UNIVERSITY OF CALIFORNIA,
IRVINE

A Scalable Association Rule Learning Algorithm for Large Datasets and Its Application to Microarray Datasets

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Engineering

by

Haosong Li

Dissertation Committee:
Professor Phillip Sheu, Chair
Professor Michael Carey
Assistant Professor Quoc-Viet Dang

2021

DEDICATION

This study is wholeheartedly dedicated to my parents, who helped me in all things great and small.

To my academic adviser Professor Phillip Sheu who guided me in this process.

And lastly, I dedicated this dissertation to Tingan Jin, who supported me in the preparation of this dissertation during the hardship of COVID-19.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Part of the text of this dissertation is a reprint of the material[29] as it appears in Journal of Big Data. The co-author listed in this publication directed and supervised research which forms the basis for the dissertation.

VITA
Haosong Li

2018	B.S. in Electrical Engineering, University of California, Irvine
2018-21	Teaching Assistant, The Henry Samueli School of Engineering, University of California, Irvine
2020-21	Teaching Assistant, The Donald Bren School of Information and Computer Sciences, University of California, Irvine
2020	M.S. in Computer Engineering, University of California, Irvine
2021	Ph.D. in Computer Engineering, University of California, Irvine

FIELD OF STUDY

Data Mining, Association Rule Learning, Artificial Intelligence, Natural Language Database

ABSTRACT OF THE DISSERTATION

A Scalable Association Rule Learning Algorithm for Large Datasets with Focus on Microarray Datasets
by

Haosong Li

Doctor of Philosophy in Computer Engineering

University of California, Irvine, 2020

Professor Phillip Sheu, Chair

Many algorithms have solved the association rule learning problem. However, most of these algorithms suffer from the problem of scalability either because of tremendous time complexity or memory usage, especially when the dataset is large and the minimum support (minsup) is set to a lower number.

Among others, association rule learning algorithms have been applied to microarray datasets to find association rules among genes. With the development of microarray technology, larger datasets have been generated recently that challenge the current association rule learning algorithms. Specifically, the large number of items per transaction significantly increases the running time and memory consumption of such tasks.

In this dissertation, we solve the above problems by introducing a new approach that follows the divide-and-conquer paradigm, which can exponentially reduce both the time complexity and memory usage, even on a single machine. It is shown from comparative experiments that the proposed heuristic approach has significant speedup over existing algorithms. The heuristic approach, with some modification, efficiently learns gene-disease association rules and gene-gene association rules from large-scale microarray datasets. The rules are ranked based on their importance. Our experiments show our algorithm outperforms the Apriori algorithm on microarray datasets by one to three orders of magnitude.

Chapter I: INTRODUCTION

The association rule learning problem has played a significant role in data mining for the past few decades. Association rules are widely used in many fields, including market basket analysis [27] and bioinformatics [36]. However, the problem has an NP-hard nature, meaning it is challenging to find the results within a reasonable period of time.

The invention of the Apriori Algorithm [1] made this problem computationally feasible for most computers on regular-sized datasets. Since then, researchers have continued to develop more scalable algorithms. Among others, FP-Growth [20] and Eclat [45] are two algorithms developed that improve the scalability of the Apriori algorithm.

The increasing popularity of the Internet in recent decades has made big data available to many research institutions and companies. Their sizes are so large that traditional algorithms may not be able to handle them efficiently. We consider big data to be datasets that, at least, are too large to fit into the memory and takes a very long time(hours or even days) for traditional algorithms to finish computation. The term big data is thus relevant to the machine. A dataset considered to be big data on a PC may be a small dataset on a powerful high-performance computer (or computing cluster). This imposes a challenge to the association rule learning problem as well. Most of the previously designed algorithms, including the Apriori algorithm, the FP-Growth algorithm, and the Eclat algorithm, suffer from the problem of scalability for big data. Still, these algorithms take an unacceptable amount of time to terminate (will be discussed in the experiments section). In addition, the FP-Tree of the FP-Growth algorithm, and the TID list of the Eclat algorithm may not fit in the memory.

This research introduces an approach that makes it possible to mine association rules and frequent itemsets for large datasets. The approach, called the Scalable Association Rule Learning (SARL) heuristic, follows the divide-and-conquer paradigm and it vertically divides a dataset into almost equivalent partitions using a graph representation and the k-way graph partitioning algorithm [2]. The total time complexity of the SARL heuristic, including the overhead of partitioning a dataset, is up to 2d faster than that of the Apriori algorithm, where d is the number of unique items in the dataset. The memory usage is also lower than those of the current algorithms. Because of the speedup, our heuristic may be applied to real-time data analysis that can benefit many scientific [41] and other applications [44] [14].

We also studied how the SARL heuristic may be applied to microarray data. Microarray technology has been widely used in bioinformatics. It efficiently measures gene expression levels for a large number of genes. Therefore, a huge amount of data can be generated from microarray datasets. Microarray datasets, after converting to transactional datasets, usually have a large number of columns (genes) and a small number of rows (assays). Since the time complexity of any precise association rule learning algorithm is 2^d, where d is the number of unique items (genes, in this case), such a large number of genes causes a huge challenge for all existing association rule learning algorithms. For a large microarray dataset, it is impractical to apply these algorithms to find all association rules.

Researchers interested in deriving association rules from microarray datasets are most likely not to use every association rule. Furthermore, a large number of genes also results in an even higher number of association rules. The computer memory, which is limited compared to the disk space, can easily be used up to run a current association rule learning algorithm. With these in mind, we propose a modified Scalable Association Rule Learning (SARL) algorithm that focuses on the learning speed and the importance of rules derived.

As more microarray datasets are generated every day, investigators seeking potential associations between genes and between genes and diseases need a tool to find candidate rules across multiple datasets quickly. SARL is such a tool that provides scalable association rule learning and rule ranking. After having a general idea of candidate rules, investigators may choose to run a more time-costly algorithm that precisely calculate the rules on a few selected datasets. Therefore, by quickly reducing the scope of datasets and giving a general idea to the investigator, our algorithm can reduce the total time needed to find a target rule and increases the success rate.

The rest of the dissertation is organized as follows. In the related work section, we survey existing association rule learning algorithms and graph partitioning algorithms as well as the application of association rule learning algorithms in microarray datasets. In our solution section, we present the SARL heuristic and algorithm with examples, formal descriptions, theorems, and proofs. We also cover the application of SARL in microarray datasets in this section. In the next section, the experiments and results are presented, followed by conclusions and future work.

Chapter II: CONTRIBUTIONS

The contributions of this dissertation include the provision of association graphs that represent an efficient estimation of potential frequent itemsets and the use of the MLkP algorithm to divide the items into partitions while minimizing the loss of information. In the context of microarray dataset, the rule ranking algorithm calculates the importance and ranks the rules, so the investigator does not have to search through millions of rules. We consider gene-disease associations. Each important association rule between genes and disease can be identified and highlighted in the result.

Chapter III: RELATED WORK

Association rule learning/frequent itemset mining has been an active research area. Among others, three approaches are considered the most popular and possibly the most efficient: the Apriori algorithm, the FP-Growth algorithm, and the Eclat algorithm.

1. The Apriori Algorithm

The Apriori algorithm [1], introduced by Agrawal and Srikant, was the first efficient association rule learning algorithm. It incorporates various techniques to speed up the process as well as to reduce the use of memory. For example, the Lk-1 × Lk-1 method used in the candidate generation process can reduce the number of candidates generated, and the pruning process can significantly reduce the number of possible candidates at each level.

One of the most important mechanisms in the Apriori algorithm is the use of the hash tree data structure. It uses this data structure in the candidate support counting phase to reduce the time complexity from O(kmn) to O(kmT+n), where k is the average size of the candidate itemset, m represents the number of candidates, n represents the number of items in the whole dataset, and T is the number of transactions.

The major advantage of the Apriori algorithm comes from its memory usage because only the k-1 frequent itemsets, Lk-1, and the candidates in level k, Ck, need to be stored in the memory. It generates the minimum number of candidates based on the L_(k-1)×L_(k-1) (described in [1]) and the pruning method, and it stores them in the compact hash tree structure. In case the candidates fill up the memory from the dataset and a low minsup setting, the Apriori algorithm does not generate all the candidates to overload the memory. Instead, it generates as many candidates as the memory can hold.

2. The FP-Growth Algorithm

The Frequent Pattern Growth algorithm was proposed by Han et al. in 2000 [20]. It uses a tree-like structure (called Frequent Pattern Tree) instead of the candidate generation method used in the Apriori algorithm to find the frequent itemsets. The candidate generation method finds the candidates of the frequent itemsets before reducing them to the actual frequent itemsets through support counting.

The algorithm first scans a dataset and finds the frequent one itemsets. Then, a frequent pattern tree is constructed by scanning the dataset again. The items are added to the tree in the order of their support. Once the tree is completed, the tree is traversed from the bottom, and a conditional FP-Tree is generated. Finally, the algorithm generates the frequent itemsets from the conditional FP-Tree.

The FP-Growth algorithm is more scalable than the Apriori algorithm in most cases since it makes fewer passes and does not require candidate generation. However, it suffers from memory limitations since the FP-Tree is fairly complex and may not fit in the memory. Traversing the complexed FP-Tree may also be time-expensive if the tree is not compact enough.

3. The Eclat Algorithm

Different from the Apriori algorithm and the FP-Growth algorithm that work on horizontal datasets (e.g., T001: {1, 3} T002:{1, 4}), the Eclat (Equivalence Class Clustering and bottom-up Lattice Traversal) algorithm [45] uses a vertical dataset (e.g. Item1: {T001, T002}, Item3: {T001}, Item4:{T002}). The Eclat algorithm only scans the dataset once. It finds the frequent itemsets by taking the intersections of the transaction sets.

The Eclat algorithm takes advantage of scanning the dataset only once. However, when the dataset is large, and the minsup is set to a low value, the TID associated with each itemset may become very long. In fact, the results can be larger than the original dataset; therefore, they may not fit into the memory.

4. Other Association Rule Learning Algorithms

There are three categories of association rule mining/frequent itemset mining algorithms [10]: Apriori-based algorithms, tree-based algorithms, and pattern growth algorithms. The Apriori algorithm, the Eclat algorithm, and the FP-Growth algorithm are the most popular algorithms for the three categories, respectively.

In the Apriori-based algorithm category, proposed by Agrawal and Srikant in [1] the AprioriTID algorithm is similar to Apriori, except that it generates Ck-bar and it mines the frequent itemsets from there instead of the dataset. The Apriori Hybrid algorithm [1] is a combination of the Apriori algorithm and the AprioriTID algorithm. The DHP (direct hashing and pruning) algorithm [37] uses a hash function to distribute the itemsets into buckets. If a bucket has the support lower than the minsup, then the bucket is discarded. The MR-Apriori [31] and HP-Apriori [35] algorithms are distributed versions of the Apriori algorithm. The MR-Apriori uses the MapReduce model on the Hadoop platform. They enable parallel execution of the Apriori algorithm.

The tree-based algorithms, represented by the Eclat algorithm, find the frequent itemset by constructing a lexicographic tree. The AIS algorithm [2] and the SETM algorithm

[22] are the two earliest association rule mining algorithms in this category. Reference [1] shows that the Apriori algorithm beats them in running time. The TreeProjection algorithm [3] counts the supports of the frequent itemsets and uses the nodes of a lexicographic tree as the representation of these support numbers. The TM algorithm [40] maps the TID of each transaction to transaction intervals before performing intersections between these intervals.

Lastly, the algorithms in the pattern growth category focus on frequent patterns. The P-Mine algorithm [7] is a parallel computing algorithm that utilizes the VLDBMine data structure to store the dataset and speed up the distribution of data, while the LP-Growth algorithm [38] makes use of an array-based linear prefix tree to improve the memory efficiency. The Can-Mining algorithm [21] finds the frequent itemsets from a canonical-order tree, which speeds up the tree traversal process when the number of frequent itemsets is low. Finally, the EXTRACT algorithm [16] uses the theory of Galois lattice to derive association rules.

The algorithms discussed above, unfortunately, have scalability problems. The Apriori-based algorithms, represented by the Apriori algorithm, have to go through the expensive candidate generation and support counting process. This causes a disadvantage in running time. The tree-based and the pattern-growth type algorithms often suffer from excessive usage of memory. For example, the FP-Growth algorithm could build a complex FP-Tree which does not fit into the memory.

We show the scalability problems of the Apriori algorithm and the FP-Growth algorithm in the experiment part of this dissertation. Both of the algorithms take too long to finish for most of the tested datasets. The need for faster, frequent itemset mining is urgent due to the vastly available data today. Companies and institutions have allocated many resources in data mining, and they need a time-saving, resource-saving solution. In addition, real-time data analysis plays an important role in government [11], scientific [41], and other [44] [14] applications. The experiments part of this dissertation shows that the current algorithms represented by the Apriori algorithm and the FP-Growth algorithm are not fast enough to complete real-time data analysis. The scalability problems of most existing association rule mining algorithms have also been addressed in [43] that is focused on paralleled computing of association rules whereas this dissertation presents a scalable algorithm that is suitable for a single machine also.

5. Graph Partitioning Algorithms

One of the key steps in the SARL heuristic that we will introduce shortly is to partition the IAG (item association graph, Section 4, part 7) into k balanced partitions. An efficient graph partitioning algorithm is crucial since the balanced graph partitioning problem is NP-complete [8]. We have implemented three algorithms and compared them for the partitioning costs and running times. They are the recursive version of the Kernighan-Lin Algorithm [28], the Multilevel k-way Partitioning Algorithm (MLkP) [25], and the recursive version of the Spectral Partitioning Algorithm [33]. Other graph partitioning algorithms include the Tabu search-based MAGP algorithm [18].

The Kernighan-Lin algorithm swaps the nodes assigned to both partitions and finds the largest decrease in the total cut size. The Multilevel k-way Partitioning algorithm (MLkP) uses coarsening-partitioning-uncoarsening/refining steps to shrink a graph into a much smaller graph. After partitioning, the graph is rebuilt to restore the original graph. A single global priority queue is used for all types of moves. The Spectral Partitioning Algorithm finds splitting of the values such that the vertices in a graph can be partitioned with respect to the evaluation of the Fiedler vector.

Experiments are conducted by us to compare the three algorithms. The datasets provided by Christopher Walshaw at the University of Greenwich [42] are used. The datasets are as large as possible while the partitioning algorithms can finish in a reasonable time on the tested machine. We also run experiments on complete graphs with 30 and 300 nodes. Each dataset is tested four rounds with the number of partitions (k) being 2, 4, 8, and 16.

dataset	# of nodes	# of edges	avg degree	k	METIS Time	Spectral Time	METIS Cost	Spectral Cost	KL Time	KL Cost
3elt.graph	4720	13722	2.90720339	2	0.1083529	54.73113894	97	94	Timeout	N/A
3elt.graph	4720	13722	2.90720339	4	0.104274511	45.01839089	220	236	Timeout	N/A
3elt.graph	4720	13722	2.90720339	8	0.082687616	34.23122334	392	341	Timeout	N/A
3elt.graph	4720	13722	2.90720339	16	0.084682226	27.92868638	618	602	Timeout	N/A
add20.graph	2395	7462	3.11565762	2	0.041537523	3.044170141	719	80	Timeout	N/A
add20.graph	2395	7462	3.11565762	4	0.069754601	5.512359381	1296	350	Timeout	N/A
add20.graph	2395	7462	3.11565762	8	0.048701763	12.52986908	1874	1199	Timeout	N/A
add20.graph	2395	7462	3.11565762	16	0.054409027	29.25708413	2370	1647	Timeout	N/A
add32.graph	4960	9462	1.90766129	2	0.06651473	63.91381288	10	8	Timeout	N/A
add32.graph	4960	9462	1.90766129	4	0.064602375	54.39832783	43	33	Timeout	N/A
add32.graph	4960	9462	1.90766129	8	0.068125963	45.34366322	85	89	Timeout	N/A
add32.graph	4960	9462	1.90766129	16	0.069462299	87.3277657	182	136	Timeout	N/A
data.graph	2851	15093	5.293931954	2	0.075086355	19.99383068	219	115	Timeout	N/A
data.graph	2851	15093	5.293931954	4	0.069795132	14.5627892	495	262	Timeout	N/A
data.graph	2851	15093	5.293931954	8	0.094658613	7.923767567	713	392	Timeout	N/A
data.graph	2851	15093	5.293931954	16	0.089031458	6.522737265	1349	992	Timeout	N/A
uk.graph	4824	6837	1.417288557	2	0.05449748	347.5232875	26	11	Timeout	N/A
uk.graph	4824	6837	1.417288557	4	0.073782206	150.0673718	57	50	Timeout	N/A
uk.graph	4824	6837	1.417288557	8	0.056687832	102.6085541	107	82	Timeout	N/A
uk.graph	4824	6837	1.417288557	16	0.060199022	53.14980578	181	145	Timeout	N/A
Complete Graph	30	870	29	2	0.0555	0.3539	225	114	0.0068	225
Complete Graph	30	870	29	4	0.003133	0.0351	337	316	0.01329	337
Complete Graph	30	870	29	8	0.00318	0.0488	393	380	0.02685	393
Complete Graph	300	8700	29	2	0.214009	0.19758	22484	8339	3.4756	22500
Complete Graph	300	8700	29	4	0.207079	0.2026431	33741	28022	4.891045	33750
Complete Graph	300	8700	29	8	0.18528	0.19459	39372	38846	4.888	39374

Table 1 Results of the Experiment that Compare MLkP, Kernighan-Lin, and Spectral Partitioning Algorithms

As shown in Table 1, the running times are highlighted in the red box. We can tell from average running time(the last row) that the MLkP algorithm has the highest speed in general. It is 560 times faster than the spectral partitioning algorithm and even faster than the recursive Kernighan-Lin algorithm. The spectral partitioning algorithm has, in general, the best partition quality. It is 1.3 times better than MLkP and much better than the recursive Kernighan-Lin algorithm. The recursive Kernighan-Lin algorithm takes too long to complete all five datasets. It also shows serious scalability issues for complete graphs.

