Chapter 5

Generate Rules from Mined Frequent Itemsets

5.1 Introduction

Association rule mining aims to find the frequent itemsets to analyze and predict. In many domains, there are numerous approaches used for improving the process of decision-making. One such application is the detection system. The intrusion data is analyzed using the association rules. It also constructs the inspection rule base. This enables the system to independently learn by itself. Thus, it provides scalability of the system [83]. The healthcare system collects its records through Electronic health records (EHRs) used by different organizations. With the help of association rule mining algorithms, the system enhances patient care and the efficiency of the delivery of healthcare. It facilitates the health care routine and gives accurate results for identifying the disease [25]. With a dynamic environment, there is a need to address the issue of decision making. An approach was introduced to improve the decision-making process. The feature of the approach is that it is a hybrid of machine learning algorithms [43]. For rule generation, the metric confidence is applied to the itemset. Confidence measures the strength of the rule and how many times X occurs in the transaction that Y also exists. $Y \rightarrow X = P(X|Y) =$ Number of transactions that contain both X and Y/ number of transactions that contain Y.
Thus, it is observed that rule mining algorithms have proved beneficial for different applications and various domains. Some of the algorithms used for generating the rules from the already mined frequent itemsets are discussed in the next section.

5.1.1 Agarwal's Algorithm

It was the algorithm introduced by Agarwal for generating the rules from the frequent itemsets. In this algorithm, the rule is generated with a consequent part containing only one item or an element. A rule of the form x $\rightarrow$ y is generated from a frequent itemset I =$\{I_1, I_2, I_3, ..., I_n\}$ such that $n \geq 2$. In the rule, the antecedent part is a subset k of I where k has n-1 items and the consequent part has I- k items. The disadvantage of the algorithm is that it is ineffective and inability to generate all the possible rules. Although there are (2^n -2) possibilities, the algorithm can test only n rules [1].

5.1.2 Srikant's 1^{st} Algorithm

This algorithm is also known as Srikant's Simple Algorithm. It is more of a generalized structure from Agarwal's algorithm. The consequent part of the rule is not constrained to one item. The frequent itemsets whose subsets are not empty are discovered first. Then, for every subset x, a rule is produced based on the threshold value. The algorithm generates all the possible rules but it wastes a lot of time due to redundant checking. Suppose an itemset ABCD, its subsets are ABC, AB, and A. Based on the confidence metric, the possible rules are as follows:

- ABC $\rightarrow$ D
- AB $\rightarrow$ C D
- A $\rightarrow$ B C D

If the confidence value of the rule ABC $\rightarrow$ D is not above the threshold value then the rule A B $\rightarrow$ C D cannot be greater than the threshold value. Also, the support count of the itemset AB cannot be more than ABC. The confidence of rule first rule AB cannot be more than the second rule ABC. Srikant's first algorithm checks for the second rule which wastes time and likewise, the rule A $\rightarrow$ BCD will be checked [3].

5.1.3 Srikant's 2^{nd} Algorithm

Srikant's 2^{nd} algorithm is an enhancement of the previous algorithm. The reason is that it removes the redundancy by avoiding redundant checking of the rule. If a has a subset c, then the support of the itemset c cannot be more than the support count of the itemset a. Similarly, the confidence of the rule $c \rightarrow l$- c cannot be more than $a \rightarrow l$- a. This approach is known as the downward closure property which states that "if an itemset is frequent then its subset is also frequent".
The algorithm produces the same consequent numerous times for different antecedents. Suppose if W $\subset$ Y, all the consequents are generated while generating the rules for W. However, the operation will be repeated for Y although these are already generated before for the Y since Y is a superset of W. This process is a setback reducing the algorithm efficiency and effectiveness [3].

5.1.4 NBG's Algorithm

This algorithm can tackle the issue faced in Srikant's 2^{nd} algorithm. The feature of the algorithm is that it can produce all the possible rules fulfilling the minimum confidence criteria. The algorithm discovers all the rules with different consequents and fixed antecedents. It proceeds to the next level with the same antecedent if the present level has at least two itemsets that satisfy the minimum confidence. It starts generating the rules starting with the fixed and single-item antecedent. After all the rules with fixed antecedents are generated then it will proceed to the next antecedent. Likewise, it also checks the consequent part with one item and proceeds to the next level for the same antecedent. If at level k, the rules have confidence less than the threshold, then no rules will be generated at level $k + 1$ [32].

5.2 Rule Generation with Existing Representation

The array itemset representation is used for representing the itemsets which are then mined using the Apriori algorithm. The rules are generated from the frequent itemsets using the algorithms mentioned in the above section. The following datasets are used for experimenting:

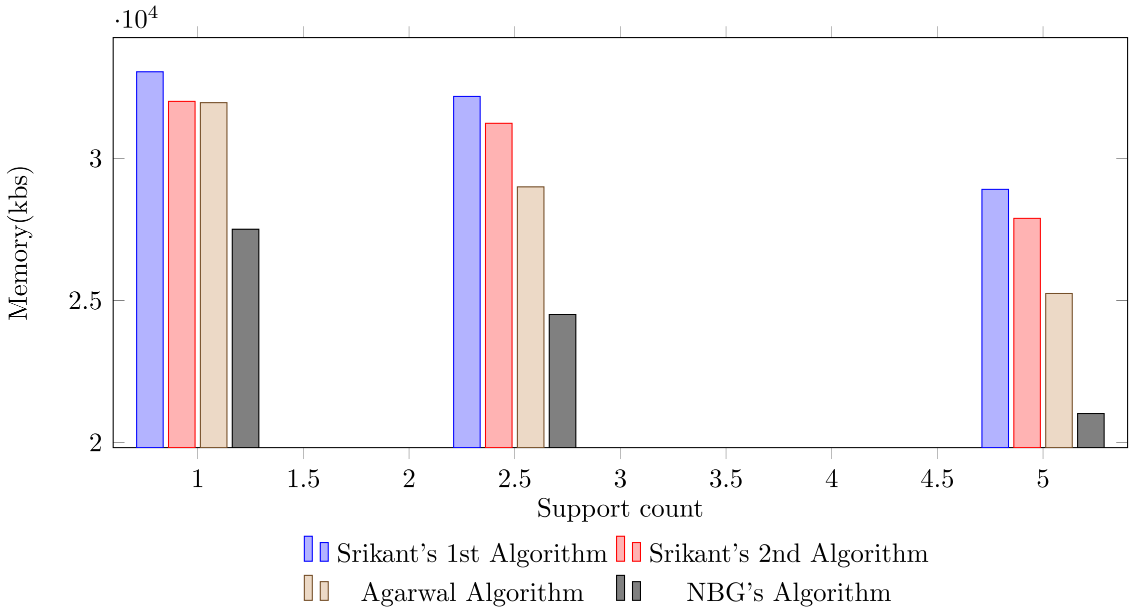

Figure 5-1: Memory(kbs) consumption of rule generation algorithms using proposed itemset representation for Hunington's dataset with confidence=1% and support count= 1%, 2.5%, 5%

- Huntington's dataset: The dataset [1] consists of the patient's details including whether the patient has repeated CAG chain, patient's age, patient's gender. The time and memory consumption for the generation of the rules using Apriori algorithms for Huntington's dataset are shown in Figure 5-1, 5-2, 5-3, 5-4, 5-5, 5-6, 5-7, 5-8, 5-9, 5-10, 5-11, 5-12.

- Synthetic dataset: There are three types of datasets depending on the size. The datasets are as follows:

  - 1st dataset: The dataset consists of 1000 transactions with 50 attributes.
  - 2nd dataset: consists of 5000 transactions including 50 attributes.
  - 3rd dataset: contains 20,000 transactions with 50 attributes.

  The experiments are performed on different criteria shown in Figure 5-13, 5-14, 5-15, 5-16, 5-17, 5-18, 5-19, 5-20, 5-21, 5-22, 5-23, 5-24.

