till now will always be in

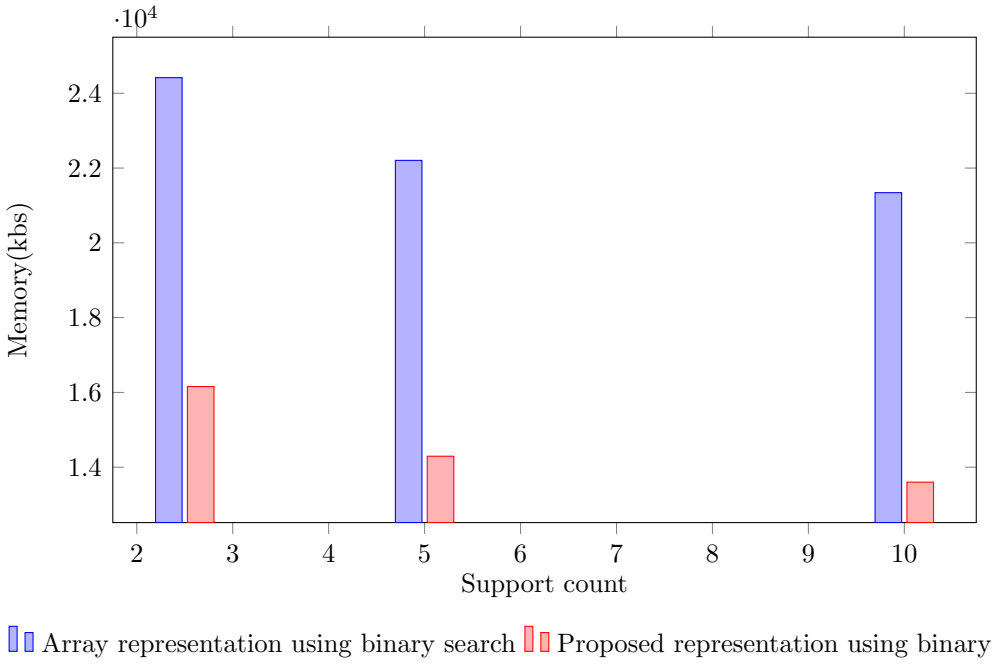

Figure 3-20: Memory consumption of candidate itemset generation represented by array and proposed itemset representation using binary search for the second dataset with support count =2.5%, 5% and 10%