Considering the MLkP algorithm has the best overall performance, we choose to use this algorithm for graph partitioning in our algorithm.

6. Applications Related to Association Rule Learning Algorithms on Microarray Data

Both Apriori and FP-Growth algorithms have been applied to microarray datasets [17], according to [17]. The main challenge for applying an existing association rule algorithm to a microarray dataset is the large number of items per transaction. Almost all microarray datasets have significantly more genes than assays. The existing approaches transpose the microarray dataset into transactional datasets. After transposing the dataset, we have significantly more columns than rows. This greatly increases the complexity of the existing association rule algorithms because they are designed for datasets with more rows than columns. With the existing algorithms, it can be shown that the FP-Growth algorithm performs slightly (around 10%) better than the Apriori algorithm. However, the author points out that a more scalable algorithm is needed to overcome the time and space complexities.

There are variations of association rule learning algorithms on microarray datasets. The FARMER algorithm [12] finds interesting rule groups instead of individual rules. The algorithm is efficient for finding some association rules between genes and labels. Huang et al. propose a ternary discretization approach [23] that converts each gene expression level to one of the three levels: under-expressed, normal, and over-expressed. Compared to traditional binary classification methods, the ternary discretization approach captures the overall gene expression distribution to prevent serious information loss.

In summary, the existing variations of a traditional approach such as Apriori or FP-Growth algorithms have issues related to scalability or coverage. The algorithms and heuristics reported in this dissertation tolerate certain accuracy for better scalability so the investigator may navigate a dataset iteratively (We call it "iterative investigation" in this dissertation) to converge in the search process quickly.

Other Data Mining Algorithms on Microarray Data

In addition to association rule learning algorithms, other data mining algorithms that address different problems have also been applied to microarray data.

Many researchers have studied classification problems on microarray data. The most popular application is classifying diseases based on gene expression levels [15]. Many algorithms have been applied to solve classification problems. The most studied algorithms include the Bayesian network [46], Support Vector Machine [34], and k-Nearest Neighbor [30].

These problems usually take gene expression levels as the input (features) and predict the disease(s) associated with an assay. It can also be used to classify tumors based on gene expression levels.

Chapter IV: OUR SOLUTION

1. Definitions

Below are some definitions that we will use in our algorithm:

K-itemset: an itemset with k items

Support: the occurrence of an item in the dataset

Minsup: the minimum requirement of support. The user usually provides this. Itemsets with support < minsup are eliminated.

Confidence: the indication of robustness of a rule in terms of percentage.

Confidence($X \rightarrow Y$) = support($X \cup Y$)/support(X)

Minconf: the minimum requirement of confidence. The user usually provides this. Rules with confidence < minconf are eliminated.

Item-Association Graph: a graph structure that stores the frequent associations between pairs of items.

Balanced K-way Graph Partitioning Problem: Divide the nodes of a graph into k parts such that each part has almost the same number of nodes while minimizing the number of edges/sum of edge weights cut off.

E_g: Importance of gene g.

I_r: Importance of rule r.

2. A Scalable Heuristic Algorithm - SARL-Heuristic

The following is an outline of our scalable heuristic:

Step 1: Find frequent one and two itemsets using the Apriori algorithm (when minsup is high) or the direct generation method (when minsup is low).

Step 2: Construct the item association graph (IAG) from the result of step 1.

Step 3: Partition the IAG using the multilevel k-way partitioning algorithm (MLkP).

Step 4: Partition the dataset according to the result of step 3.

Step 5: Call the modified Apriori algorithm or the FP-Growth algorithm to mine frequent itemsets on each transaction partition.

Step 6: Find the union of the results found from each partition.

Step 7: Generate association rules by running the Apriori-ap-genrules on the frequent itemsets found from step 6.

2.1 An Example

Suppose the dataset shown in Table 2 is given and minsup is set to 0.1 (or 10%, or 7*0.1≈1 occurrence), and minconf is set to 0.7 (or 70%):

TID	Items
T000	1, 2
T001	1, 2, 3
T002	4, 5
T003	1, 4, 5
T004	2, 3
T005	1, 2, 3
T006	1, 4, 5

Table 2 Example Dataset 1

First, we use the Apriori algorithm to find the frequent two itemsets. As an intermediate step, the Apriori algorithm finds the frequent one-itemset first (shown in Table 3):

Frequent Itemsets	Support
{1}	5
{2}	4
{3}	3
{4}	3
{5}	3

Table 3 Frequent one Itemsets

The frequent two-itemsets are found afterward (shown in Table 4):

Frequent Itemsets	Support
{1, 2}	3
{1, 3}	2
{1, 4}	2
{1, 5}	2
{2, 3}	3
{4, 5}	2

Table 4 Frequent two Itemsets

Next, we transform the above frequent two-itemsets into an item association graph (IAG), shown in Figure 1:

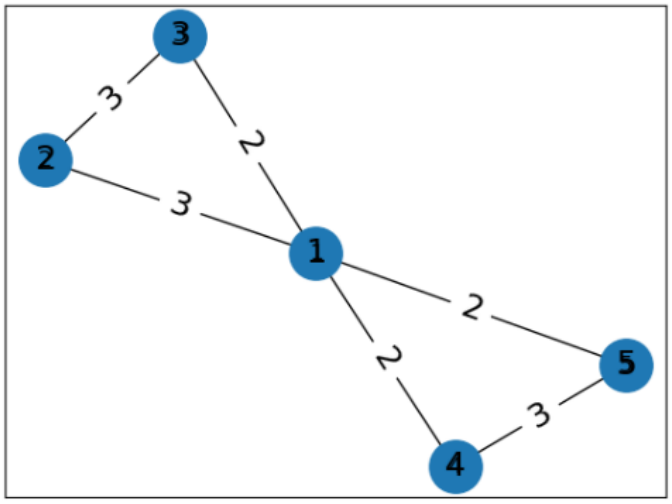


Figure 1 An Item Association Graph

To construct the graph, we first take the itemset {1, 2} with support 3. For this, we create node 1 and node 2 corresponding to the two items in the itemset. The edge between node 1 and node 2 has weight 3, representing the support of the itemset. The process is repeated for every frequent two-itemset found in the previous step.

Next, we use the multilevel k-way partitioning algorithm (MLkP) to partition the IAG. In this case, the number of nodes is small, so we only bisect the graph by setting k = 2. The result is shown in Figures 2 and 3.

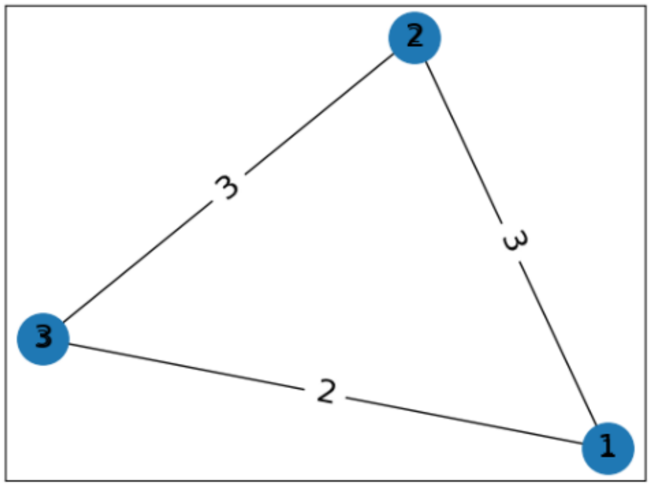


Figure 2 Item Association Graph Partition 1

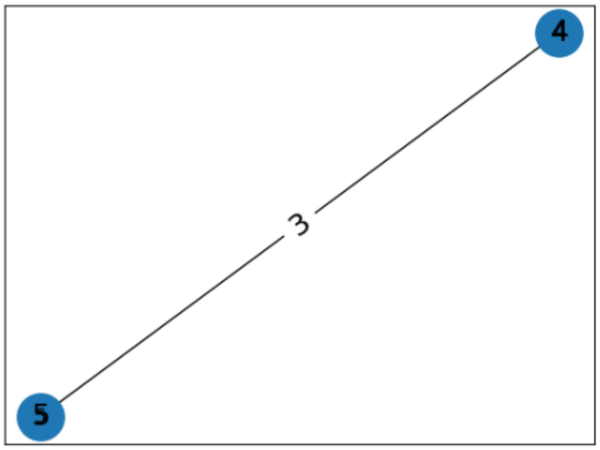


Figure 3 Item Association Graph Partition 2

The MLkP algorithm divides the IAG into two equal or almost equal sets in linear time while the sum of the weights of edges that are cut off is the minimum.

Next, we partition the dataset according to the partitions of the IAG, as shown in Tables 5 and 6. Each transaction partition has all the items from the corresponding IAG partition. However, since the algorithm has already found all frequent one and two itemsets, a transaction is not added to a transaction partition if the transaction has less than three items. For example, T000: {1, 2} is not added to the transaction partition 1, since it only has two items. Some items in the original dataset may not appear in any of the transaction partitions, because the infrequent one/two-itemsets are dropped in the IAG. This simplifies the subsequent computations. In this example, however, all the items are kept in the IAG because the IAG is a relatively dense graph. Tables 5 and 6 show the transaction partitions:

TID	Items
T001	1, 2, 3
T005	1, 2, 3

Table 5 Transaction Partition 1

TID	Items
None	None

Table 6 Transaction Partition 2

The next step is to pick the best algorithm and use it to find the frequent k-itemsets with k > 2. For this example, we choose the modified Apriori algorithm because it is faster for mining small datasets as it avoids the process of finding the one and two-itemsets again. The results from partition 1 are shown in Table 7:

Frequent Itemsets	Support
{1, 2, 3}	2

Table 7 Frequent Itemsets from Transaction Partition 1

Since the modified Apriori algorithm starts with three-itemsets, there are no additional frequent itemsets in the first partition. Table 8 shows the results found in transaction partition 2:

Frequent Itemsets	Support
None	N/A

Table 8 Frequent Itemsets from Transaction Partition 2

The final results (shown in Table 9) of frequent itemsets are simply the union of Tables 3, 4, 7, and 8:

Frequent Itemsets	Support
{1}	5
{2}	4
{3}	3
{4}	3
{5}	3
{1, 2}	3
{1, 3}	2
{1, 4}	2
{1, 5}	2
{2, 3}	3
{4, 5}	3
{1, 2, 3}	2

Table 9 Frequent Itemset Final Results

After running the Apriori-ap-genrules algorithm, the association rules can be found in Table 10.

Rules	Confidence
{2} → {1}	0.75
{3} → {2}	1
{5} → {1}	1
{2} → {3}	0.75
{5} → {4}	1
{4} → {5}	1
{1, 3} → {2}	1

Table 10 Association Rules Generated

All frequent itemsets generated by the SARL heuristic are sound, meaning each frequent itemset generated indeed is correct, and the support number is accurate. However, it is possible that some frequent itemsets cannot be found by the SARL heuristic, as will be discussed shortly. In this example, the SARL heuristic loses one frequent itemset {1, 4, 5} and two related rules generated from {1, 4, 5}.

2.2 Formal Description of the SARL Heuristic

```
SARL(dataset, minsup, minconf, k, threshold):
# if dataset is smaller than the threshold(explained later), then use direct_gen algorithm to get frequent one and two itemsets
if size(dataset) <= threshold:
    results, two_itemsets = direct_gen(dataset)

# otherwise, use modified Apriori
else:
results, two_itemsets = mod1-Apriori(dataset)

# build IAG from two itemsets
graph = build_IAG(two_itemsets)

# partition IAG with METIS that implements MLkP algorithm
partitions = METIS.partition(k, graph)

# partition dataset into smaller parts
parts = partition-dataset(partitions)

# repartition if some partitions cannot fit into the memory
while partition size > memory size:
    k+=1
partitions = METIS.partition(k, graph)
    parts = partition-dataset(partitions)

# compute 3+ frequent itemsets
for part in parts:
```

```
# when part is small, use Apriori
    if size(part) < threshold/k:
        results += mod2-Apriori(part, minsup, two_itemsets)
    # when part is large, use FP-Growth(defined in [20])
    else:
        results += FP-Growth(part, minsup)

# generate rules with Apriori-gen(defined in [1])
rules = Apriori-gen(results, minconf)

direct_gen(dataset, minsup):
# count the support of all one itemsets and store in C1
C1 = {}
for transaction in dataset:
    for item in transaction:
        if item not in C1:
            add item to C1
item.counter = 1
        else:
            item.counter += 1

# eliminate those itemsets with support < minsup and store in L1
L1 = {}
```

```
for candidate in C1:
    if candidate.counter >= minsup:
        add candidate to L1

# compute support of all two itemsets
C2 = {}
for trans in dataset:
    # for each two-item pair in a transaction
    for comb in combinations(trans, 2):
        if comb not in C2:
            add comb to C2
comb.counter = 1
        else:
            comb.counter += 1

# eliminate those two itemsets with support < minsup
L2 = {}
for candidate in C2:
    if candidate.counter >= minsup:
        add candidate to L1



mod1-Apriori(dataset, minsup):
```

```
# count the support of all one itemsets and store in C1
C1 = {}
for transaction in dataset:
    for item in transaction:
        if item not in C1:
            add item to C1
item.counter = 1
        else:
            item.counter += 1
# eliminate those itemsets with support < minsup and store in L1
L1 = {}
for candidate in C1:
    if candidate.counter >= minsup:
        add candidate to L1


# generate two-itemset candidates from L1 and store in C2
C2 = {}
for itemset1 in L1:
    for itemset2 in L1:
        if itemset1 != itemset2:
            add itemset1 U itemset2 to C2
for transaction in dateset:
    for candidate in C2:
        if candidate.issubset(transaction):
            candidate.counter += 1

# eliminate those two-itemsets with support < minsup and store in L2
L2 = {}
for candidate in C2:
    if candidate.counter >= minsup:
        add candidate to L2
return L1, L2

build_IAG(itemsets):
for itemset in itemsets:
    graph.add_node(itemset[0])
graph.add_node(itemset[0])
graph.add_edge(itemset[0], itemset[1], weight += 1)
return graph

partition-dataset(partitions, dataset):
for transaction in dataset:
    for partition in partitions:
        intersect = parition intersect transaction
        if len(inersect) > 2:
add intersect to dataset_partition_i
```

```
return transaction partition names


mod2-Apriori(dataset, minsup, two_itemsets):
# generate frequent three-itemsets based on two_itemsets
results = []
Lk = {}
Ck = apriori-gen(two_itemsets) # apriori-gen is defined in [1]
for transaction in dataset:
    Ct = subset(Ck, t)
    for c in Ct:
        c.count += 1
for c in Ck:
    if c.count >= minsup:
        Lk.add(c)
result.append(Lk)


# generate frequent 3+ itemsets
while Lk-1 != {}:
    Lk = {}
    Ck = apriori-gen(two_itemsets)
for transaction in dataset:
        Ct = subset(Ck, t)
        for c in Ct:
                c.count += 1
for c in Ck:
        if c.count >= minsup:
                Lk.add(c)
result.append(Lk)
return results
```

2.2.1 Finding Frequent 2 Itemsets using the Apriori Algorithm or dirct_gen algorithm

The first step of the SARL heuristic is to find the frequent 2 itemsets efficiently.

Although the Apriori algorithm has scalability issues for very large datasets, it provides a fast and convenient feature to extract intermediate results and a tolerable speed for the first two passes.

The Apriori algorithm finds frequent itemset L_k for each k, and each L_k is stored separately. We run the Apriori algorithm until it finds L2, the frequent two-itemset. It first tries to find the frequent one itemsets by traversing the dataset and count the occurrence of each unique item. If the number of occurrences of an item is less than the minsup provided by the user, that item is eliminated from the list of frequent one-itemset. The frequent two itemsets are discovered based on the frequent one itemsets. The algorithm generates C2, the candidate sets for the frequent two itemsets, using $L_{k-1} \times L_{k-1}$:

```
$insert\ into\ C_k$
$select\ p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}$
```

$from\ L_{k-1}\ p, L_{k-1}\ q$

$where\ p.item_1 = q.item_1, \dots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1};$

This method generates a minimum number of candidates from the frequent one itemsets so that we can have fewer candidates to consider in the support counting phase. The Apriori algorithm also predicts and eliminates some infrequent itemsets before support counting by implementing the Apriori principle in the pruning step. If an item in C2 is not in L1, which means that the item is infrequent, so all the two itemsets that include this item are dropped. We modify the Apriori algorithm, so it terminates after L2 is found.

Another method to find frequent one and two itemsets are through direct counting and generation. The algorithm to find frequent one itemsets is the same as the Apriori algorithm. To find frequent two itemsets, we can simply find all two-item pairs in each transaction and count the occurrence of them. The advantage of this algorithm is that it does not require candidate generation from L1, and avoids much unnecessary membership testing during support counting. However, this method is not efficient on large datasets since it does not use pruning and saves all two itemsets.