- Chess dataset: The dataset [2] consists of tables with game-theoretic values for the legal position. The game-theoretic values stored designate which are the winning positions for both sides consisting of 28056 instances.

[1]http://biogps.org/#goto=genereport&id=1017&show_dataset=E-GEOD-8762
[2]httpshttps://archive.ics.uci.edu/dataset/21/chess+king+rook+vs+king+knight

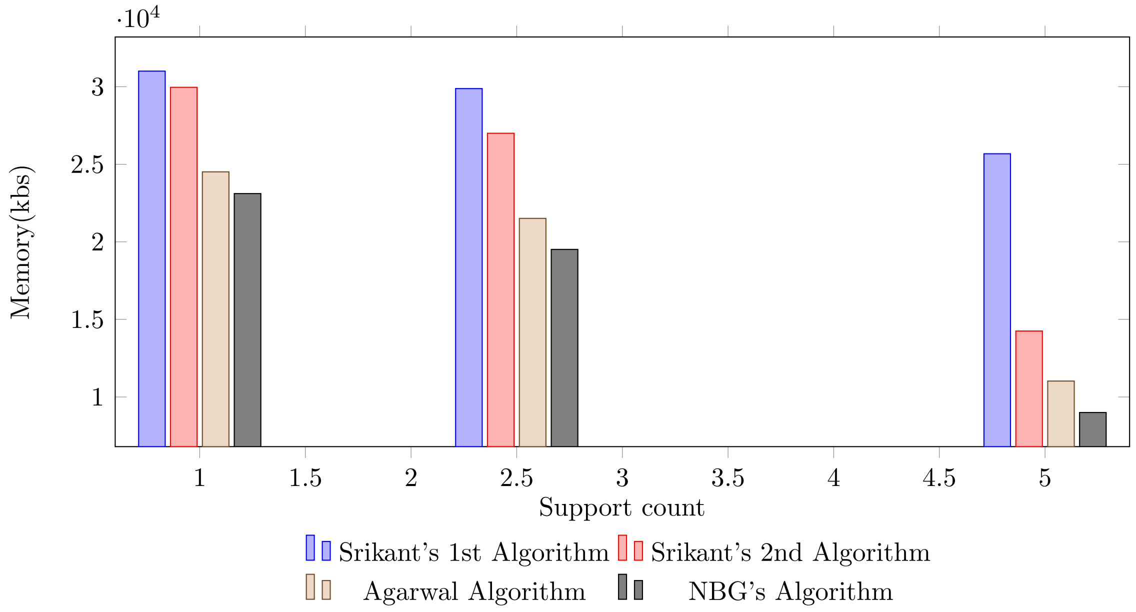

Figure 5-2: Memory(kilobits) consumption of rule generation algorithms using proposed itemset representation for Hunington's dataset with confidence 2.5% and support count 1%, 2.5 %, 5 %

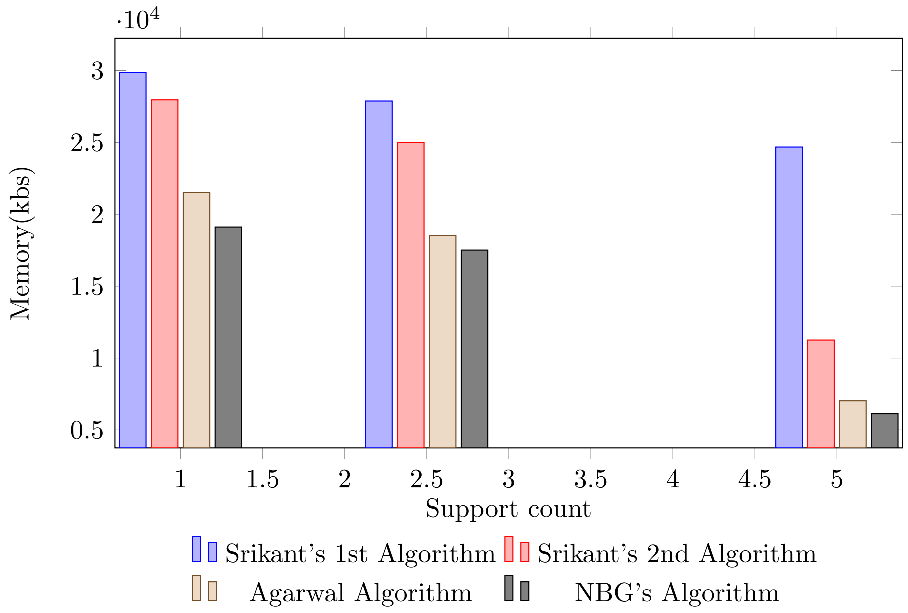

Figure 5-3: Memory(kilobits)consumption of rule generation algorithms using proposed itemset representation for Hunington's dataset with confidence 5% and support count =1%, 2.5 %, 5 %

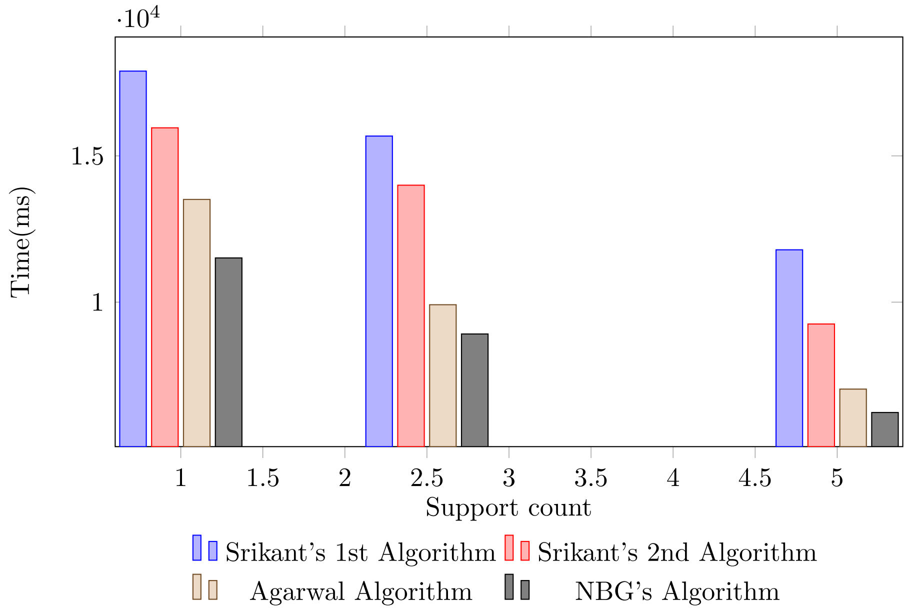

Figure 5-4: Time(ms) consumption of rule generation algorithms using proposed itemset representation for Hunington's dataset with confidence 1% and support count 1%, 2.5% and 5%