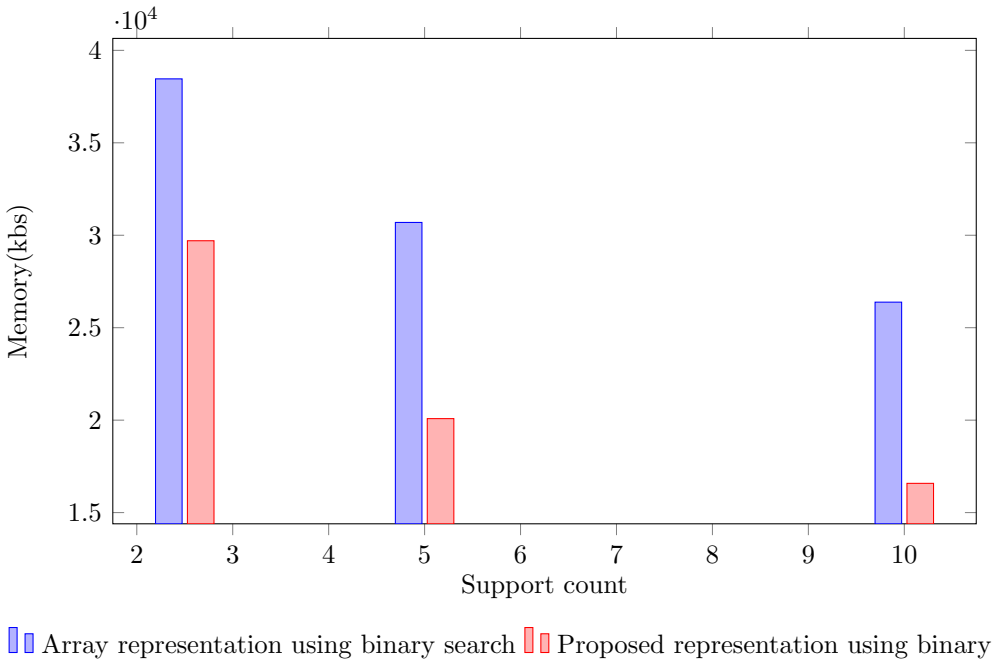

Figure 3-21: Memory consumption of candidate itemset generation represented by array and proposed itemset representation using binary search for the third dataset with support count =2.5%, 5% and 10%

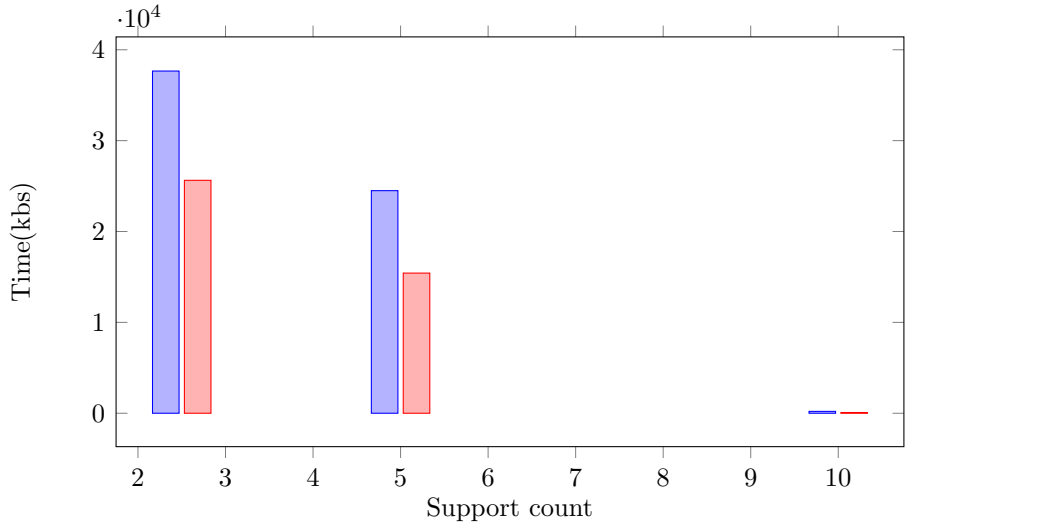

Figure 3-22: Time consumption of candidate itemset generation represented by array and proposed itemset representation using binary search for the first dataset with support count =2.5%, 5% and 10%