In the SARL heuristic, we ask the user for a threshold of the dataset size. If the dataset is larger than the threshold, the SARL heuristic will use the modified Apriori algorithm. Otherwise, it will use the direct_gen algorithm to compute the frequent one and two itemsets.

2.2.2 Construction of the Item Association Graph

The item association graph G is constructed based on the two itemsets generated by the Apriori algorithm. G is an undirected, weighted graph. A node Vi is created for each unique item i in the two itemsets T with the maximum item number being n.

$$\{V\} = \{\cup_{i=0}^{n} V_i \,|i \in |T|\}$$

The edges E in graph G are formed for each itemset in T:

$$\{E\} = \{\bigcup_{i=0,j=0}^{n} E_{ij} \,|\{i,j\} \in T\}$$

The weight of each edge E_{ij} is equal to the support of itemset {i, j} in T:

$$W(E_{ij}) = Support(\{i,j\}) \mid \{i,j\} \in T$$

2.3 Partition the IAG using the Multilevel k-way Partitioning algorithm (MLkP)

The Multilevel k-way partitioning (MLkP) algorithm [25] is an efficient graph partitioning algorithm. The time complexity is O(E), where E is the number of edges in the graph, and the maximum load imbalance is limited to 3%.

The general idea of MLkP is to shrink (coarsen) the original graph into a smaller graph, then partition the smaller graph using an improved version of the KL/FM algorithm. Lastly, it restores (uncoarsen) the partitioned graph to a larger, partitioned graph.

METIS is a software developed by Karypis at the University of Minnesota [26]. It includes an implementation of the MLkP algorithm that takes a graph as the input and outputs groups of nodes separated after the partition.

2.3.1 Transaction Partitioning

Based on the results of the MLkP algorithm that divide the items into groups P1, P2,…,Pm, we can partition the transactions into the same number of groups, where each group D_i contains only the items in partition P_i. For a transaction to be included in D_i, it must have all the items from partition P_i. If a transaction includes more items than the items from partition P_i, only the items in P_i that are included in the transaction are added to D_i. That is, only a part of the transaction is added to D_i. As a result, each transaction in a transaction partition must be a subset of the corresponding transaction in the original dataset. If a transaction has less than three items, the transaction is not added. This is because we have already mined the one and two itemsets, and are only interested in itemsets that have 3 or more items. This optimization helps to reduce the size of transaction partitions.

$$D_i = \{\bigcup_{j=1}^{n} T_j \,|(T_j \rightarrow \mathrm{S_j} \,\cap P_i|S_j \in D)\}$$

In the above, D_i is transaction partition i, T_j is the transactions to be added to partition i, S_j is the jth transaction in the original dataset, P_i is the item partition i, and D is the original dataset.

Since the number of unique items in each partition is less than or equal to $\frac{number\ of\ nodes\ in\ IAG}{k}$ rather than $\frac{total\ number\ of\ unique\ items}{k}$, the size of each partition should be small compared to the original dataset. In rare cases, if the size of a transaction partition is greater than the memory size, the SARL heuristic can partition the IAG and the transactions again with k incremented by 1. This guarantees that each partition fits into the memory.

2.3.2 Selecting an Algorithm on Transaction Partitions

One of the benefits that come with our solution is that the association rule learning on each transaction partition can be optimized by using an algorithm that best fits the partition.

During the association rule learning on the partitioned datasets, we have three candidates that are considered efficient: the Apriori algorithm, the FP-Growth algorithm, and the Eclat algorithm.

Since the modified Apriori algorithm has already computed the one itemsets and two itemsets during the preparation phase, the candidate generation feature of the Apriori algorithm is handy in this case. We modify the Apriori algorithm to skip the frequent one/two itemsets finding stages and start with the frequent three itemsets from the transaction partitions. This modification is particularly helpful when the minsup is set to a high value so that the expected number of itemsets is limited after the two itemsets are found.

We can estimate the expected number of itemsets from the average transaction length of each transaction partition. A higher average transaction length indicates a higher possibility of the presence of a long "tail" in the result. Results with long tails have itemsets with considerable maximum lengths, while results with short tails only contain itemsets with small maximum lengths. A dataset with an expected long tail means the association rule learning algorithm does not terminate soon after the two itemsets are found.

The average transaction length provides a fast and straightforward reference for selecting the best algorithm for each transaction partition. If the average transaction length is low, the Apriori algorithm can be the right choice, as the modified Apriori algorithm continues from the two itemsets that the preparation phase has already calculated. If the average transaction length is high, we can take advantage of the scalability of the FP-Growth algorithm. We omit the Eclat algorithm because the FP-Growth and the Eclat algorithms do not have the same advantage provided by the modified Apriori algorithm, of which the algorithm can start with the two itemsets. In addition, studies [24] show that the Eclat algorithm is slightly less scalable than the FP-Growth algorithm.

Next, the selected algorithm is used to find the frequent local itemsets from the given transaction partition. After the algorithm terminates, a simple union is performed on the frequent itemsets found from each partition. Finally, Apriori-ap-genrule is used to derive the rules from the frequent itemsets. This step is relatively simple.

2.4 Time Complexity and Space Complexity

The theoretical time and space complexity of the Apriori algorithm is $O(2^d)$ where d is the number of unique items in the dataset.

2.4.1 Time Complexity

The theoretical time complexity of the SARL heuristic consists of the complexity of several parts:

2.4.2 2-itemsets generation

Finding frequent 2-itemsets requires finding 1-itemsets first. This step is simply $O(n)$ as the algorithm traverses the dataset once. Next, the candidate generation for 2-itemsets takes $O(d^2)$ where d is the number of unique items in the dataset. Finally, the support checking requires $O(n + d^2T)$ where T is the number of transactions in the dataset. Therefore, the time complexity of this step is $O(d^2T + n)$.

2.4.3 IAG construction

Since each edge in the IAG is a representation of a frequent two-itemset, and the maximum number of two-itemsets is $\frac{d^2+d}{2}$, the maximum number of edges in IAG is also $\frac{d^2+d}{2}$. Therefore, constructing the IAG takes $O(d + \frac{d^2+d}{2})$ or $O(d^2)$.

2.4.4 IAG partition

The time complexity of the IAG partition process is equal to the time complexity of the MLkP algorithm, which is $O(E)$ or $O(d^2)$.

2.4.5 Transaction partition

The dataset is traversed once to assign items into different partitions. Hence the time complexity is O(n).

2.4.6 Running a selected algorithm

The algorithm selection requires the calculation of the average transaction width of each transaction partition. The time complexity of this is $O(kn)$, where k is the number of partitions.

If the modified Apriori algorithm is selected, the theoretical time complexity for each partition is $O(2^{1.03d/k})$ where the coefficient 1.03 comes from the 3% maximum imbalance of the partitions caused by the MLkP algorithm. The total running time for all the partitions is $O\left(k * 2^{\frac{1.03d}{k}}\right) \rightarrow O(2^{\frac{1.03d}{k}})$, and the total time complexity of the SARL algorithm, when the modified Apriori algorithm is selected, is $O\left(d^2T + n + d^2 + d^2 + n + 2^{\frac{1.03d}{k}}\right) \rightarrow O(d^2T + n + 2^{\frac{1.03d}{k}})$. Assume $n \gg d$, and $2^{\frac{1.03d}{k}} \gg n$, the time complexity can be simplified to $O(2^{\frac{1.03d}{k}})$. Compared with the time complexity of the Apriori algorithm, the SARL is $O\left(\frac{2^d}{2^{\frac{1.03d}{k}}}\right) \rightarrow O(2^{\frac{k-1.03}{k}d})$ times faster than the Apriori algorithm. The exponential speedup comes from the smaller number of unique items in each transaction partition. The algorithm that is chosen to mine frequent itemsets from the transaction partitions only needs to consider a portion of all the items for each partition.

2.4.7 Space Complexity

Like time complexity, the space complexity of the SARL heuristic consists of the complexity of several parts:

2.4.8 2-itemsets generation

Finding the frequent two itemsets requires finding the one itemsets first. This step is $O(d)$, where d is the number of unique items in the dataset, as we need to keep at most d items in the memory. Next, the candidate generation step for the 2-itemsets takes $O(d^2)$ space for at most $\frac{d(d-1)}{2}$ frequent 2-itemsets as candidates. Finally, the support checking requires another $O(d^2)$ space to store the support numbers. Hence, this step requires $O(d^2)$ space.

2.4.9 IAG construction

Since each edge in the IAG is a representation of a frequent two-itemset, and the maximum size of the two-itemsets is $\frac{d^2+d}{2}$, the maximum number of edges in IAG is also $\frac{d^2+d}{2}$. Therefore, storing the IAG takes $O(d^2)$ space. This d^2space occupation only occurs when every unique item in the dataset is included frequent two-itemsets with every other unique item in the dataset. In most cases, the actual space required to store IAG is smaller than the memory size.

In rare cases, if the IAG cannot fit into the memory, then the Apriori algorithm and FP-Growth algorithm must have memory issues, too. For the Apriori algorithm, all frequent two-itemsets must be stored in the memory to generate the candidates in the next level, and the size of frequent two-itemsets is similar to the IAG. FP-Tree must be stored in the memory for the FP-Growth algorithm. The space complexity of the FP-Tree is also $O(d^2)$, however, all unique items need to be stored in the tree while only the unique items in the frequent two-itemsets need to be stored in the IAG. Therefore, IAG has a lower space complexity than the FP-Tree.

2.4.10 IAG partition

The space complexity of the IAG partition is equal to the space complexity of the MLkP algorithm, which is $O(E)$ or $O(d^2)$.

2.4.11 Transaction partition

The dataset is traversed once to assign items into different partitions. We can assume each partition can fit into the memory. Therefore, the space complexity is $O(\frac{n}{k})$.

2.4.12 Selecting and running the selected algorithm

The algorithm selection requires the calculation of the average transaction width of each transaction partition. The space complexity of this is $\mathrm{O}(k) = O(1)$, where k is the number of partitions.

If the modified Apriori algorithm is selected, the theoretical space complexity for each partition is $O\left(2^{\frac{1.03d}{k}}\right)$, where the coefficient 1.03 comes from the default 3% maximum imbalance of partitions caused by the MLkP algorithm. The total space complexity for all partitions is therefore $O\left(k * 2^{\frac{1.03d}{k}}\right) \rightarrow O(2^{\frac{1.03d}{k}})$, and the total space complexity of the SARL heuristic, when the modified Apriori algorithm is selected, is $O\left((3-1) * d^2 + \frac{n}{k} + 2^{\frac{1.03d}{k}}\right) \rightarrow O(d^2 + \frac{n}{k} + 2^{\frac{1.03d}{k}})$. Assume $\frac{n}{k} \gg d$, and $2^{\frac{1.03d}{k}} \gg \frac{n}{k}$, the space complexity can be simplified to $O(2^{\frac{1.03d}{k}})$. Compared with the space complexity of the Apriori algorithm, SARL uses only $O\left(\frac{2^{\frac{1.03d}{k}}}{2^d}\right) \rightarrow O\left(2^{\frac{1.03-k}{k}d}\right) \rightarrow o(\frac{1}{2^{\frac{k-1.03}{k}d}})$ space comparing to the Apriori algorithm. The exponential reduction of space usage comes from the smaller number of unique items in each transaction partition. If the modified Apriori is chosen to mine frequent itemsets from the transaction partitions, it only generates a smaller number of candidates for each transaction partition, since it does not consider items in other partitions.

2.5 Error Bound

The SARL heuristic sacrifices some precision to obtain the speed up. However, every frequent itemset found by the algorithm is correct, and the support associated with each frequent itemset is also correct. The heuristic may miss some trivial frequent itemsets, i.e., the itemsets with low support. During the IAG partition phase, the MLkP algorithm makes cuts on the IAG to minimize the sum of the weights of the edges that are cut off. This feature helps to prevent large weights from cut off, while some trivial, small-weight (support) edges may be lost.

In the most (extreme) case, when every transaction has all the items and *minsup* is set to 0, we can calculate the error bound. In this case, the IAG is a complete graph, and the fraction of the edges cut off by the MLkP algorithm is $\frac{n*\left(n-\frac{n}{k}\right)}{E} = \frac{(k-1)n}{k(n-1)}$. When *n* is very large, the fraction is approximately $\frac{k-1}{k}$. In this case, we can set *k* as low as 2 to still maintain 50% coverage for the frequent three or more itemsets. The calculation of frequent one and two itemsets is always accurate because they are calculated using the Apriori algorithm or the direct-generate algorithm.

The error rate should be significantly lower in more practical cases. However, it is difficult to estimate such an error rate considering it is affected by many factors such as the closeness of groups of items (i.e., does an item appear with only a small number of other items?), the choice of *minsup*, and the max length of the frequent itemsets. We can make a rough estimation by introducing a parameter P_{out}, the ratio of the edges cut off in the IAG. $P_{out} = \frac{E_{cut}}{E_{total}}$. This parameter is determined by the characteristics of a dataset, the *minsup* choice, and the number of partitions we choose. P_{out} is also a rough estimation of the error rate for the frequent two or more itemsets. Assume the ratio of the frequent two or more itemsets found is P_m,

$$P_m = \frac{\#\, frequent\ 2 +\ itemsets}{\#\, total\ frequent\ itemsets}$$

then the total error bound can be computed as

$$Error_{total} = P_m * P_{out}$$

2.6 Initial Selection of Number of Partitions, k

The selection of k determines the speed and accuracy of the SARL heuristic. A larger k usually means faster speed and lower accuracy, and vice versa. Depending on the size of the dataset and the application, k = 2, 3, or 4 are some balanced choices. In rare cases, the heuristic will increase the k value if any transaction partition cannot fit into the memory based on the current setting of k.

2.7 Benefits of Having Datasets Fit into the Memory

According to Section 4, Part 8, the transaction partitions are guaranteed to be small enough to fit into the memory. Therefore, any operations performed on these in-memory datasets should be faster than before. For example, the Apriori algorithm makes the number of passes on the dataset equal to the maximum length of frequent itemsets. Each of these passes requires reading the dataset from the disk. With our solution, the SARL heuristic makes at most two passes to the dataset. The first pass is to generate the frequent one and two itemsets, and in the second pass, the algorithm brings a fraction of the dataset into the memory. All further passes are made directly in the memory, resulting in speedup.

We do not analyze the communication cost between the main memory and the hard disk quantitatively in this dissertation. Due to the nature of our divide-and-conquer approach, we do not implement any additional swapping mechanism, so each partition is only brought into the memory once. Therefore, such cost should be no larger than the Apriori algorithm.

2.8 Theorems and Proofs

Theorem 1: Soundness - All frequent itemsets and association rules generated by the SARL algorithm are correct.

Proof:

Assume the SARL heuristic generates an incorrect frequent itemset. We can assume the correctness of the Apriori algorithm and the FP-growth algorithm. Therefore, there must be an error in transaction partitioning. There could be two possible types of error in transaction partitioning:

(Possibility 1) The support of some itemsets is higher or lower than it should be.

(Possibility 2) Some transactions include additional items or lose some items.

Assume the first possibility is true. We divide the dataset vertically (item-wise) during the transaction partitioning phase. Since every item in the original dataset D that belongs to P_i must be added to D_i, all unique items in a transaction partition must appear in the same number of transactions as the original dataset. Hence, the support of each itemset should be the same as the original dataset. This conflicts with the first possibility: the support of some itemsets is higher or lower than it should be.

Assume the second possibility is true. During the transaction partitioning phase, each transaction in the original dataset may be assigned to a transaction partition, or it may be split into different disjoint parts. Therefore, each transaction in a transaction partition must be a subset of the corresponding transaction in the original dataset, and this process cannot add any new items into any transactions. If some items are lost during the transaction partitioning phase, the results may have incorrect supports. However, we know that the union of the unique items in each transaction partition is equal to the unique items of the frequent two-itemsets, since the IAG partitioning cuts off some edges of IAG but not the nodes. According to the Apriori principle, a three-itemset can be frequent if and only if all its two-item subsets are frequent. This means that the unique items of three or more frequent itemsets must be a subset of the unique items of frequent two-itemsets. Hence, we have

$$\forall n \geq 3, I_n \subseteq I_2 = \bigcup_{j=1}^{m} P_j$$

where I_n is the unique items of frequent n-itemsets, P_j is the unique items of transaction partition *j*, and *m* is the number of transaction partitions. Therefore, all items needed by the frequent three (or higher) itemsets are present in the transaction partitions. Hence, we find a contradiction between our algorithm and the second possibility.

In summary, since both possibilities are proved to be false, the SARL heuristic is sound. ■

Theorem 2: Computing the frequent two itemsets is considered relatively trivial compared to computing the frequent three or more itemsets.

Proof:

If the computation of the frequent two itemsets takes more than half of the total computation time, we may say computing frequent two itemsets is not trivial.