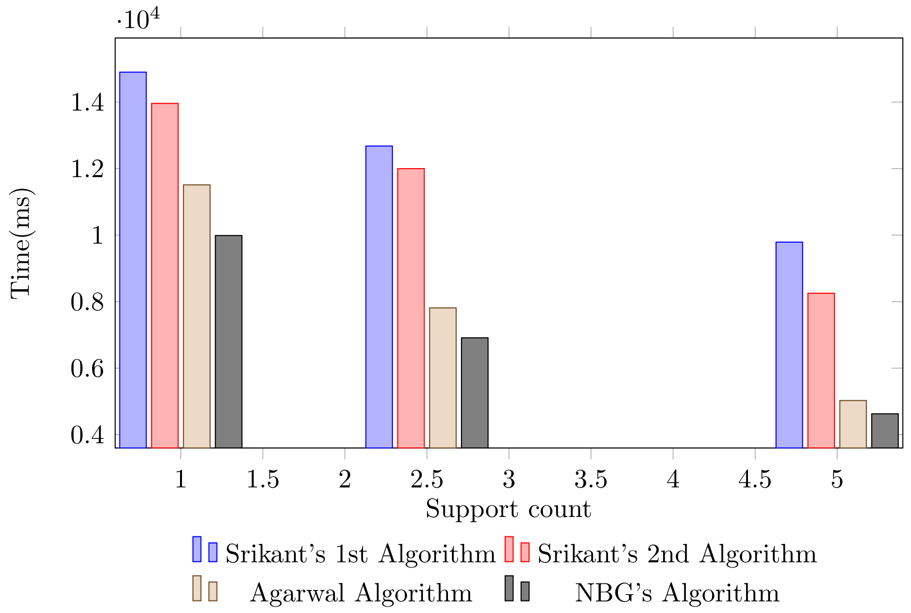

Figure 5-5: Time(ms) consumption of rule generation algorithms using proposed itemset representation for Hunington's dataset with confidence 2.5% and support count 1%, 2.5 % and 5%

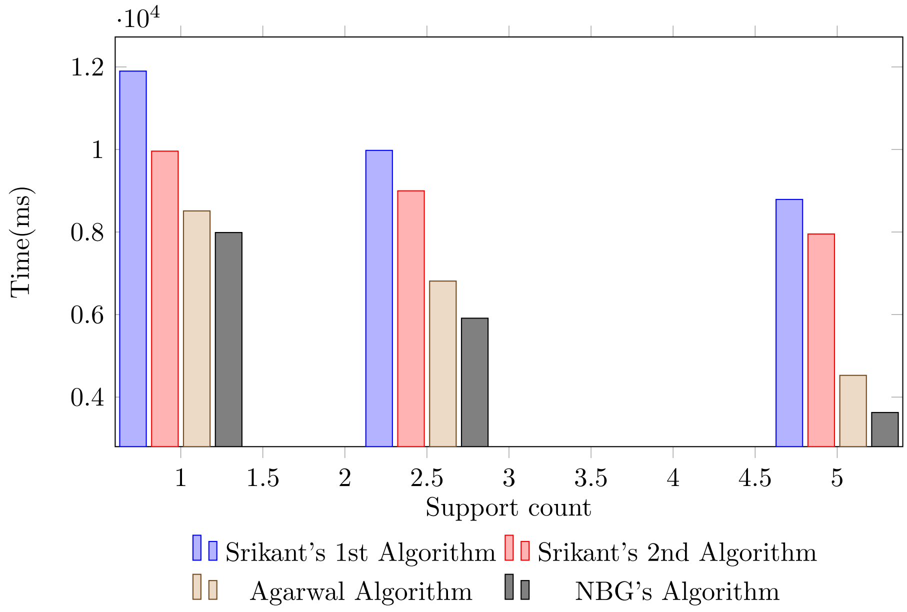

Figure 5-6: Time(ms) consumption of rule generation algorithms using proposed itemset representation for Hunington's dataset with confidence 5% and support count 1%, 2.5% and 5%

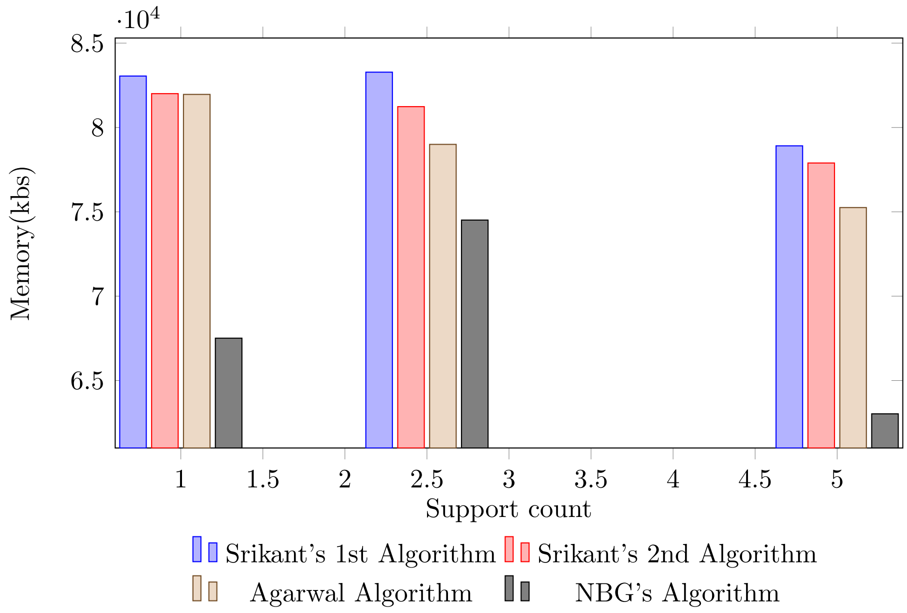

Figure 5-7: Memory(kilobits) consumption of rule generation algorithms using array itemset representation for Hunington's dataset with confidence 1% and support count 1%, 2.5% and 5%

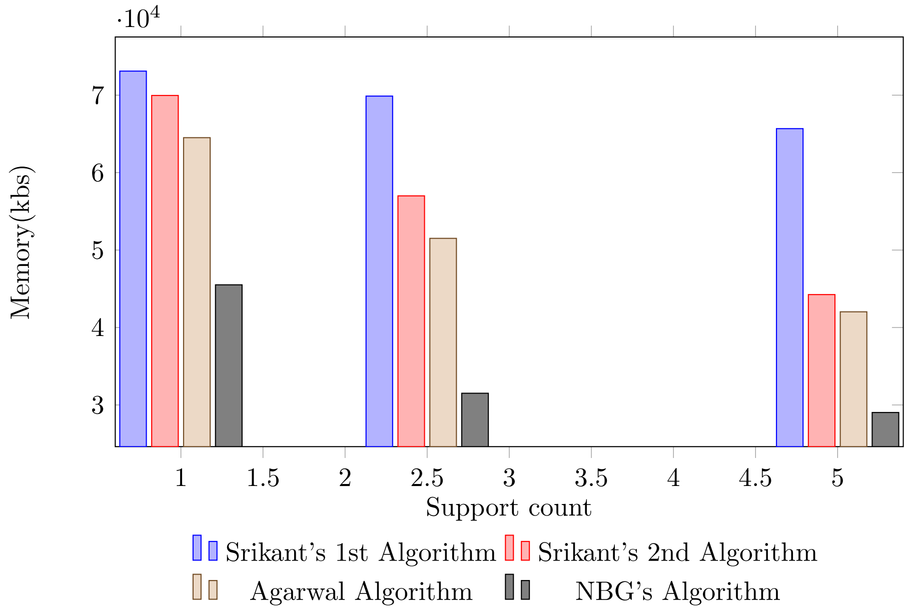

Figure 5-8: Memory(kilobits) consumption of rule generation algorithms using array itemset representation for Hunington's dataset with confidence of 2.5% and support count 1%, 2.5% and 5%