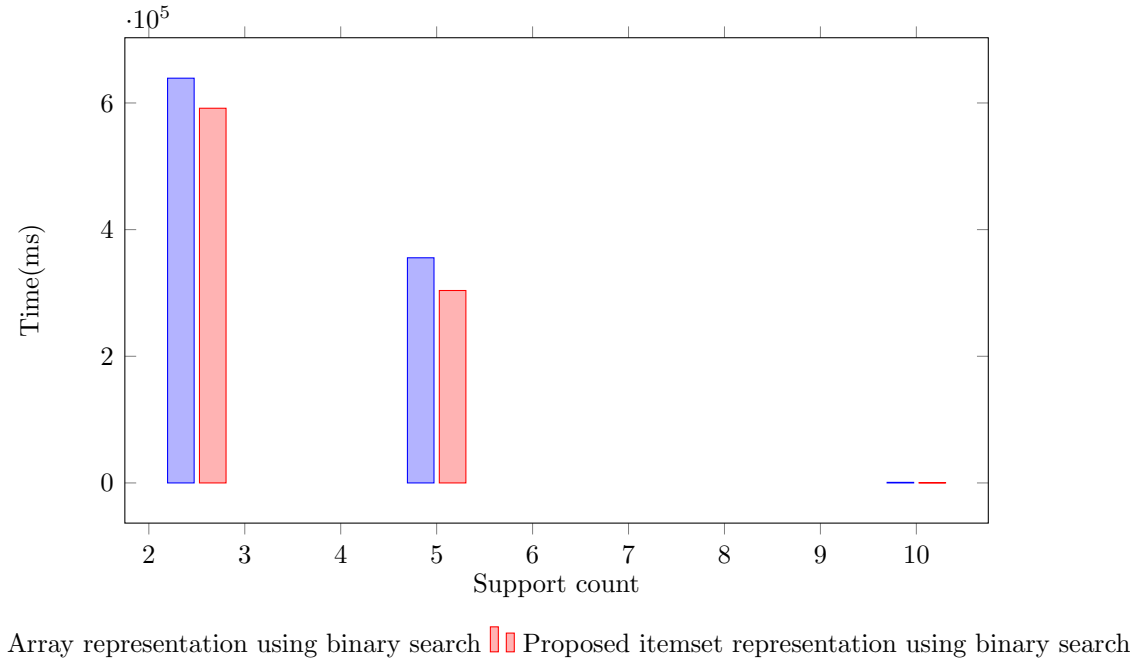

Figure 3-23: Time consumption of candidate itemset generation represented by array and proposed itemset representation using binary search for the second dataset with support count =2.5%, 5% and 10%

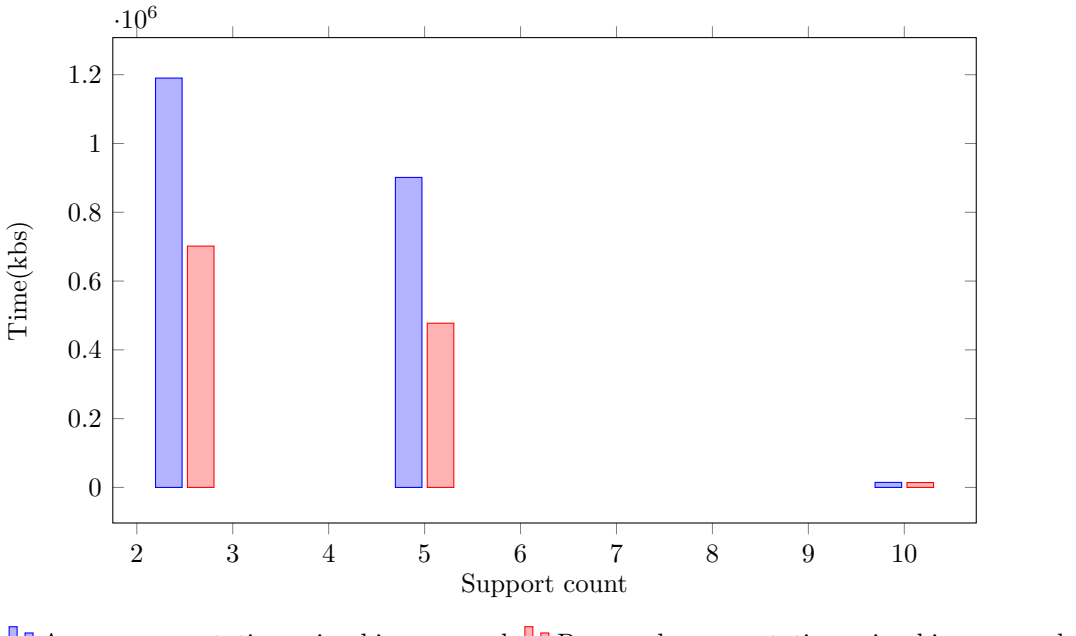

Figure 3-24: Time consumption of candidate itemset generation represented by array and proposed itemset representation using binary search for the third dataset with support count =2.5%, 5% and 10%

order without the need for any separate way of sorting procedure. Furthermore, the binary search technique was integrated into the Apriori Algorithm. All the experimental results were produced pertinently. The outcomes show that the proposed representation surpasses the array representation of itemsets when used in the Apriori algorithm.