To characterize the distribution of frequent itemsets is relatively difficult due to the challenges in modeling the data. We develop a mathematical model to simulate the characteristics of any dataset. The relationships of all the frequent itemsets can be depicted using an itemset lattice diagram shown below:

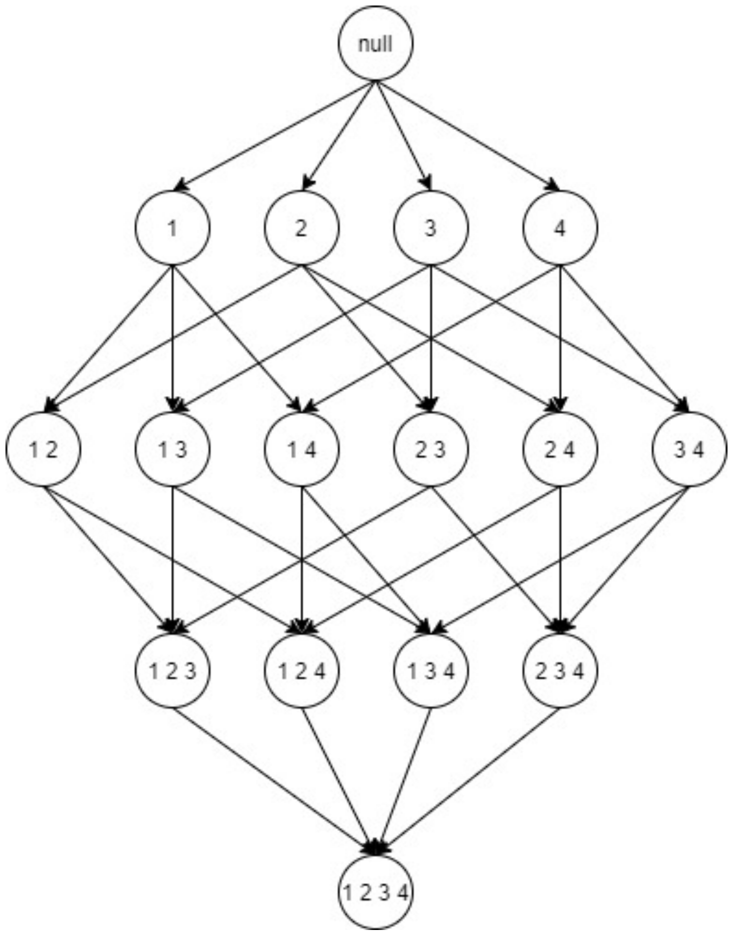


Figure 4 An Itemset Lattice

Figure 4 shows the case when every itemset has a support greater than *minsup*. However, in most cases, each layer will have some itemsets being removed due to either one of the two reasons: the anti-monotone property of the Apriori principle or the lack of support (i.e., support < *minsup*). To model the former, we apply the anti-monotone property to the itemset lattice. The anti-monotone property is as follows:

$$\forall X, Y \in J: (X \subset Y) \rightarrow f(Y) \leq f(X)$$

where if $J = 2^I$, I being a set of items, X is a subset of Y, then the measure f must be anti-monotone. Applying this property to the lattice, we can have the following explanation: if an itemset is infrequent, then all of its supersets must also be infrequent.

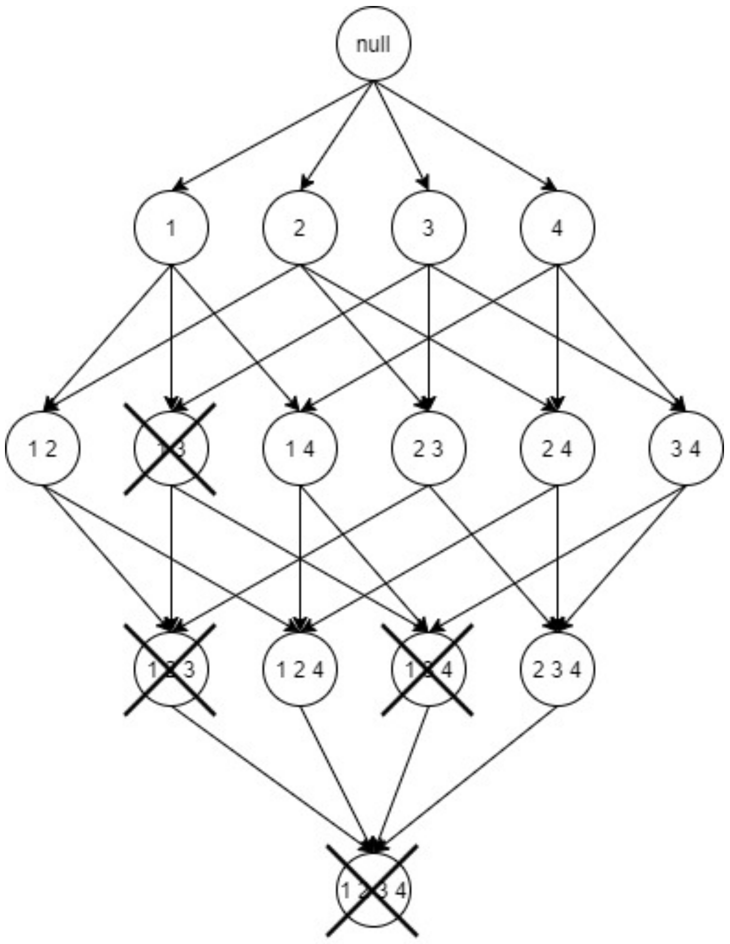


Figure 5 An Example of Pruning

For example, in Figure 5, if *{1, 3}* is infrequent, then *{1, 2, 3}*, *{1, 3, 4}*, and *{1, 2, 3, 4}* are all infrequent. To model this property, we can imagine that each infrequent itemset in the same layer causes some supersets in the next layer to be infrequent. The first infrequent itemset results in *n-k+1* infrequent itemsets in the next layer, where *n* is the number of unique items in the dataset, and k is the current layer number or the number of items in each itemset of the current layer. We know that each layer has C_k^n itemsets if none of them is infrequent. Then the next layer will have C_{k+1}^ntotal itemsets. Since *n-k+1* is the number of current infrequent itemsets in the next layer, $\frac{n-k+1}{C_{k+1}^n}$ is the current fraction of frequent itemsets over all the itemsets in the next layer. Therefore, $1-\frac{(n-k+1)}{C_{k+1}^n}$ is the probability of having a frequent itemset in the next layer if we randomly choose an itemset, and the second infrequent itemset should cause $\left(1-\frac{(n-k+1)}{C_{k+1}^n}\right)*(n-k+1)$ infrequent itemsets in the next layer. For the same reason, the third infrequent itemset in the current layer should cause

$\left(1-\frac{(n-k+1)+\left(1-\frac{(n-k+1)}{C_k^n}\right)*(n-k+1)}{C_{k+1}^n}\right)*(n-k+1)$ infrequent itemsets in the next layer. We can now estimate the number of infrequent itemsets I in the next layer using the number of infrequent itemsets in the current layer:

$$I_k = (n-k+1)+\left(1-\frac{(n-k+1)}{C_{k+1}^n}\right)*(n-k+1)+\left(1-\frac{(n-k+1)+\left(1-\frac{(n-k+1)}{C_k^n}\right)*(n-k+1)}{C_{k+1}^n}\right)*$$
$$(n-k+1)+\cdots$$

The remaining frequent itemsets in layer *k,* considering the above estimation of the influence of the Apriori principle, is $C_k^n - I_{k-1}$. Let us assume the probability p that an itemset to be frequent, assuming its parent is frequent. We can have the final estimated number of frequent itemsets for layer k:

$$f_k = (C_k^n - I_{k-1}) * p^k$$

For n = 200, p = 0.8, 0.6, 0.4, 0.2, 0.1, we can estimate the number of two, three, and more itemsets as shown in Table 11:

p	# two itemsets	# three itemsets	# four itemsets	2/(3+4)
0.8	12736	228346	972761	0.010603552
0.6	7164	41585	714273	0.009477971
0.4	3184	6761	30960	0.084409215
0.2	796	589	1898	0.320064335
0.1	199	67	118	1.075675676

Table 11 Estimation of the Number of Itemsets

For n = 2000, p = 0.8, 0.6, 0.4, 0.2, 0.1, we can estimate the number of two, three, and more itemsets as shown in Table 12:

p	# two itemsets	# three itemsets	2/3
0.8	1279360	231482728	0.005527
0.6	719640	42159431	0.017069
0.4	319840	6855578	0.046654
0.2	79960	597871	0.133741
0.1	19990	68301	0.292675

Table 12 Estimation of the Number of Itemsets for Larger Datasets

The above model with examples shows that the number of two itemsets is, on average, less than only 10% of the number of three or more itemsets. This means that only less than 10% of all computation power is consumed by the two itemsets. Thus, our algorithm speeds up the costly part, the part that mines three or more itemsets. ■

Theorem 3: Consider a value of minsup such that the fraction of frequent one itemset over the total number of unique items, d, denoted by f, is less than (1 - the maximum imbalance rate), where the maximum imbalance rate is usually set to 3% based on the MLkP algorithm. If the partition by MLkP is k-way, then each partition contains less than d/k unique items, where d is the total unique items in the original dataset. As a consequence, the complexity of each partition can be reduced.

Proof:

Assume that given f < 100% - 3% or f < 97%, and a transaction partition has $d_i \geq d/k$ unique items. According to our algorithm, since $d_i \geq d/k$, a partition in the IAG must have more than or equal to *d/k* nodes. As we assumed earlier, the maximum imbalance rate for the MLkP algorithm is set to 3%, then the number of nodes *n* in the IAG can be calculated as $\frac{d}{k} * 0.97 * k \leq\ n \leq \frac{d}{k} * 1.03 * k\ or\ 0.97d \leq n \leq 1.03d$. Since *n* cannot be more than the total number of unique items, $0.97d \leq n \leq d$. However, we know *f < 97%* or *f *d< 0.97d*, and $n \leq f * d$ since some frequent one itemsets may not appear in any frequent two itemsets, so $n \leq f * d < 0.97d$ and $n < 0.97d$. This contradicts $0.97d \leq n \leq d$. Therefore, the assumption $d_i \geq d/k$ is false, and the reverse, $d_i < \frac{d}{k}$, must be true. ■

3. Scalable Precise Algorithm, SARL-Precise

3.1 An Example

The following example aims to show the motivation for the development of the SARL-Precise algorithm.

Suppose the same dataset discussed earlier, as shown in Table 13, is given and *minsup* is set to 0.1 (or 10%, or $7 * 0.1 \approx 1$ occurrences), and *minconf* is set to 0.7 (or 70%):

TID	Items
T000	1, 2
T001	1, 2, 3
T002	4, 5
T003	1, 4, 5
T004	2, 3
T005	1, 2, 3
T006	1, 4, 5

Table 13 Example Dataset

First, we use the Apriori algorithm to find frequent two-itemsets. As an intermediate step, the Apriori algorithm finds frequent one itemset, as shown in Table 14:

Frequent Itemsets	Support
{1}	5
{2}	4
{3}	3
{4}	3
{5}	3

Table 14 Frequent One Itemsets

The frequent two-itemsets are found afterward, as shown in Table 15:

Frequent Itemsets	Support
{1, 2}	3
{1, 3}	2
{1, 4}	2
{1, 5}	2
{2, 3}	3

{4, 5}	2

Table 15 Frequent Two Itemsets

Next, we transform the above frequent two-itemsets into an item association graph, as shown in Figure 6.

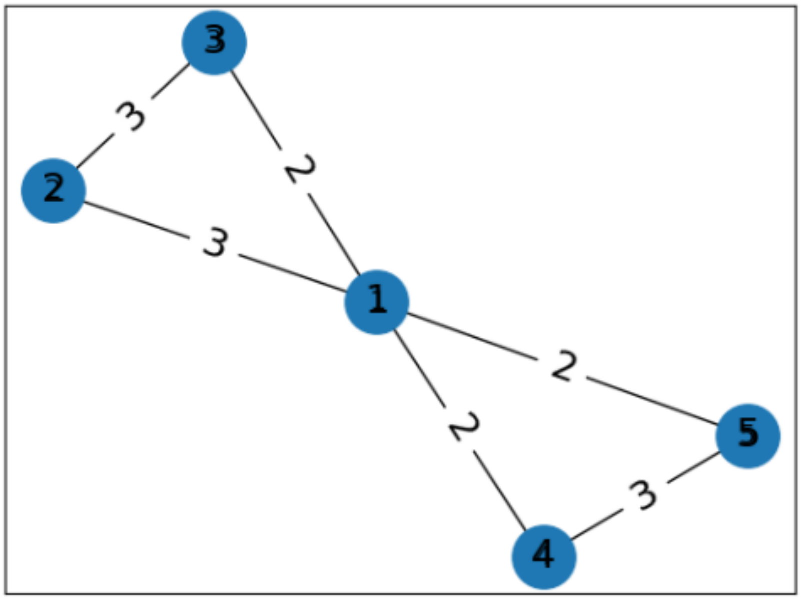


Figure 6 Item Association Graph Example

To construct the graph, we take itemset {1, 2} with support 3, create node 1 and node 2 that correspond to the two items in the itemset. The edge between node 1 and node 2 has weight 3, representing the support of the itemset. The process is repeated for every frequent two-itemset found in the previous step.

Next, we use the multilevel k-way partitioning algorithm (MLkP) to partition the IAG. In this case, the number of nodes is small, so we only bisect the graph by setting *k* = 2.

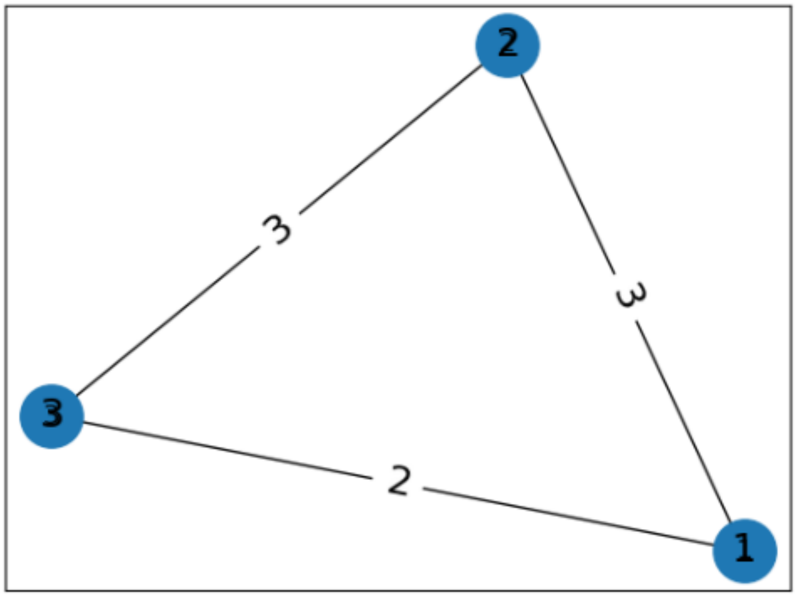


Figure 7 Item Association Graph Partition 1

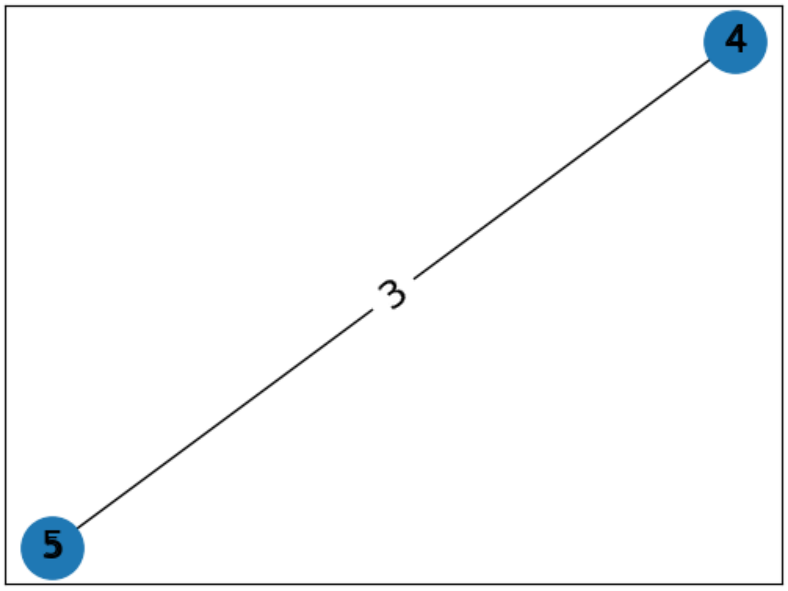


Figure 8 Item Association Graph Partition 2

The MLkP algorithm divides the IAG into two equal or almost equal sets in linear time while the sum of the weights of edges cut off is the minimum. The results are shown in Figure 7 and Figure 8.

Next, we partition the dataset according to the partitions of the IAG, as shown in Tables 16 and 17. Each transaction partition has all the items from the corresponding IAG partition. However, since the algorithm has already found all frequent one and two itemsets, a transaction will not be added to a transaction partition if the transaction has fewer than three items. For example, T000: {1, 2} is not added to the transaction partition 1, since it only has two items. Some items in the original dataset may not be included in any of the transaction partitions, because the infrequent one/two-itemsets are dropped in the IAG. This simplifies the subsequent computations. In this example, however, all the items are kept in the IAG because the IAG is a relatively dense graph. Figures 16 and 17 show the transaction partitions.

TID	Items
T001	1, 2, 3
T005	1, 2, 3

Table 16 Transaction Partition 1

TID	Items
None	None

Table 17 Transaction Partition 2

Note that there is no transaction in partition 2. This is because partition 2 has only two unique items, it is not possible to mine any frequent three itemsets from this partition. Therefore, partition 2 is discarded.

In this step, TID T003 and T006 are marked as *divided* transactions, because they have elements that are divided between two different IAG partitions. The first partition is not enough to discover the potential frequent itemset {1, 4, 5}.