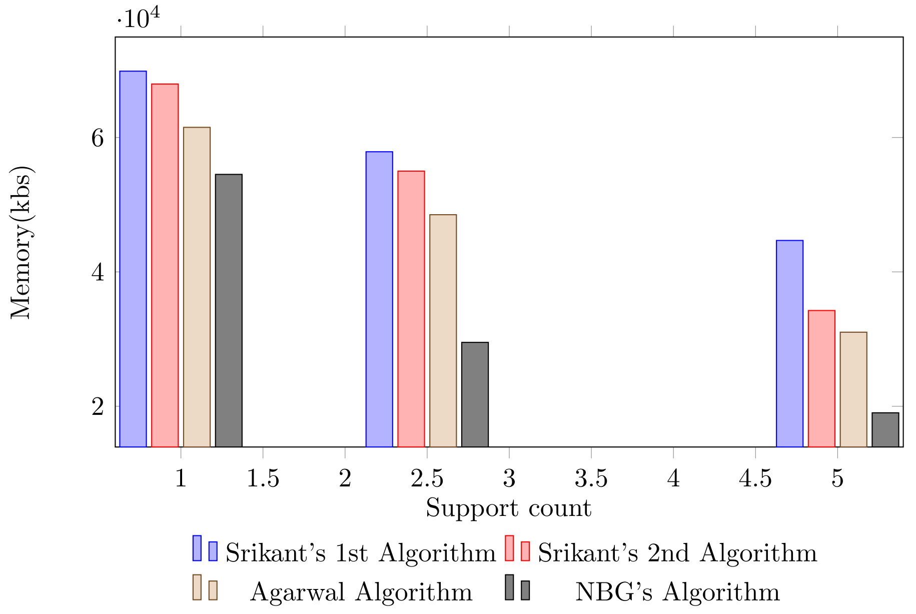

Figure 5-9: Memory(kilobits) consumption of rule generation algorithms using array itemset representation for Hunington's dataset with confidence 5% and support count 1%, 2.5% and 5%

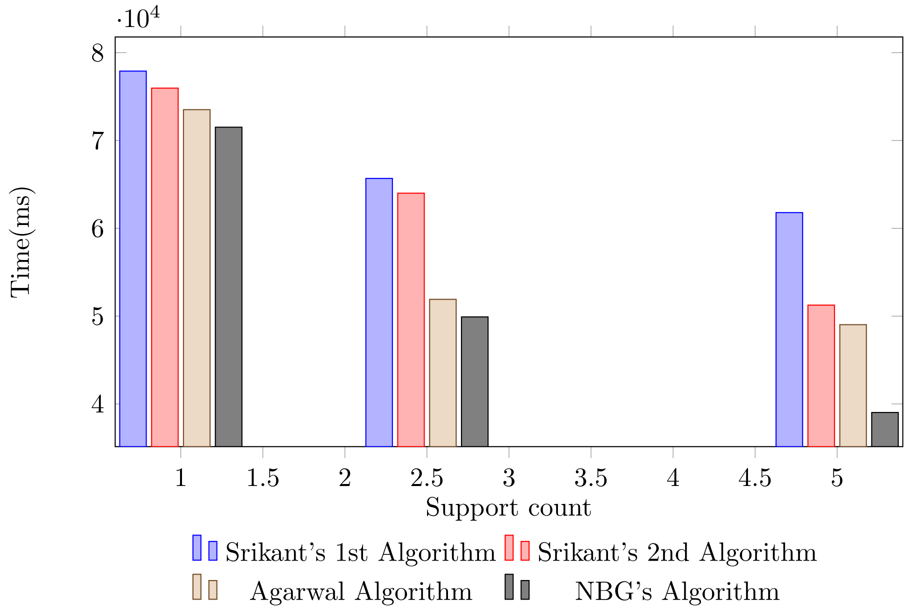

Figure 5-10: Time(ms) consumption of rule generation algorithms using array itemset representation for Hunington's dataset with confidence 1% and support count 1%, 2.5% and 5%

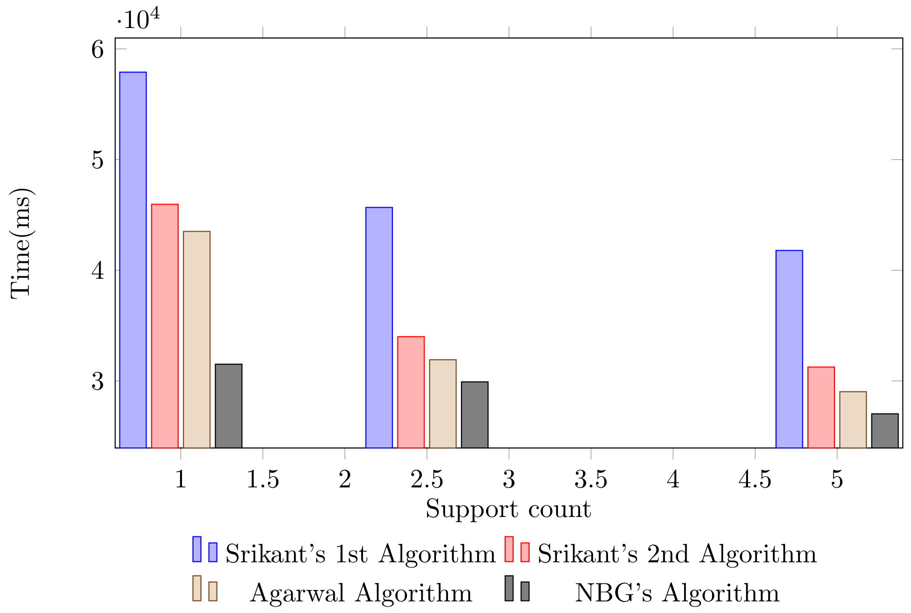

Figure 5-11: Time(ms) consumption of rule generation algorithms using array itemset representation for Hunington's dataset with confidence 2.5% and support count =1%, 2.5% and 5%

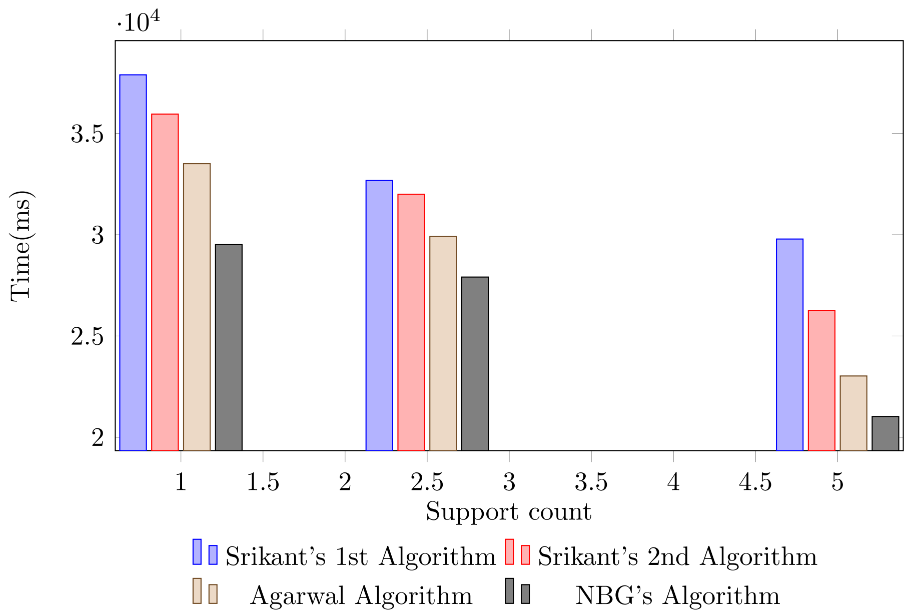

Figure 5-12: Time(ms) consumption of rule generation algorithms using array itemset representation for Hunington's dataset with confidence of 5% and support count= 1%, 2.5% and 5%