To avoid the loss of the divided transactions, in the scalable precise algorithm, a *bridge* transaction partition is constructed based on the divided transactions from the last step, as shown in Table 18.

TID	Items
T003	1, 4, 5
T006	1, 4, 5

Table 18 Bridge Transaction Partition

According to the algorithm, the above bridge transaction partition is converted into an IAG and then bisected by the MLkP algorithm. The results are two partitions, [1, 4] and [5]. At the same time, similar to the previous step, both transactions are marked as divided and will be added to the next bridge partition. The following (Table 19 and 20) are two small bridge partitions generated from the first bridge. They are both empty because none of them contains any transactions of three items or more.

TID	Items
None	None

Table 19 Partition 1 of The Bridge

TID	Items
None	None

Table 20 Partition 2 of The Bridge

From the above two partitions, a second bridge(shown in Table 21) is derived using the generate-bridge function.

TID	Items
T003	1, 4, 5
T006	1, 4, 5

Table 21 The Second Bridge

The second bridge has the same size as the first bridge, so we can stop here and discard the second bridge.

The next step is to pick the best algorithm and use it to find the frequent k-itemsets with k > 2. For this example, we choose the modified Apriori algorithm because it is faster for mining small datasets, and it avoids the process of finding the frequent one and two itemsets again. The results from partition 1 are shown in Table 22:

Frequent Itemsets	Support
{1, 2, 3}	2

Table 22 Frequent Itemsets from Transaction Partition 1

Since the modified Apriori algorithm starts at finding three-itemsets, there are no additional frequent itemsets in the second partition, as shown in Table 23.

Frequent Itemsets	Support
None	N/A

Table 23 Frequent Itemsets from Transaction Partition 2

Table 24 shows the frequent itemset found in the bridge transaction partition.

Frequent Itemsets	Support
{1, 4, 5}	2

Table 24 Frequent Itemsets from Bridge Transaction Partition

We can compute the final result by taking the union of the frequent itemsets from Tables 14, 15, 22, 23, and 24. The support of each frequent itemset is the maximum support among the results of all partitions.

Frequent Itemsets	Support
{1}	5
{2}	4
{3}	3
{4}	3
{5}	3
{1, 2}	3
{1, 3}	2
{1, 4}	2
{1, 5}	2
{2, 3}	3
{4, 5}	3
{1, 2, 3}	2
{1, 4, 5}	2

Table 25 Frequent Itemset Final Results

After running the Apriori-ap-genrules algorithm, the rules shown in Table 26 can be discovered.

Rules	Confidence
{2} → {1}	0.75
{3} → {2}	1
{5} → {1}	1
{2} → {3}	0.75
{5} → {4}	1
{4} → {5}	1
{1, 3} → {2}	1
{1, 4} → {5}	1
{1, 5} → {4}	1

Table 26 Generated Association Rules

Comparing the result with the result of the Apriori algorithm, the algorithm is sound and complete. Every frequent itemset and rule is correct, and it finds all frequent itemsets as well as all the rules.

3.2 Concise Description of the SARL-Precise Algorithm

Step 1: Find itemsets with size one and two using the Apriori algorithm or direct generation algorithm.

Step 2: Construct the item association graph(IAG) from the result of step 1.

Step 3: Partition the IAG using multilevel k-way partitioning algorithm(MLkP)

Step 4: Partition the database according to the result of step 3, mark those transactions required to be assigned to the bridge partition

Step 5: Construct the bridge partition using the result from step 4. Recursively reduce the bridge partition into smaller partitions.

Step 5: Choose an algorithm to mine frequent itemset on each database partition based on the characteristic of different algorithms.

Step 6: Summarize the result. For each frequent itemset found in any partition, choose the highest support among all partitions.

3.3 Another Example

This example shows how the SARL algorithm works on a slightly more complex dataset, shown in Table 27, which requires more recursions for bridge generation.

TID	Items
T000	1, 2, 4
T001	2, 4, 5
T002	2, 3, 4
T003	1
T004	1, 2, 3
T005	2, 3, 5

T006	1, 3, 4
T007	2, 3, 5
T008	2, 3

Table 27 Another Example Dataset

Let us run the SARL algorithm. Firstly, the modified Apriori algorithm is used to find frequent one and two itemsets. The results are shown in Tables 28 and 29.

Frequent Itemsets	Support
{1}	4
{2}	7
{3}	6
{4}	4
{5}	3

Table 28 Frequent One Itemsets

Frequent Itemsets	Support
{1, 2}	2
{1, 3}	2
{1, 4}	2
{2, 3}	5
{2, 4}	3
{2, 5}	3
{3, 4}	2
{3, 5}	2

Table 29 Frequent Two Itemsets

Similar to the previous examples, the IAG is constructed according to the frequent two itemsets found above, as shown in Figure 9.

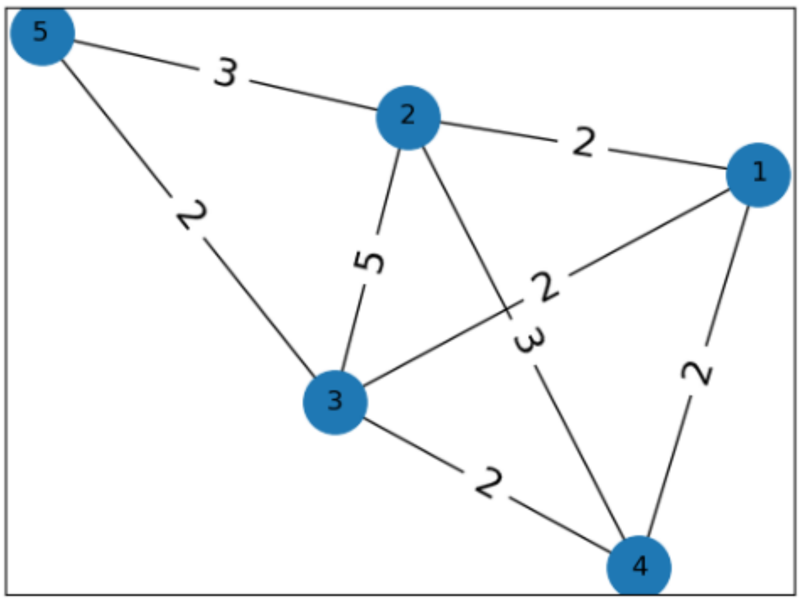


Figure 9 IAG for Example 2

The MLkP algorithm divides the IAG into two partitions, as shown in Figure 10 and Figure 11:

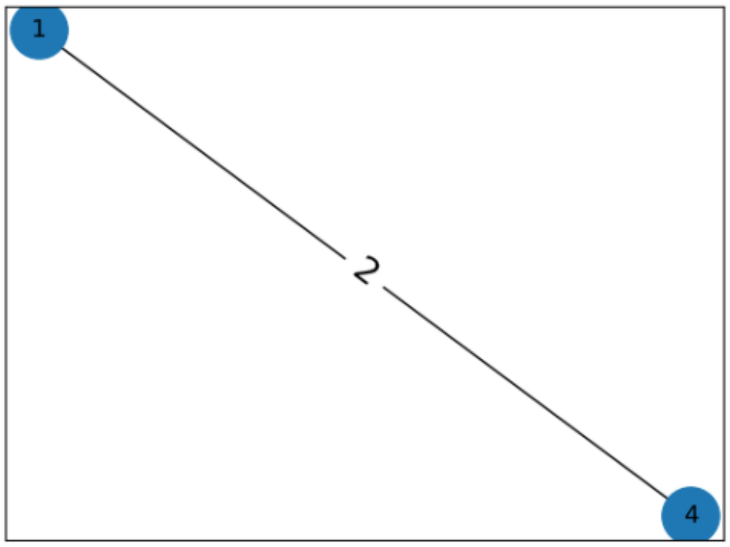


Figure 10 IAG Partition 1

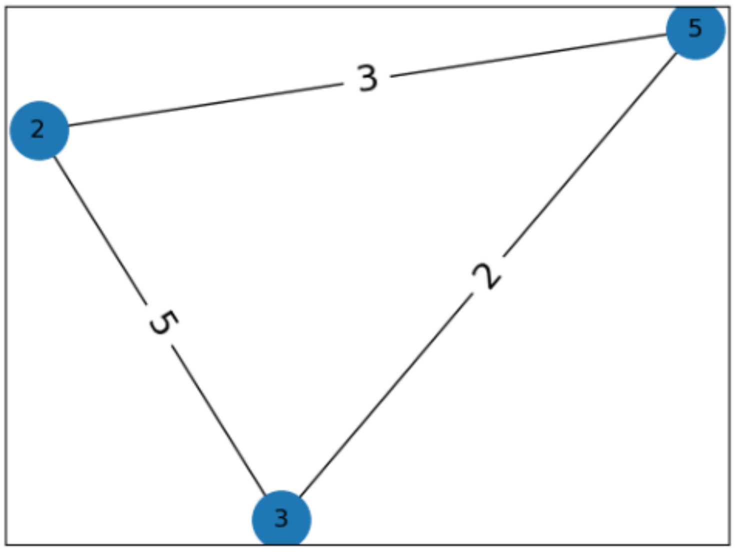


Figure 11 IAG Partition 2

Next, the SARL algorithm divides the original dataset into two transaction partitions, as shown in Table 30 and Table 31, according to the two IAG partitions above with the same method discussed in the previous example:

TID	Items
None	None

Table 30 Transaction Partition 1

TID	Items
T005	2, 3, 5
T007	2, 3, 5

Table 31 Transaction Partition 2

In this process, we mark each divided transaction and put its TID into a list if that transaction has three or more items: [T000, T001, T002, T004, T006]

Next, the first bridge partition is constructed with the transactions in the list above, as shown in Table 32.

TID	Items
T000	1, 2, 4
T001	2, 4, 5
T002	2, 3, 4
T004	1, 2, 3
T006	1, 3, 4

Table 32 The First Bridge Partition

Now, the first bridge partition is treated as the original dataset to generate an IAG, partitioned by the MLkP algorithm ([1,3] and [2, 4, 5]), and generates two small bridge partitions, as shown in Table 33 and Table 34:

TID	Items
None	None

Table 33 Small Bridge Partition 1

TID	Items

T001	2, 4, 5

Table 34 Small Bridge Partition 2

A second bridge partition can be built during this process, as shown in Table 35:

TID	Items
T000	1, 2, 4
T002	2, 3, 4
T004	1, 2, 3
T006	1, 3, 4

Table 35 The Second Bridge Partition

Comparing to the previous bridge, the second bridge has one fewer transaction and one fewer unique item. Therefore, we have a reduced bridge partition. Since the decomposition is lossless, the first bridge can be safely discarded.

If we repeat this process one more time, the IAG partitions are [1,2] and [3, 4]. Then we can obtain two empty bridge partitions and the bridge partition shown in Table 36.

TID	Items
T000	1, 2, 4
T002	2, 3, 4
T004	1, 2, 3
T006	1, 3, 4

Table 36 The Third Bridge Partition

This third bridge partition is the same partition as the second one. Therefore, we can discard this one. In summary, we have losslessly transformed the original dataset into the partitions shown in Tables 37, 38, and 39.

TID	Items
T005	2, 3, 5
T007	2, 3, 5

Table 37 The Transaction Partition 1

TID	Items
T001	2, 4, 5

Table 38 The Transaction Partition 2

TID	Items
T000	1, 2, 4
T002	2, 3, 4
T004	1, 2, 3
T006	1, 3, 4

Table 39 The Second Bridge Partition

After running the modified Apriori algorithm on each of the partitions. We can find the frequent itemset shown in Table 40:

Frequent Itemsets	Support
{2, 3, 5}	2

Table 40 Frequent Itemsets Found

The example shown above has only one frequent three itemset. However, according to Theorem 2, given a low *minsup* and a large dataset, there should be more frequent three or more itemsets than frequent two itemsets.

We can find the union of all frequent itemsets to compute the final result, as shown in Table 41. The support of each frequent itemset is the maximum of those across the results from all partitions for the same itemset.

Frequent Itemsets	Support
{1}	4
{2}	7
{3}	6
{4}	4
{5}	3
{1, 2}	2
{1, 3}	2
{1, 4}	2

{2, 3}	5
{2, 4}	3
{2, 5}	3
{3, 4}	2
{3, 5}	2
{2, 3, 5}	2

Table 41 Final Frequent Itemsets

By running the Apriori-ap-genrules algorithm, the rules shown in Table 42 can be found.

Rules	Confidence
{2} → {3}	0.71
{3} → {2}	0.83
{5} → {2}	1
{4} → {2}	0.75
{3, 5} → {2}	1

Table 42 Association Rules

Again, both frequent itemsets and association rules found by the SARL algorithm are sound and complete.

3.4 Formalized Algorithm

Following is the pseudo-code for the SARL-Precise Algorithm.

3.4.1 Pseudo Code

```
SARL_Precise:
bridges = []
results, two_itemset = mod1-Apriori(dataset)
graph = build_IAG(two_itemset)
partitions = METIS.partition(k, graph)
files, div_index = partition-dataset(dataset, partitions)
```

```
bridges = generate-bridge(files, div_index, k, dataset, 0, infinity)
files += bridges
for file in files:
    results += mod2-Apriori(file)  #when files are small
    results += FP-Growth(file) #when files are large
rules = Apriori-gen(results)


mod1-Apriori(dataset):
C1 = {}
for transaction in dataset:
    for item in transaction:
        if item not in C1:
            add item to C1
item.counter = 1
        else:
            item.counter += 1
L1 = {}
for candidate in C1:
    if candidate.counter >= minsup:
        add candidate to L1
C2 = {}
for itemset1 in L1:
    for itemset2 in L1:
        if itemset1 != itemset2:
            add itemset1 U itemset2 to C2
for transaction in dateset:
    for candidate in C2:
        if candidate.issubset(transaction):
            candidate.counter += 1
L2 = {}
for candidate in C2:
    if candidate.counter >= minsup:
        add candidate to L2
return L1, L2


build_IAG(itemsets):
for itemset in itemsets:
    graph.add_node(itemset[0])
graph.add_node(itemset[0])
graph.add_edge(itemset[0], itemset[1], weight += 1)
return graph


partition-dataset(dataset, partitions):
div_index = []
for transaction in dataset:
    for partition in partitions:
```

```
                intersect = parition intersect transaction
        if len(intersect) < len(transaction):
                  div_index.append(TID_of_transaction)
           elif len(intersect) > 2:
add intersect to dataset_partition_i
return dataset_partition_names, div_index


generate-bridge(div_index, k, dataset, i, last_len):
        count = 0
for TID in div_index:
                bridge_i. add (dataset[TID])
                count+=1
        if count == last_len:
                return bridge_i
        else:
                dataset = bridge_i
                results, two_itemsets = mod1-Apriori(dataset)
                graph = build_IAG(two_itemset)
                partitions = METIS.partition(k, graph)
                temp_files, div_index = partition-dataset(dataset, partitions)
                return generate-bridge( div_index, k, dataset, i+1, count), temp_files
```

3.4.2 Recursively Generating Bridge Transaction Partitions

The bridge dataset and their partitions are necessary to achieve a precise calculation of frequent three or more itemsets. The main purpose of deriving them is to reconsider all potential errors, so the SARL algorithm does not miss any frequent itemsets.

The algorithm is greedy and defined recursively. In each recursion, it first finds all divided transactions that were labeled during the transaction partition step, and they are added to the bridge partition. The divided transactions were labeled so that there is no need to make a whole pass of the dataset again to check if a transaction is divided. Then, it calculates the number of transactions in the bridge partition and compares it to the previous, undivided bridge partition. If the current bridge partition has the same number of transactions as the previous one, that means the current reduction does not reduce the size of the bridge partition any further. The algorithm discards the current bridge partitions. On the other hand, if the total number of transactions in the new bridge partition is less than that of the previous bridge partition, then the algorithm divides the bridge partition again.

Some additional transaction partitions are generated through this process. However, calculating frequent itemsets in these smaller partitions are relatively simple since they are small enough to fit into the memory and contain fewer unique items than their parent bridge partition. Reducing the size of the bridge partition could result in an exponential reduction in the complexity of finding frequent itemsets, as we analyzed earlier.

3.5 Analysis

The time and the space complexities of the SARL-Precise algorithm, when the modified Apriori algorithm is chosen, are the same as those of the Apriori algorithm. That is, $\boldsymbol{O}(\mathbf{2}^{\boldsymbol{d}})$ for both time and space complexities. From our previous analysis, the SARL-Heuristic algorithm has a complexity of $\boldsymbol{O}(\mathbf{2}^{\frac{1.03d}{k}})$ for both time and space. In the most extreme case, every transaction of the original dataset is divided, then the bridge partition will be the same as the original dataset, and the total time complexity will be $\boldsymbol{O}\left(\mathbf{2}^{\frac{1.03d}{k}}\right) + \boldsymbol{O}(\mathbf{2}^{\boldsymbol{d}}) = \boldsymbol{O}(\mathbf{2}^{\boldsymbol{d}})$. Similarly, the total space complexity is also $\boldsymbol{O}(\mathbf{2}^{\boldsymbol{d}})$. However, the SARL-Precise algorithm runs faster than the Apriori algorithm in most cases. This is because the MLkP algorithm finds the sub-optimized solution to cut the minimum number of transactions. Thus, the size of the bridge is usually much smaller than the original dataset.