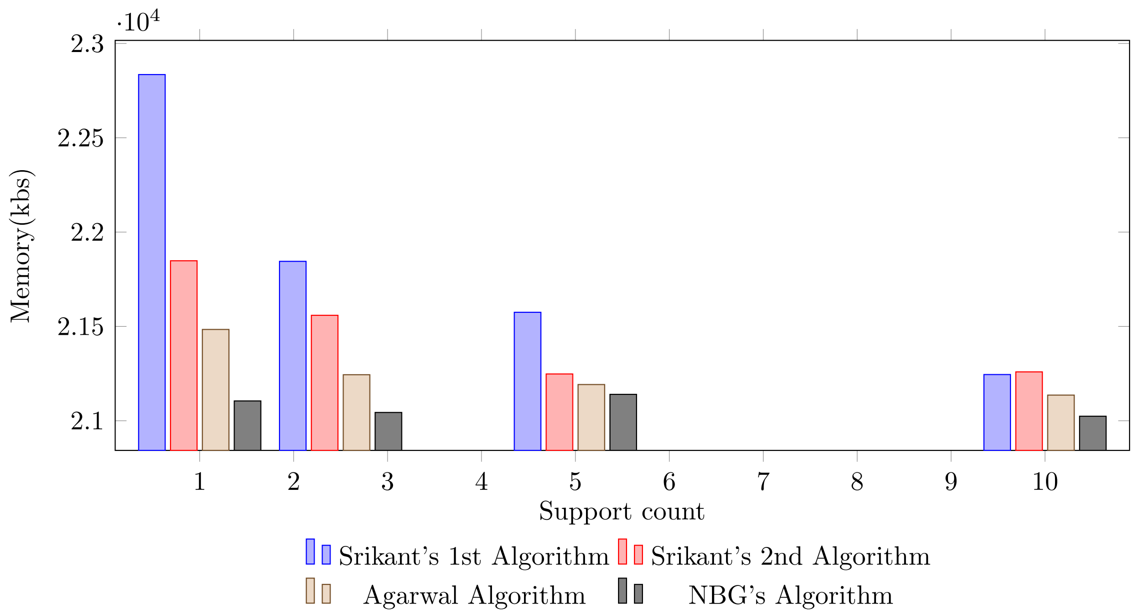

Figure 5-13: Memory(kilobits) consumption of rule generation algorithms using array representation for 1^{st} dataset with support count=1%, 2.5%, 5% and 10%

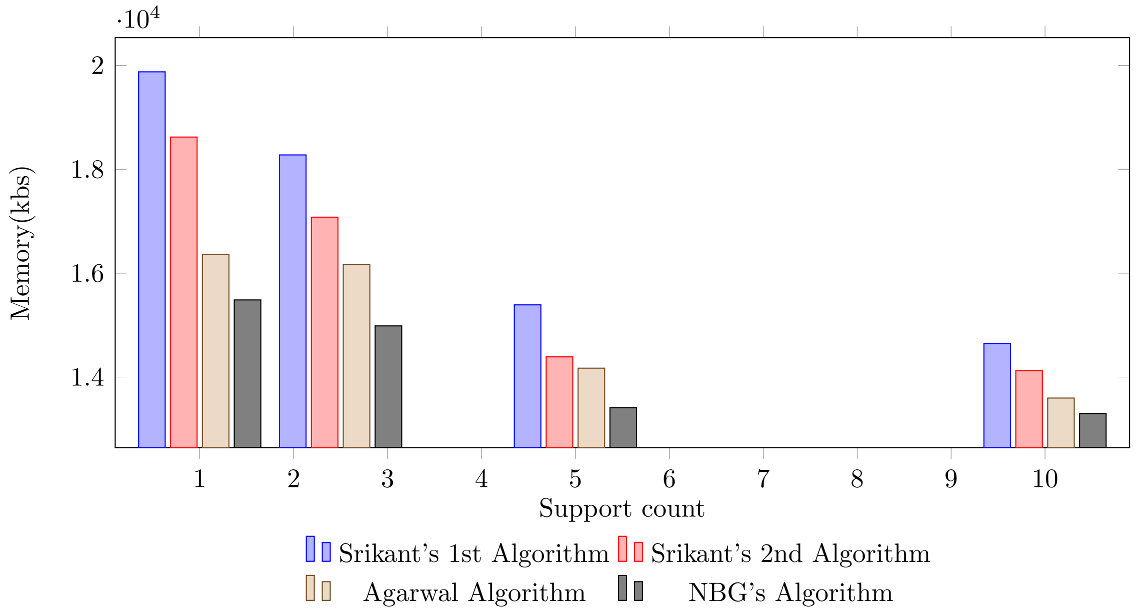

Figure 5-14: Memory(kilobits) consumption of rule generation algorithms using proposed itemset representation for 1^{st} dataset with support count =1%, 2.5%, 5% and 10%

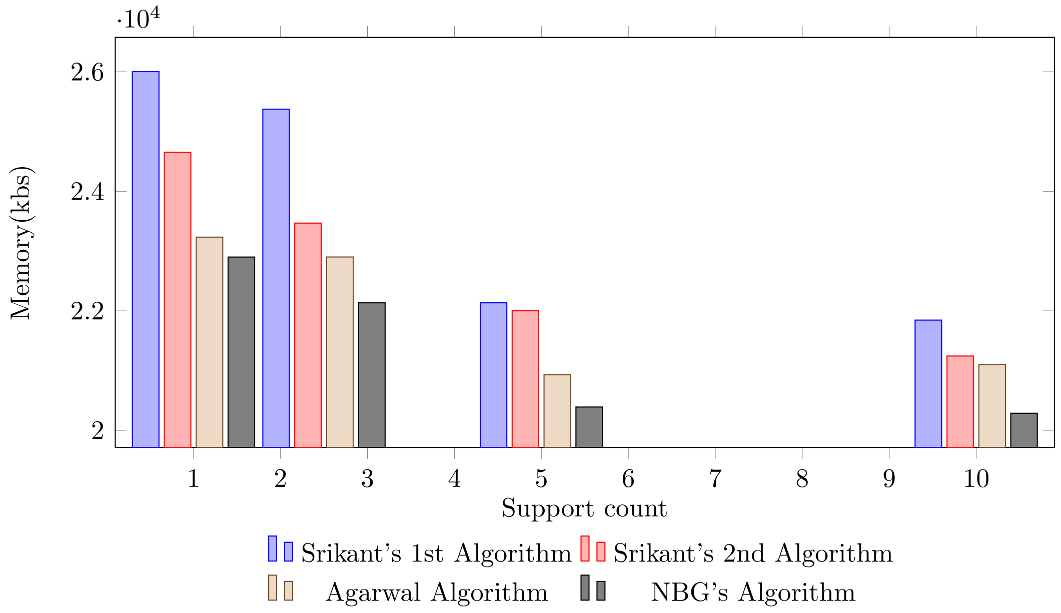

Figure 5-15: Memory(kilobits) consumption of rule generation algorithms using array representation for 2^{nd} dataset with support count=1%, 2.5%, 5% and 10%

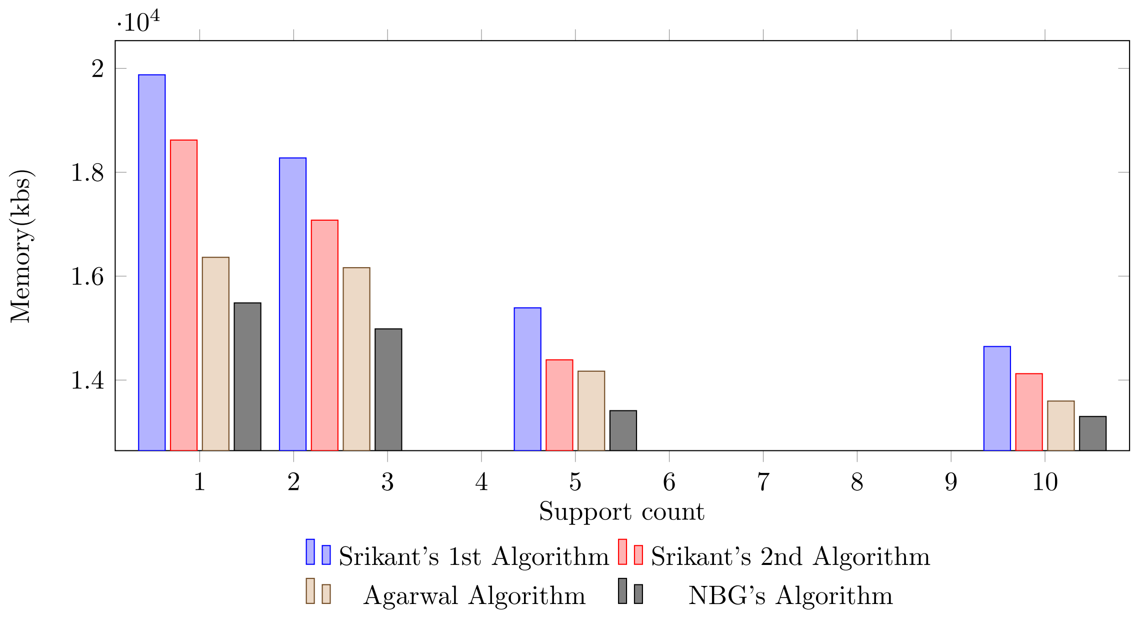

Figure 5-16: Memory(kilobits) consumption of rule generation algorithms using proposed representation for 2nd dataset with support count= 1%, 2.5%, 5% and 10%

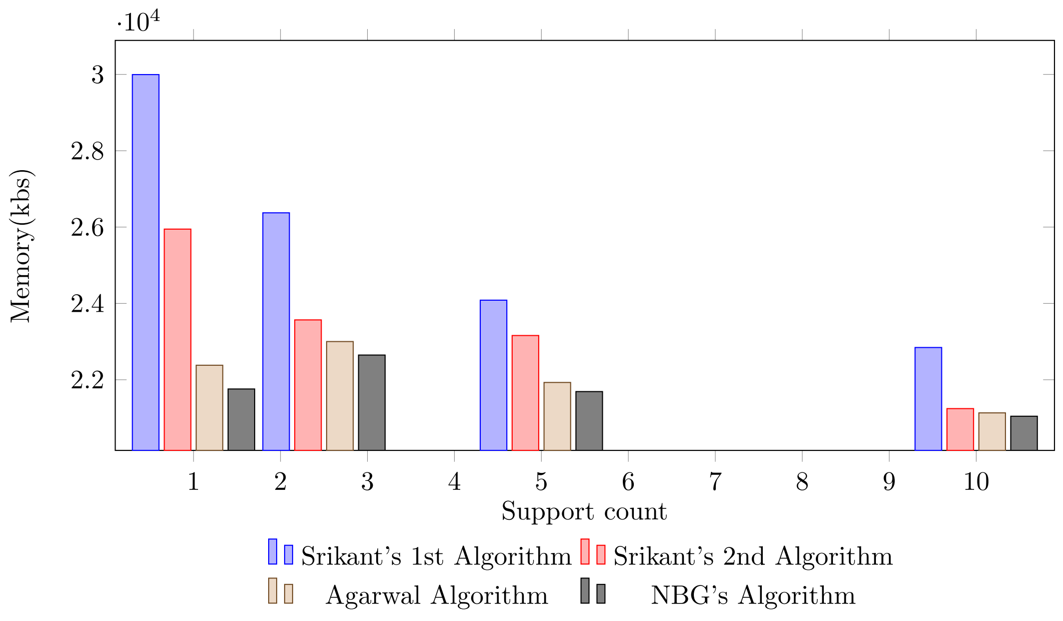

Figure 5-17: Memory(kilobits) consumption of rule generation algorithms using array representation for 3rd dataset with support count=1 %, 2.5%, 5% and 10%

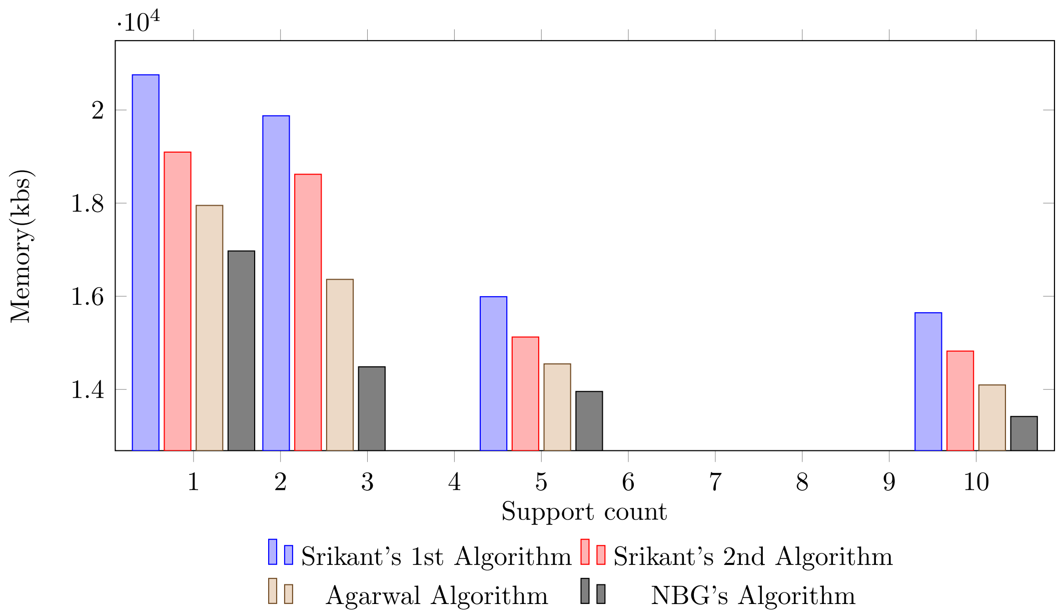

Figure 5-18: Memory(kilobits) consumption of rule generation algorithms using proposed itemset representation for 3^{rd} dataset with support count=1%, 2.5%, 5% and 10%

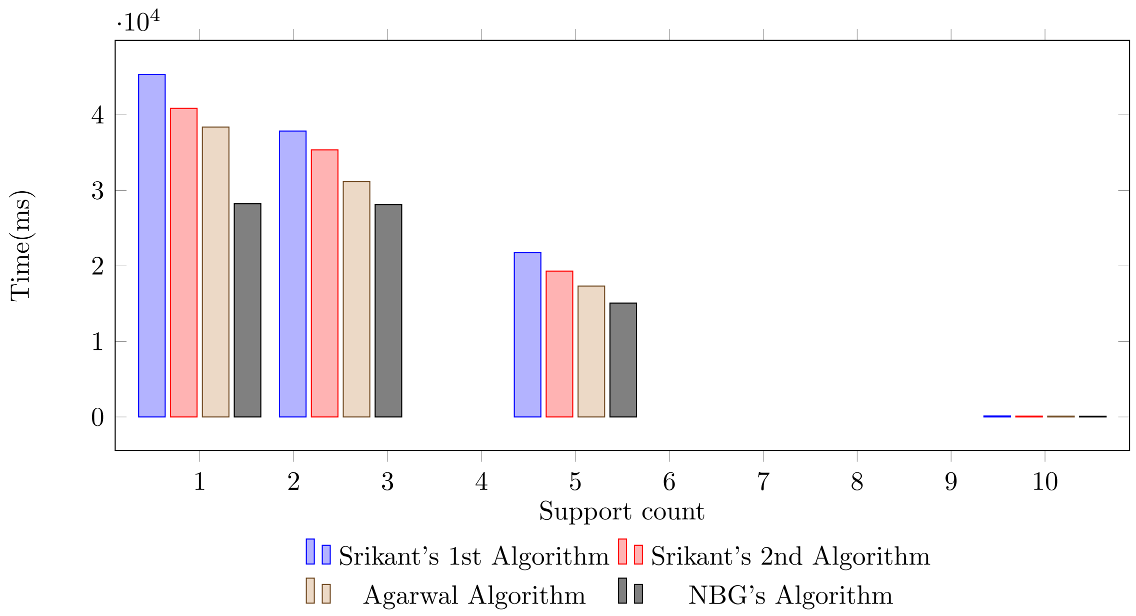

Figure 5-19: Time(ms) consumption of rule generation algorithms using array representation for 1^{st} dataset with support count=1%, 2.5%, 5% and 10%