3.5.1 Theorems and Proofs

Theorem 5: Each decomposition of a bridge partition or the original dataset is lossless.

Proof:

After dividing the original dataset into two transaction partitions, there are two possibilities for each transaction of the original dataset. Each transaction T must be either divided and assigned to different partitions or assigned to a single partition as a whole. For the latter case, all existing subsets of T, including duplications of T, must also be assigned into the same partition. We can say that partition contains the complete information for any itemsets that are subsets of T. Therefore, both the Apriori algorithm and the FP-growth algorithm will get the same support for frequent itemsets that are subsets of T. If a frequent itemset F is a subset of any transaction T, T being a transaction completely(i.e. all items in the transaction are assigned) assigned to P_i , this will be the latter case.

As for the former case, the bridge partition B includes all divided transactions. If a frequent itemset F is not a subset of any transaction T, T being a transaction completely assigned to P_i , then all transactions K that are supersets of F must be assigned to B. The former case can be described as,

$$F \subset \begin{cases} T \mid T \in P_i \ \wedge (\forall t \subset T) \in P_i \\ K \mid K \in B \ \wedge (\forall k \subset K) \in B \end{cases}$$

Therefore, for each frequent itemset F, all transactions that contain any supersets of F are assigned to only one partition. So all support numbers are guaranteed to be the same as the ones before decomposition.

For the same reasons, all subsequent decompositions are performed on the bridge partitions and are lossless as well.

4. SARL, An Application to Microarray Datasets

4.1 Data Preprocessing

4.1.2 Dataset Reduction

It may not be very useful for a large dataset with hundreds of thousands of genes to find rules that cover all the genes. Since we may be only interested in over-expressed and under-expressed genes and all diseases, normally expressed genes could be eliminated from our dataset. We can also adjust the threshold of over-expressed and under-expressed genes to classify fewer genes as over/under-expressed ones.

A common approach is the following method that converts the gene expression levels into log-scale values [39].

First, we arbitrarily pick a reference assay and calculate the relative expression levels based on the reference assay. Assuming the absolute gene expression levels of the reference assay is $E_{r1}, E_{r2} \ldots E_{rn}$, we can calculate the relative gene expression levels for another assay A as: $R_{A1}, R_{A2} \ldots R_{An} = \frac{E_{r1}}{E_{A1}}, \frac{E_{r2}}{E_{A2}} \ldots \frac{E_{rn}}{E_{An}}$ where $E_{A1}, E_{A2} \ldots E_{An}$ are absolute gene expression levels for assay A. We can use the above method to calculate the relative gene expression levels for all other assays.

Next, the relative gene expression levels are used to find the log-scale gene expression levels. For each assay A, the log-scale gene expression levels are calculated as: $L_{A1}, L_{A2} \ldots L_{An} = \log_2 R_{A1}, \log_2 R_{A2} \ldots \log_2 R_{An}$

In the end, a user-defined threshold h is used to filter out some normally expressed expression levels. A lower h value means more gene expression levels are kept, and the computation time is longer. This step can dramatically reduce the size of the dataset while keeping valuable information.

4.1.3 Converting into Transactional Datasets

Microarray datasets are matrices of data. Each row of a matrix represents a gene, while each column represents an assay. However, to perform association rule learning, we need to convert microarray datasets into transactional datasets. Each row is an assay in a transactional dataset, and each "transaction" has a different number of genes.

Our algorithm transposes the matrix that we obtain from the earlier steps. Next, each log-scale gene expression level is converted into a ternary item [23]. If the level exceeds the positive threshold, an item $+G$ replaces the corresponding expression level where G is the gene number. Likewise, if the log-scale expression level is less than the negative threshold, it will be replaced by $-G$.

For example, if we have an assay that has genes G1, G2, and G3 with log-scale expression levels {-100, 10, 300}, respectively. Assuming the thresholds are -50 and +50, we convert the expression levels into {-G1, +G3}. G2 is not included in the above transaction because its expression level is not significant.

4.1.4 Extracting Disease Information

We introduce disease information to the transactional dataset so that our association rule learning algorithm can derive gene-disease association rules. The prior approaches do not address disease information. To find association rules that involve genes and diseases, we need to convert the disease information associated with each assay into an item.

First, the disease information is extracted from the sample information. A disease, in this case, can be a specific disease or "normal." If the disease information is provided, our algorithm will copy the disease name as an item name to the corresponding assay(transaction). Therefore, for each transaction, there are one or more gene items and a disease item.

For example, an assay is labeled as "Tumor" in the original dataset and calculated in the above steps to have items {-G1, +G3}. The algorithm will add "Tumor" to the transaction to have {-G1, +G3, "Tumor"}.

4.1.5 Calculating Gene Importance

An important aspect of association rule ranking is evaluating the importance of each gene. The following is our approach for calculating gene importance, of which the importance of a gene can be viewed as the average degree of over/under expression in the dataset. We also want to consider $+G$ and $-G$ individually since they are considered different items in the transformed dataset. We define the gene importance for gene g, E_g, as below:

$$E_g = \sqrt{\frac{\sum_{j=1}^{m}\left(K_j - K_g\right)^2}{m}}$$

In the above, K_j is the average gene expression level of gene j, K_g is the average gene expression level for gene g. E_g is the RMS deviation of gene g, related to all other genes. If the deviation is high, the expression level of a gene is outstanding, and we can say it is important.

For example, if the average gene expression level of all the genes is 20, and we calculate that gene G1 has an average expression level of 100, while G2 has an average expression level of 2. Then E_g for G1 and G2 are 80 and 18, respectively. Therefore, G1 should be ranked above G2.

4.2 An Example

	Assay 1	Assay 2	Assay 3	Assay 4	Assay 5	Assay 6	Assay 7	Assay 8
Gene 1	0.11	0.03	1.51	0.34	10.21	0.01	0.28	1.33
Gene 2	5.23	5.78	1.37	1.44	7.65	21.35	1.98	1.28
Gene 3	1.32	4.89	1.05	1.37	8.45	17.56	1.79	1.79
Gene 4	1.56	0.97	0.05	0.12	1.02	1.34	0.19	1.12
Gene 5	6.33	1.28	0.15	0.53	1.02	2.15	0.34	0.98

Table 43 Microarray Dataset

Suppose the microarray dataset in Table 43 is given, and minsup is set to 0.2 (or 20%, or $8 * 0.2 \approx 2$ occurrences), and *minconf* is set to 0.7 (or 70%). We select assay 8 as the reference assay then calculate the relative expression levels. The results are shown in Table 44.

	Assay 1	Assay 2	Assay 3	Assay 4	Assay 5	Assay 6	Assay 7
Gene 1	0.08270676	0.02255639	1.13533653	0.25563903	7.67666664	0.0075188	0.21052624
Gene 2	4.08600225	4.51563851	1.07031095	1.1250039	5.97657933	16.6796834	1.54687862
Gene 3	0.73742999	2.73183434	0.58659228	0.76536297	4.72066465	9.81004667	1
Gene 4	1.39285975	0.86607148	0.04464285	0.10714289	0.91071402	1.19642503	0.16964291
Gene 5	6.45917541	1.14286104	0.13265307	0.46938772	0.90816353	1.91836867	0.3061225

Table 44 Relative Expression Levels

Next, we calculate the log-scale gene expression levels by taking log base 2: $\log(x, 2)$ where x is the relative expression level. The results are shown in Table 45.

	Assay 1	Assay 2	Assay 3	Assay 4	Assay 5	Assay 6	Assay 7
Gene 1	-3.595851	-5.47032	0.18312	-1.96782	2.94048	-7.055282	-2.247928
Gene 2	2.03069	2.17493	0.09803	0.16993	2.57932	4.06002	0.62936
Gene 3	-0.439422	1.44987	-0.76957	-0.385784	2.23899	3.29426	0
Gene 4	0.47805	-0.207442	-4.485427	-3.222392	-0.13493	0.25873	-2.559427
Gene 5	2.69135	0.19265	-2.91427	-1.091148	-0.138976	0.93988	-1.707819

Table 45 Log-scale Expression Levels

Then we normalize the expression levels by applying a threshold to the log-scale expression levels. Here, we choose the threshold to be 1, meaning all log-scale expression levels that are above 1 or below -1 are set to 1 and -1, respectively. Levels between -1 and 1 are set to 0. The results are shown in Table 46.

	Assay 1	Assay 2	Assay 3	Assay 4	Assay 5	Assay 6	Assay 7
Gene 1	-1	-1	0	-1	1	-1	-1
Gene 2	1	1	0	0	1	1	0
Gene 3	0	1	0	0	1	1	0
Gene 4	0	0	-1	-1	0	0	-1
Gene 5	1	0	-1	-1	0	0	-1

Table 46 Normalized Expression Levels

Next, we transpose the matrix to prepare it for the transactional dataset. Each row is now an assay and each column is a gene. The results are shown in Table 47.

	Gene 1	Gene 2	Gene 3	Gene 4	Gene 5
Assay 1	-1	1	0	0	1
Assay 2	-1	1	1	0	0
Assay 3	0	0	0	-1	-1
Assay 4	-1	0	0	-1	-1
Assay 5	1	1	1	0	0
Assay 6	-1	1	1	0	0
Assay 7	-1	0	0	-1	-1

Table 47 Transposed Matrix

Finally, we convert the transposed matrix to a transactional dataset, shown in Table 48, each expression level that equals -1 or 1 is transformed into an item. In Table 48, the items of each transaction include the genes that are over-expressed (denoted by a + symbol) and genes that are under-expressed (denoted by a - symbol). For example, a transaction with TID T000 is an assay that contains three significantly (over or under) expressed genes, gene 1 (under-expressed), gene 2 (over-expressed), and gene 5 (over-expressed).

TID	Items
T000	-1, +2, +5
T001	-1, +2, +3
T002	-4, -5
T003	-1, -4, -5
T004	+1, +2, +3
T005	-1, +2, +3
T006	-1, -4, -5

Table 48 Transactional Dataset

Now, we use the Apriori algorithm to find the frequent two itemsets. As an intermediate step, the Apriori algorithm finds the frequent one-itemset first (shown in Table 49):

Frequent Itemsets	Support
{-1}	5
{+2}	4
{+3}	3
{-4}	3

{-5}	3

Table 49 Frequent One Itemsets

The frequent two-itemsets are found afterward (shown in Table 50):

Frequent Itemsets	Support
{-1, +2}	3
{-1, +3}	2
{-1, -4}	2
{-1, -5}	2
{+2, +3}	3
{-4, -5}	2

Table 50 Frequent Two Itemsets

Next, we transform the above frequent two-itemsets into an item association graph (IAG), shown in Figure 12:

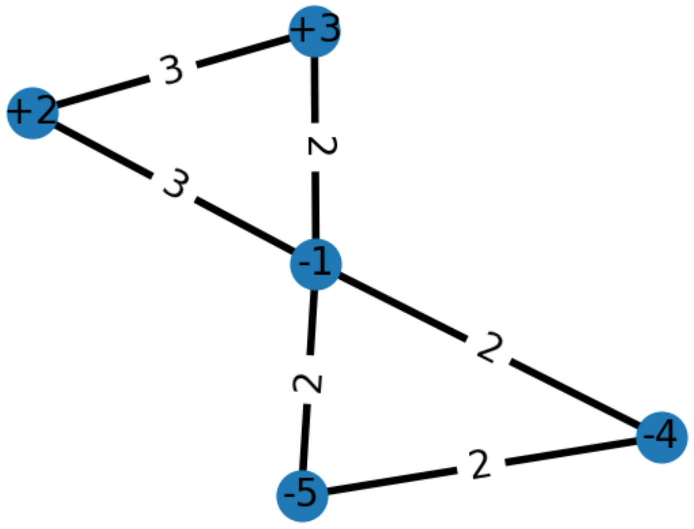


Figure 12 An Item Association Graph

To construct the graph, we first take the itemset {-1, +2} with support 3. For this, we create node -1 and node +2 corresponding to the two items in the itemset. The edge between node -1 and node +2 has weight 3, representing the support of the itemset. The process is repeated for every frequent two-itemset found in the previous step.

Subsequently, we use the multilevel k-way partitioning algorithm (MLkP) to partition the IAG. In this case, the number of nodes is small, so we only bisect the graph by setting k = 2. The result is shown in Figures 13 and 14.

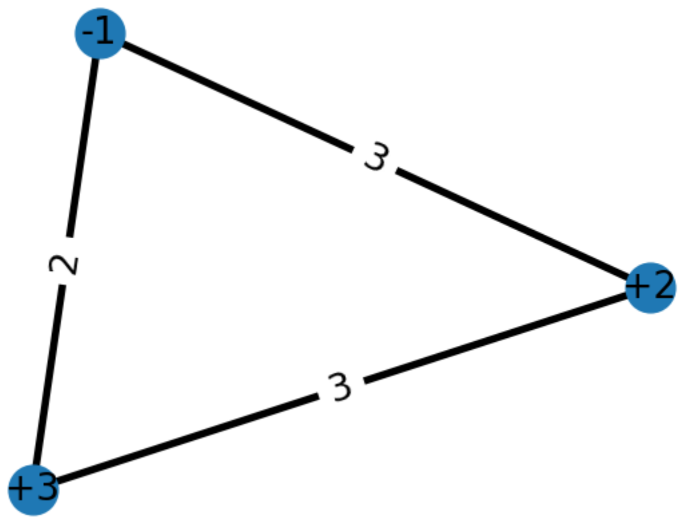


Figure 13 Item Association Graph Partition 1

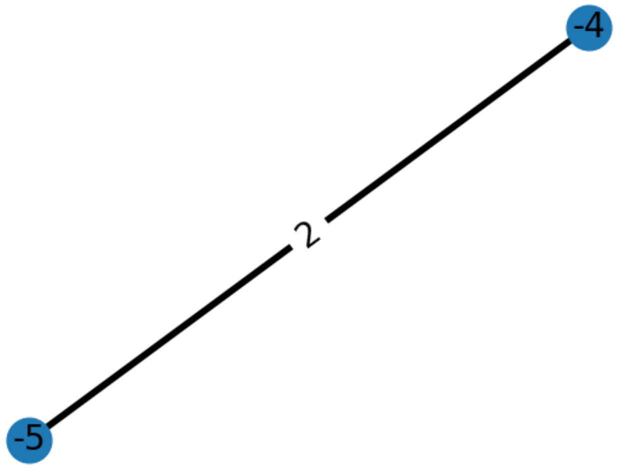


Figure 14 Item Association Graph Partition 2

The MLkP algorithm divides the IAG into two equal or almost equal sets in linear time while the sum of the weights of edges that are cut off is the minimum.

Next, we partition the dataset according to the partitions of the IAG, as shown in Tables 51 and 52. Each transaction partition has all the items from the corresponding IAG partition. However, since the algorithm has already found all the frequent one and two itemsets, a transaction is not added to a transaction partition if the transaction has less than three items. For example, T000: {-1, +2} is not added to the transaction partition 1, since it only has two items. Some items in the original dataset may not appear in any of the transaction partitions, because the infrequent one/two-itemsets are dropped in the IAG. This simplifies the subsequent computations. In this example, however, all the items are kept in the IAG because the IAG is a relatively dense graph. Tables 51 and 52 show the transaction partitions:

TID	Items

T001	-1,+ 2, +3
T005	-1, +2, +3

Table 51 Transaction Partition 1

TID	Items
None	None

Table 52 Transaction Partition 2

The next step is to pick the best algorithm and use it to find the frequent k-itemsets with k > 2. For this example, we choose the modified Apriori algorithm because it is faster for mining small datasets as it avoids the process of finding the one and two-itemsets again. The results from partition 1 are shown in Table 53:

Frequent Itemsets	Support
{-1, +2, +3}	2

Table 53 Frequent Itemsets from Transaction Partition 1

Since the modified Apriori algorithm starts with three-itemsets, there are no additional frequent itemsets in the first partition. Table 54 shows the results found in transaction partition 2:

Frequent Itemsets	Support
None	N/A

Table 54 Frequent Itemsets from Transaction Partition 2

The final results (shown in Table 55) of frequent itemsets are simply the union of Tables 49, 50 and 53:

Frequent Itemsets	Support
{-1}	5
{+2}	4
{+3}	3
{-4}	3
{-5}	3
{-1, +2}	3
{-1, +3}	2
{-1, -4}	2
{-1, -5}	2
{+2, +3}	3
{-4, -5}	2
{-1, +2, +3}	2

Table 55 Frequent Itemset Final Results

After running the Apriori-ap-genrules algorithm, the association rules can be found in Table 56.

Rules	Confidence
{+2} → {-1}	0.75
{+3} → {+2}	1
{-5} → {-1}	1

{+2} → {+3}	0.75
{-5} → {-4}	1
{-4} → {-5}	1
{-1, +3} → {+2}	1

Table 56 Association Rules Generated

All the frequent itemsets generated by the SARL heuristic are sound, meaning each frequent itemset generated indeed is correct, and the support number is accurate. However, it is possible that some frequent itemsets cannot be found by the SARL heuristic, as will be discussed shortly. In this example, the SARL heuristic loses one frequent itemset {-1, -4, -5} and two related rules generated from {-1, -4, -5}.

4.3 The SARL(Scalable Association Rule Learning) Heuristic

We introduced the SARL heuristic earlier in this dissertation. SARL is a highly scalable heuristic algorithm for association rule learning problems on horizontal transactional datasets. In this section, a modified version of SARL serves as the core of our algorithm. A summary of the SARL heuristic is shown below. A more detailed and formal description, including the pseudo-code, can be found in the earlier sections of this dissertation.