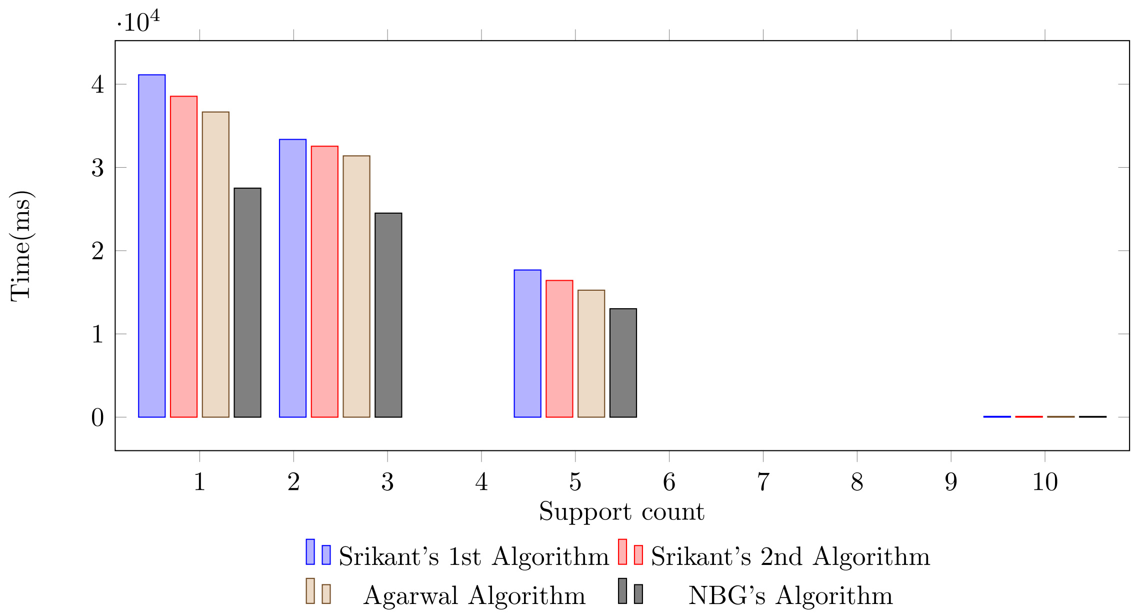

Figure 5-20: Time(ms) consumption of rule generation algorithms using proposed itemset representation for 1st dataset with support count=1%, 2.5%, 5% and 10%

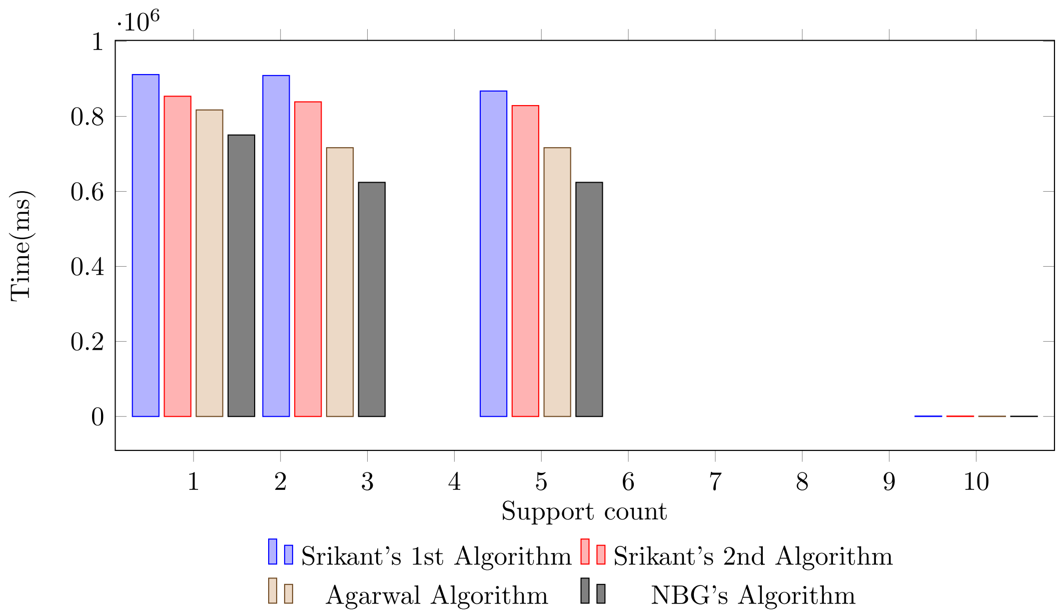

Figure 5-21: Time(ms) consumption of rule generation algorithms using array representation for 2nd dataset with support count= 1%, 2.5%, 5% and 10%

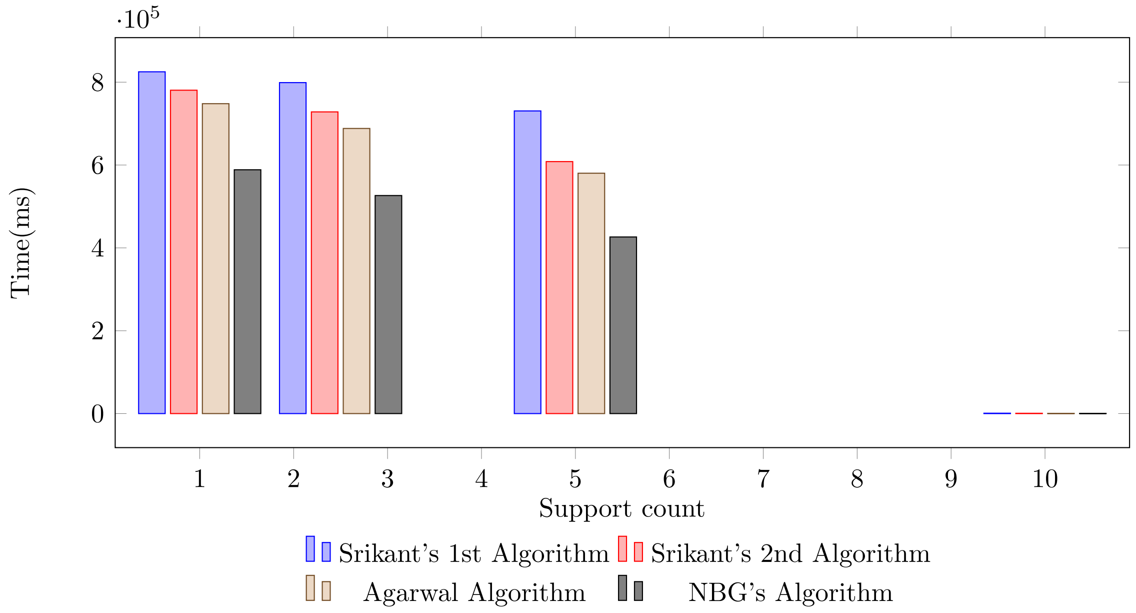

Figure 5-22: Time(ms) consumption of rule generation algorithms using proposed representation for 2^{nd} dataset with support count= 1%, 2,5 %, 5% and 10%

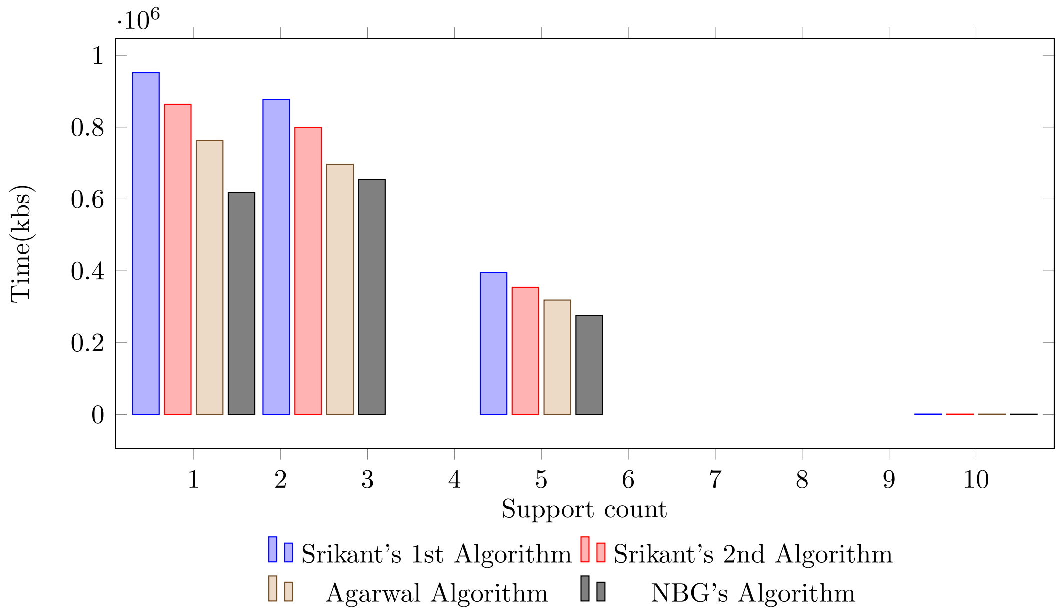

Figure 5-23: Time(ms) consumption of rule generation algorithms array representation for 3^{rd} dataset with support count =1%, 2.5%, 5% and 10%

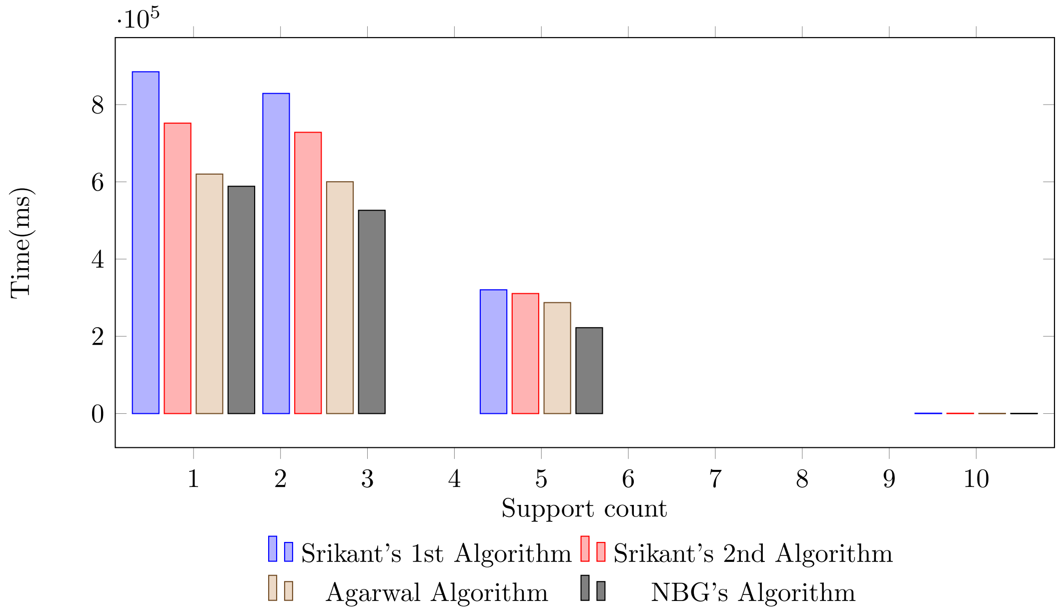

Figure 5-24: Time(ms) consumption of rule generation algorithms proposed itemset representation for 3^{rd} dataset with support count=1%, 2.5%, 5% and 10%

- Religious dataset: The dataset [3] were obtained from the UCI machine learning repository consisting of 8265 attributes and 590 instances.
- Corel Dataset: The dataset [4] consists of the features extracted from a Corel image. The dataset consists of 68040 instances and 89 attributes.

The execution time and memory consumption of the itemset represented using array representation for different datasets show different behavior. It has been observed that with different sizes and characteristics, the execution time and memory consumption also change. From the results shown in Figures 5-1 to 5-24, the algorithms incorporating the proposed itemset representation takes less time and memory as compared to the array representation. NBG's algorithm incorporated with proposed itemset representation outperforms the other three algorithms in terms of execution time and memory consumption.

5.3 Rule Generation with Proposed Representation

The proposed itemset representation is applied to the datasets mentioned in the above section. The generated frequent itemsets are then used for generating rules.

[3]https://archive.ics.uci.edu/ml/datasets/A+study+of+Asian+Religious+and+Biblical+Texts
[4]https://archive.ics.uci.edu/ml/datasets/Corel+Image+Features/

From the results 5-1 to 5-24, it is observed that NBG's Algorithm takes less time and memory irrespective of the dataset size and support count. The reason is that it uses the already existing frequent itemset present in the memory and does not produce the subsets of the frequent itemset. The NBG's algorithm is a better competitor as compared to Agarwal's algorithm, Srikant's 1^{st}, and Srikant's 2^{nd} algorithm since it can tackle the issues faced by the other three algorithms and generate all the rules with even less time. The number of generated rules for each dataset using NBG's algorithm are shown in Table 5.1. For example, the rules generated from the Hunington's dataset are as follows:-

- Repeated CAG Chain → Hunington's disease
- Patient having mental disease → Hunington's disease

It is evident from these rules that a patient having repeated CAG Chain is prone to Hunington's disease. Although, mental disease is not mostly neglected by medical expert. However, the rule shows that mental illness is also one of the symptoms of Hunington's disease. Few examples of rules generated from Religious dataset are as follows:-

- BookOfProverb_Ch15 → fools, joy, foolishness
- fools → joy, foolishness

From the above rules, it is concluded that from the Book of Proverbs, that a fool will only rejoice in foolishness.

5.4 Discussion and Conclusion

The performance of the proposed itemset and array representation is observed in different datasets with distinct sizes and features. It has been observed that the execution time and memory consumption using the proposed itemset representation for Apriori performs better than the itemset represented using array representation.
From Table 5.1, the rules are extracted using NBG's Algorithm that incorporates the proposed itemset representation outshines that of the rules generated from the frequent itemset that uses array representation. Thus, the efficiency of the rule generation algorithms increases when the itemsets are represented using the proposed itemset representation.

Table 5.1: Number of generated rules represented by Array and Proposed itemset representation using NBG's Algorithm

Sl.No	Dataset	Support	Confidence	Number of generated rules using NBG's Algorithm	
				Proposed Representation	Array Representation
1.	Huntington's	1%	100%	137	137
			98%	130	130
			95%	130	130
			90%	127	127
		2.5%	100%	80	80
			98%	79	79
			95%	79	79
			90%	78	78
		5%	100%	16	16
			98%	16	16
			95%	16	16
			90%	16	16
2.	Corel	1%	100%	312	312
			98%	311	311
			95%	310	310
			90%	310	310
		2.5%	100%	110	110
			98%	110	110
			95%	109	109
			90%	106	106
		5%	100%	43	43
			98%	43	43
			95%	42	42
			90%	42	42
3	Religious	1%	100%	55	55
			98%	55	55
			95%	55	55
			90%	55	55
		2.5%	100	15	15
			98%	14	14
			95%	13	13
			90%	13	13
		5%	100%	8	8
			98%	8	8
			95%	7	7
			90%	7	7

4	Chess	1%	100%	1556	1556
			98%	1555	1555
			95%	1555	1555
			90%	1555	1555
		2.5%	100%	801	801
			98%	801	801
			95%	800	800
			90%	800	800
		5%	100%	90	90
			98%	88	88
			95%	88	88
			90%	87	87