The Apriori algorithm or the direct counting and generation algorithm is used to generate frequent one and two itemsets, depending on the size of the dataset. Apriori is faster on very large datasets, where the direct counting and generation algorithm is faster on small and medium-sized datasets. SARL then builds the item association graph (IAG) based on the frequent two itemsets. Each frequent two itemset is converted into an edge on the IAG, and each item in the itemset is converted into a node. Then, the MLkP algorithm is used to partition the IAG into k subgraphs. The dataset is partitioned based on the subgraphs. Each partition of the dataset should contain all the items (nodes) of a subgraph of the IAG across all the transactions of the datasets. During this process, some transactions may end up undivided, and all the possible frequent itemsets related to these transactions will be preserved in later stages. Next, the Apriori algorithm or the FP-Growth algorithm is selected based on an analysis of the dataset to ensure the most efficient execution. Finally, SARL calls the selected algorithm on each dataset partition to complete the computation. If the Apriori algorithm is selected, SARL will call the modified Apriori that starts from the frequent three itemsets computation to avoid any redundant work.

The SARL heuristic divides the dataset into k partitions. The size of each partition should be smaller than $\frac{1}{k} \times$ size of the dataset because the dataset is partitioned according to IAG, and the number of items (nodes) in the IAG should be smaller than the number of unique items in the dataset. A more detailed explanation can be found in the Transaction Partitioning section of the Appendix section. This indicates that each dataset partition can always fit into the memory. All later steps of the SARL heuristic significantly benefit from processing the dataset in the memory rather than on the disk.

4.4 Ranking of Association Rules

Considering the nature of microarray datasets, the number of unique items (genes) is usually large. This leads to a tremendous number of association rules. Therefore, it is necessary to rank the association rules by their importance so that the results can be easily used. The goal of this study aims to help scientists explore and validate new association rules more efficiently.

To achieve the goal, we introduce the following measurement of the importance of rule *r: x*→*y*:

$$I_r = L_r \times (\frac{\sum E_{gr}}{n} + B_r)$$

where $L_r = \frac{conf(r)}{\sup(r)}$, and $E_g = \sqrt{\frac{\sum_{j=1}^{m}(K_j - K_g)^2}{m}}$

In the above, I_r is the importance of the rule, L_r is the lift of the rule r, n is the number of unique genes included in rule r, E_{gr} is the RMS deviation of the expression level of gene g that is included in rule r, K_j is the gene expression level of gene j, where j represents all other genes; K_g is the gene expression level for gene g, and B_r is the bias applied to this rule. The bias should be positive if a disease is in the rule.

The intuition here is to emphasize three factors that are related to the importance of an association rule. The first is the lift of a rule [32]. A higher lift indicates the rule has a higher response compared to the other rules. If the lift value is large, then the antecedent and the consequent of the rule are more dependent on each other, and this further indicates a high significance of the rule. The second factor is the average significance of each gene included in the rule. If all or most of the genes are significant, then the rule is likely to be more important. When we convert the microarray dataset into a transactional dataset, the absolute gene expression levels are converted into relative expression levels, and some information related to the absolute levels is missing. Here, we reconsider the influence of the absolute gene expression levels and calculate the average significance of a gene-based on it. E_g, the deviation of the average absolute gene expression level, is calculated by taking the difference of the average absolute gene expression levels of all genes and the average absolute gene expression level of gene g. The significance of a rule contributed by its genes is then calculated by taking the average of each E_g that is included in rule r.

For example, assuming the rule $G1 \rightarrow G2$ is an association rule found by the SARL heuristic. Genes 1, 2, and 3 have expression levels of 10, 8, and 5, respectively. The rule has confidence of 0.7 and support of 10. Then we can calculate $L_r = \frac{conf(r)}{\sup(y)} = 0.07$, $E_1 = \sqrt{\frac{(K_2-K_1)^2+(K_3-K_1)^2}{2}} = 3.8$, $E_2 = \sqrt{\frac{(K_3-K_2)^2+(K_1-K_2)^2}{2}} = 2.5$. Since the rule does not involve a disease, bias is set to 0. Hence, $I_r = L_r \times \left(\frac{\sum E_{gr}}{n} + B_r\right) = 0.07 \times \left(\frac{3.8+2.5}{2} + 0\right) = 0.22$.

We incorporate the three most important measurements in the ranking of the rules. The lift measurement generally addresses the ranking of the significance of each rule, the average gene significance traces back to the microarray dataset and considers the importance of each gene, and the bias B_r highlights the rules that involve disease information.

Chapter V: EXPERIMENTS AND RESULTS

We design and conduct experiments on both small and large datasets to demonstrate the scalability of our algorithm. The experiments are performed on a computer with the following settings:

OS: Ubuntu 64-bit running on a virtual machine

CPU: Intel Core i7-4720HQ

Memory: 8192MB allocated to the virtual machine

Disk: 5400RPM, 64MB Cache, 6.0Gb/s, SSHD, 8GB flash memory

Programming Language: Python 3.7

The datasets [19] we use include Bible [17], T10I4D100K [19], and T40I10D100K [19]. The details of each dataset will be discussed later.

For each of these datasets, we test the SARL heuristic with various settings for the FP-Growth and the Apriori algorithms on different values of *minsup*. The various settings of the SARL heuristic are as follows:

2ap: k = 2, Apriori-based

2fp: k = 2, FP-Growth-based

4ap: k = 4, Apriori-based

4fp: k = 4, FP-Growth-based

The Bible Dataset

The Bible dataset has the following metrics:

Number of unique items: 13905

Number of transactions: 36396

Average transaction width: 21.6

File size: 5.4 MB

This is a small to medium-sized dataset. The experiments are done repeatedly for *minsup* of 50%, 40%, 30%, 20%, and 10%. The time limit for each experiment is set to 800 seconds for each of the experiments. The results are shown in Table 57:

	fp	sarl 2apF	sarl 2fpF	sarl 4apF	sarl 4fpF	ap
50	timeout	8.69149	9.099829	8.689143	13.36314	17.57703
40	timeout	8.169094	8.981468	9.30843	14.59894	17.60924
30	timeout	9.39442	10.64038	11.12533	17.64162	21.45831
20	timeout	10.68418	13.63021	10.22187	10.61261	27.32679
10	timeout	20.87115	30.99064	27.77191	30.65479	51.8529

Table 57 Running Times of Different Algorithms on the Bible Dataset

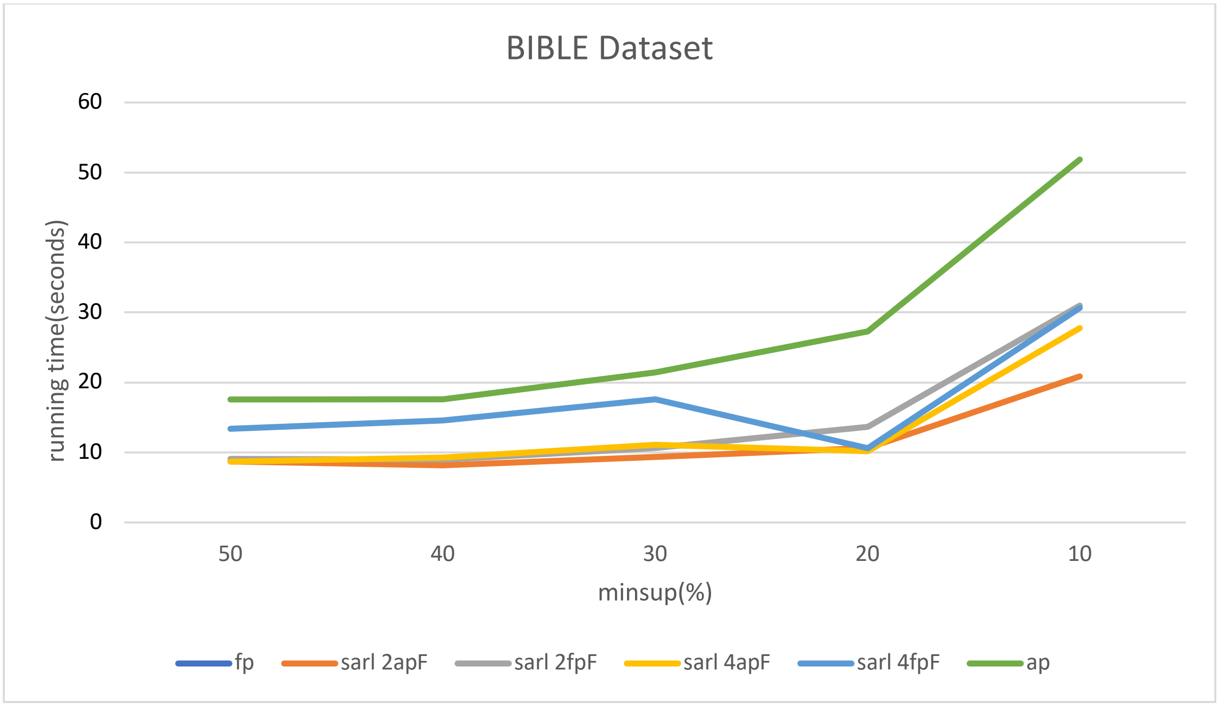
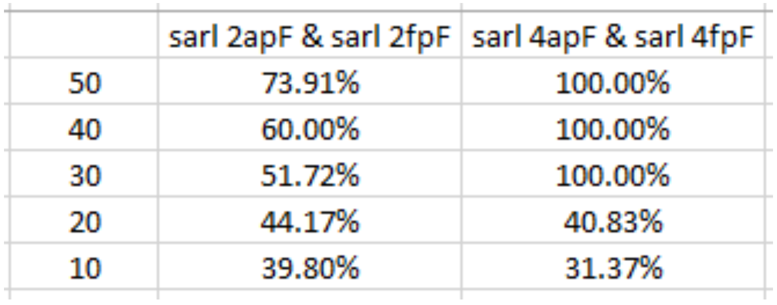


Figure 15 Running Times of Different Algorithms on the Bible Dataset

According to Figure 15, the two-partition, Apriori-based SARL heuristic scales the best for this dataset regardless of the *minsup* value. It is 2 to 2.5 times faster than the Apriori algorithm. The FP-Growth algorithm reaches the 800-second time limit for all test cases. It is possible that the number of unique items in this dataset is large; therefore the FP-tree cannot fit into the memory. As a result, the FP-growth algorithm does not perform well here. All other three settings of the SARL heuristic outperform the Apriori algorithm. Comparing to the FP-growth algorithm and the Apriori algorithm, the SARL heuristic is more scalable with all values set for *minsup*.



	sarl 2apF & sarl 2fpF	sarl 4apF & sarl 4fpF
50	73.91%	100.00%
40	60.00%	100.00%
30	51.72%	100.00%
20	44.17%	40.83%
10	39.80%	31.37%

Table 58 Accuracy of the SARL Algorithm on the Bible Dataset

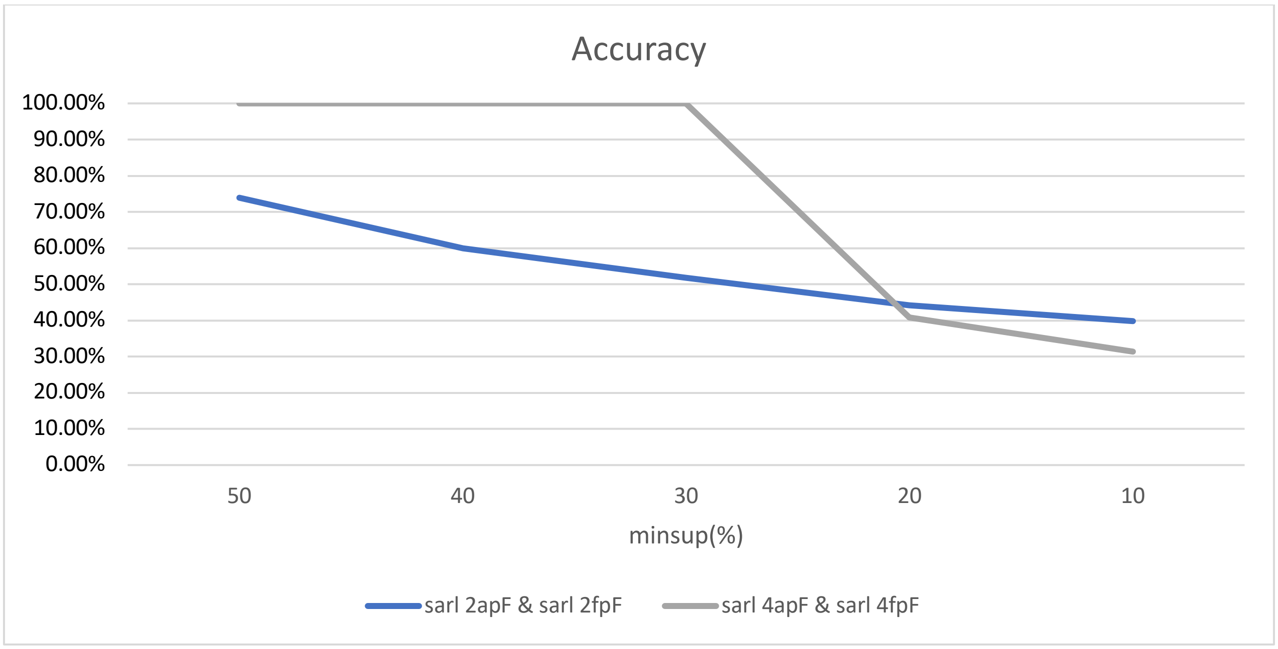


Figure 16 Accuracy of Different Configurations of SARL Heuristic on the Bible Dataset.

As we proved earlier, all the frequent itemsets found by the SARL heuristic are accurate, with the correct support. This is important because we need the accurate support to calculate the confidence of the rules. The SARL heuristic may miss some frequent itemsets with a lower support. Here, we calculate the accuracy = $\frac{\text{number of frequent itemsets found by SARL}}{number\ of\ frequent\ itemsets\ found\ by\ Apriori}$. The accuracy of the SARL heuristic drops on the Bible dataset when the value of *minsup* is low. From Table 58 and Figure 16, both settings of the four-partition SARL heuristic achieve 100% accuracy from the *minsup* range of 50% to 30%. This is because the MLkP algorithm is able to find a perfect or almost perfect cut on the IAG so that there are no inter-partition frequent itemsets for this range. When 100% accuracy is achieved, the SARL heuristic discovers not just the one and two frequent itemsets, but also the three or higher frequent itemsets. As for the two-partition SARL heuristic settings, the accuracy starts at 73.91% at 50% *minsup* and drops to 39.8% at 10% *minsup*.

The T10I4D100K Dataset

The second dataset we have tested is T10I4D100K. It has the following statistics:

Number of unique items: 870

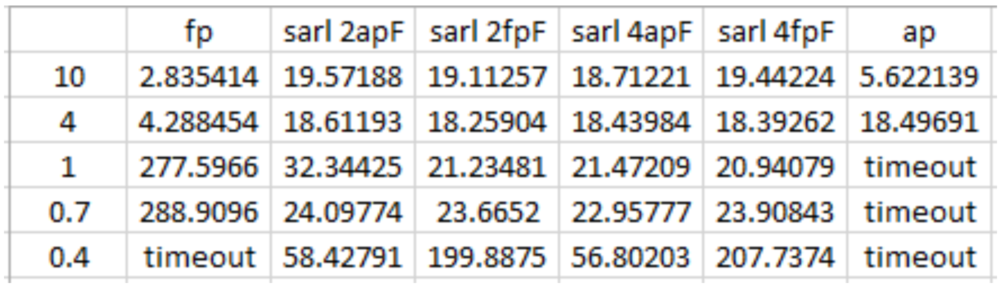
Average size of transactions: 10

Number of transactions: 100000

File size: 4MB

The algorithms are tested on T10I4D100K for *minsup* of 10%, 4%, 1%, 0.7%, and 0.4%. This dataset has a medium size (for this environment), so the time limit is set to 300 seconds for each of the experiments.

Table 59 and Figure 17 shows the results for T10I4D100K:



	fp	sarl 2apF	sarl 2fpF	sarl 4apF	sarl 4fpF	ap
10	2.835414	19.57188	19.11257	18.71221	19.44224	5.622139
4	4.288454	18.61193	18.25904	18.43984	18.39262	18.49691
1	277.5966	32.34425	21.23481	21.47209	20.94079	timeout
0.7	288.9096	24.09774	23.6652	22.95777	23.90843	timeout
0.4	timeout	58.42791	199.8875	56.80203	207.7374	timeout

Table 59 T10I4D100K Running Times

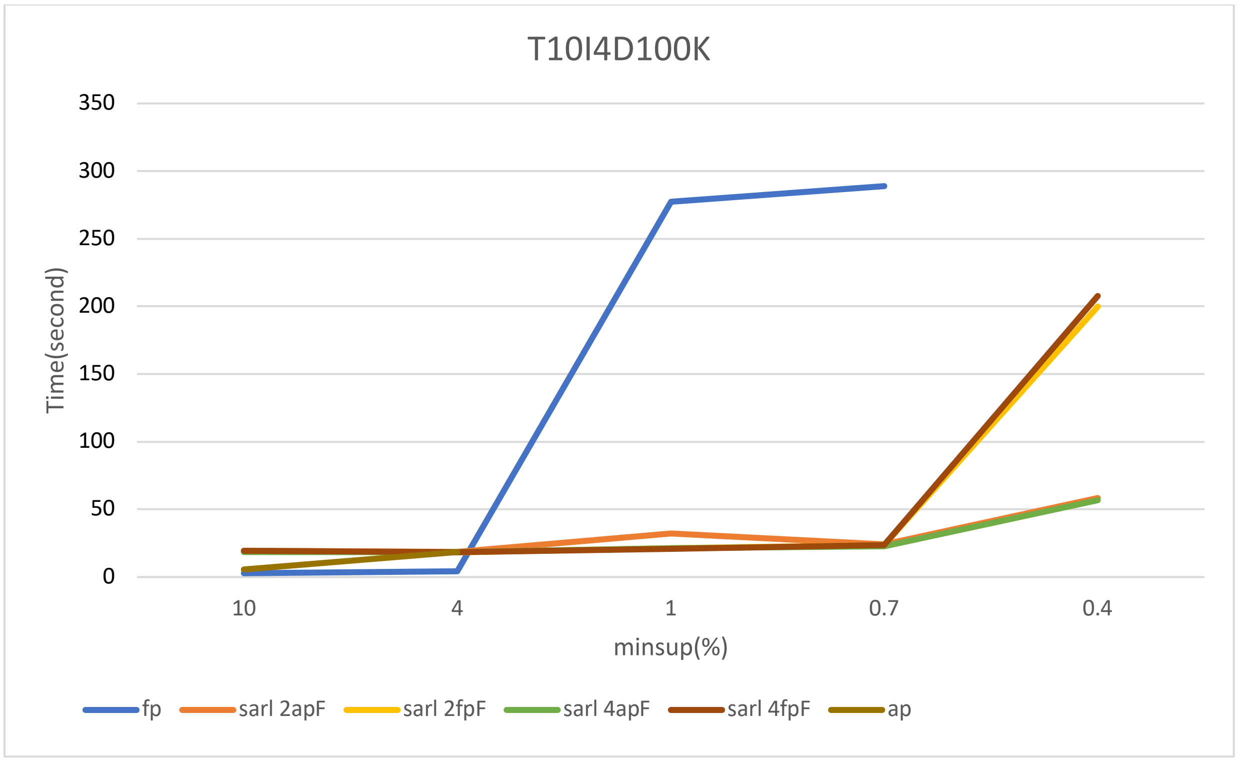
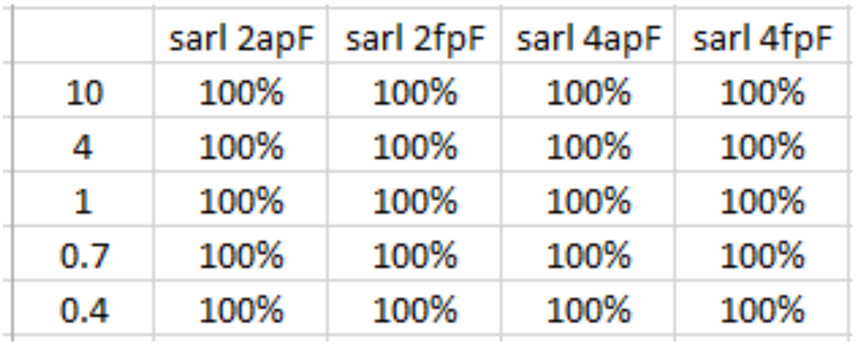


Figure 17 Running Times of Different Algorithms on the T10I4D100K Dataset

From Figure 17, the Apriori algorithm has an average performance for the initial *minsup* of 10% and 4%. However, it quickly reaches the maximum running time after that and unable to finish the task in time for all subsequent settings of *minsup*. The FP-Growth algorithm has a better performance. It is the fastest for a higher value of *minsup* of 10% and 4%, but it jumps to almost 300 seconds for 1% and 0.7%, before timeout at 0.4%. All settings of the SARL heuristic outperform the Apriori and the FP-Growth algorithm for middle and low settings of *minsup*. It is 8.6 to 13.8 times faster than the FP-Growth algorithm on minsup = 1% and 0.7%. The SARL heuristic is slightly slower at a high *minsup* of 10%, and they are tied with the Apriori but slightly slower than FP-Growth at a *minsup* of 4%.



	sarl 2apF	sarl 2fpF	sarl 4apF	sarl 4fpF
10	100%	100%	100%	100%
4	100%	100%	100%	100%
1	100%	100%	100%	100%
0.7	100%	100%	100%	100%
0.4	100%	100%	100%	100%

Table 60 Accuracy of SARL Heuristic on T10I4D100K

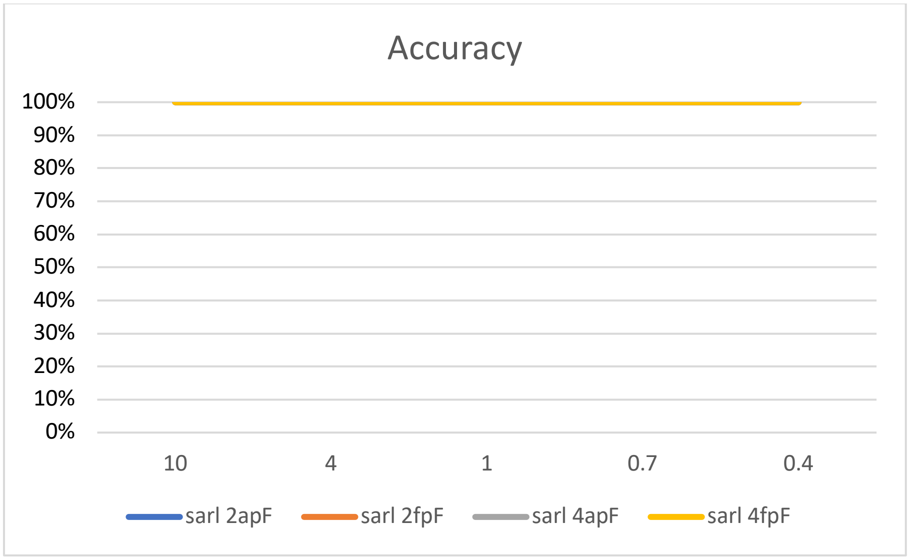


Figure 18 Accuracy of SARL Heuristic on T10I4D100K

The accuracy of the SARL heuristic is high on the T10I4D100K dataset. As shown in Table 60 and Figure 18, all four settings of the SARL heuristic achieve 100% accuracy for the values of *minsup* from 10% to 0.4%. This is because, for a high *minsup*, the number of frequent three or more itemsets for this dataset is small comparing to frequent two itemsets, and the mining of the one and two frequent itemsets is accurate. For low *minsup* values, the MLkP algorithm successfully finds a perfect or almost perfect cut on the IAG, so the results are accurate.

The T40I10D100K Dataset

The dataset T40I10D100K has the following statistics:

Number of unique items: 942

Average size of transactions: 40

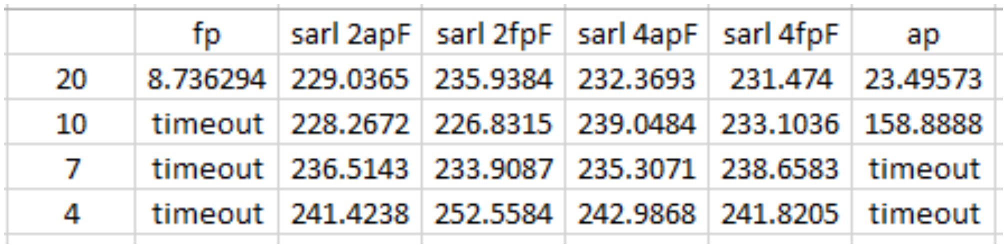
Average size of the maximal potentially large itemsets:10

Number of transactions:100000

File size: about 15 MB

This relatively large-size dataset was tested on *minsup* values of 20%, 10%, 7%, and 4%. The maximum running time was set to 300 seconds each for the experiments.

Table 61 shows the results of the experiments:



	fp	sarl 2apF	sarl 2fpF	sarl 4apF	sarl 4fpF	ap
20	8.736294	229.0365	235.9384	232.3693	231.474	23.49573
10	timeout	228.2672	226.8315	239.0484	233.1036	158.8888
7	timeout	236.5143	233.9087	235.3071	238.6583	timeout
4	timeout	241.4238	252.5584	242.9868	241.8205	timeout

Table 61 Running Times of Different Algorithms on the T40I10D100K Dataset

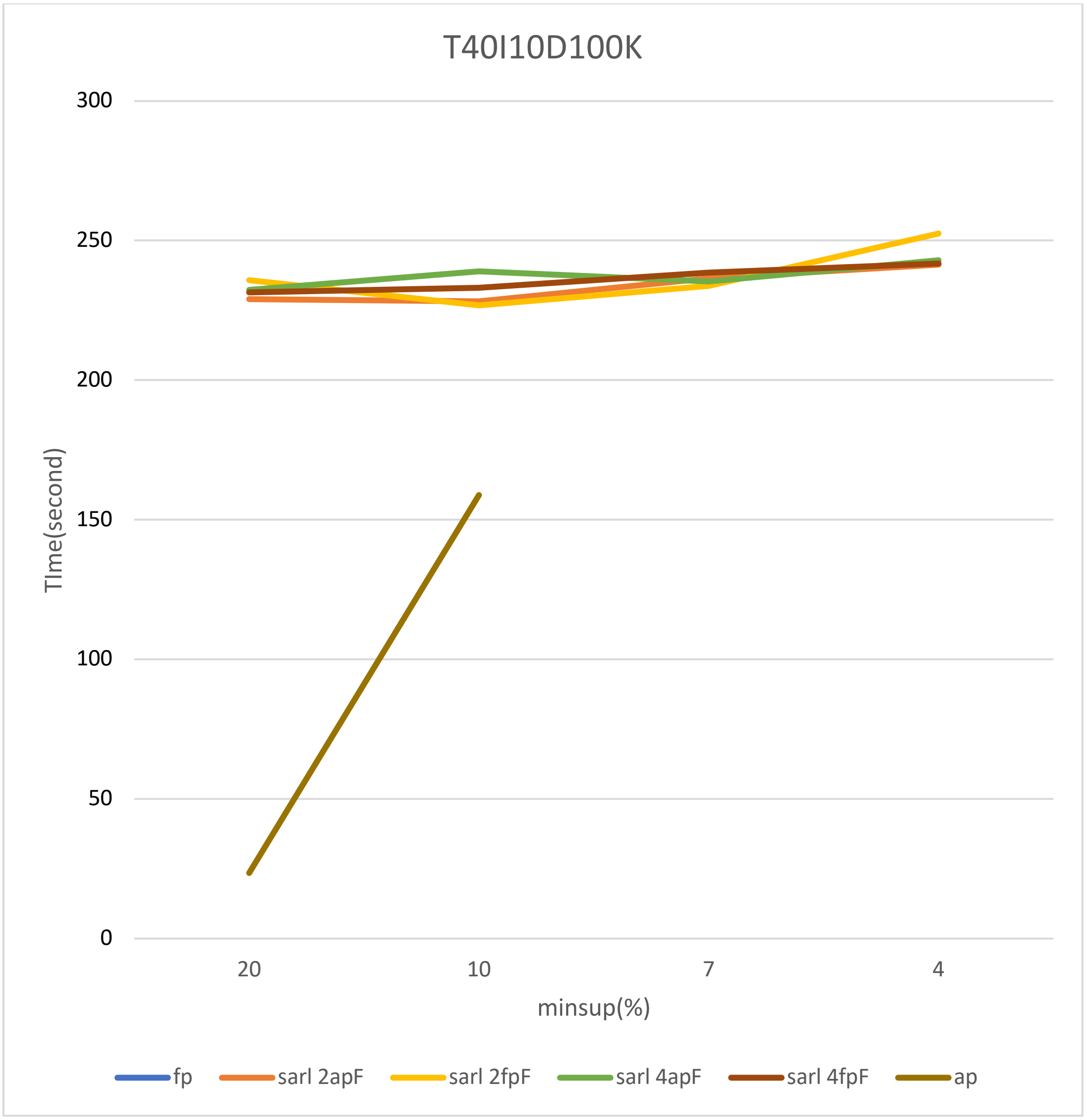
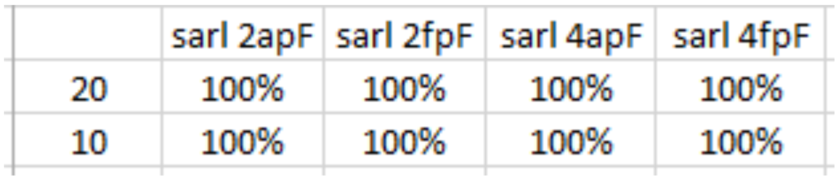


Figure 19 Running Times of Different Algorithms on the T40I10D100K Dataset

The results of the experiments (shown in Table 61 and Figure 19) show an obvious distinction between the scalability of different algorithms. All settings of the SARL heuristic demonstrate high scalability. Almost all settings of the SARL heuristic have stable running time throughout the entire range of *minsup*. Surprisingly, the Apriori algorithm performs better than the FP-Growth algorithm with a *minsup* between 20% and 7%. However, it is still unable to terminate within the time limit for *minsup* = 4%. Lastly, the FP-Growth algorithm does not scale very well on this dataset. It fails to terminate within the given time for both 7% and 4% of *minsup*.



	sarl 2apF	sarl 2fpF	sarl 4apF	sarl 4fpF
20	100%	100%	100%	100%
10	100%	100%	100%	100%

Table 62 Accuracy of SARL Heuristic on T40I10D100K Dataset

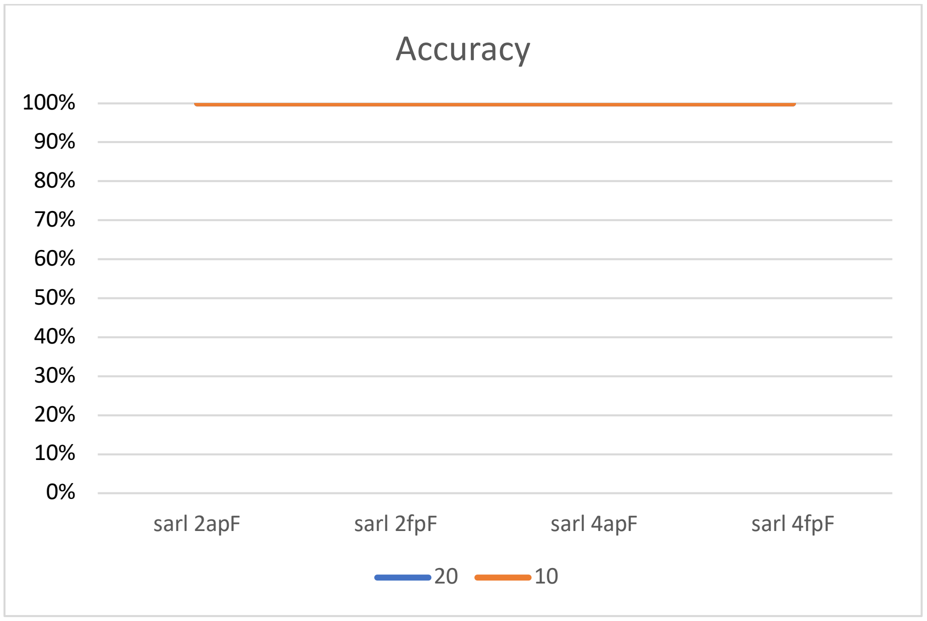


Figure 20 Accuracy of SARL Heuristic on T40I10D100K Dataset

The accuracy of the SARL heuristic on the T40I10D100K dataset is the same as the T10I4D100K dataset. Table 62 and Figure 20 show that the SARL heuristic has 100% accuracy based on similar reasons as we explained above in the analysis of the T10I4D100K experiment results.

Another set of experiments was done on three datasets. In this set of experiments, we compare both SARL-heuristic and SARL-precise with other algorithms. The datasets [19] we use include Mushroom [13], T10I4D100K [19], and T40I10D100K [19]. The details of each dataset will be covered later.

For each of these datasets, we tested the SARL algorithm with various settings for the FP-Growth and the Apriori algorithms on different values of *minsup*. The various settings of the SARL algorithm are as follows:

2apF: *k* = 2, Apriori-based, heuristic mode.

2apT: *k* = 2, Apriori-based, precise mode.

2fpF: *k* = 2, FP-Growth-based, heuristic mode.

2fpT: *k* = 2, FP-Growth-based, precise mode.

4apF: *k* = 4, Apriori-based, heuristic mode.

4apT: *k* = 4, Apriori-based, precise mode.

4fpF: *k* = 4, FP-Growth-based, heuristic mode.

4fpT: *k* = 4, FP-Growth-based, precise mode.

The Mushroom Dataset

The mushroom dataset has the following metrics:

Number of unique items: 119

Number of transactions: 8124

Average transaction width: 23

File size: 558 KB

This is a small dataset considering its size. However, the complexity of mining frequent itemsets is non-trivial due to its large average transaction width. The experiments was done repeatedly for *minsup* of 20%, 10%, 7%, 4%, 1%, 0.7%, 0.4%, and 0.1%. The time limit for each experiment was set to 400 seconds for each of the 80 experiments. The results are shown in Figure 21:

	fp	sarl 2apF	sarl 2apT	sarl 2fpF	sarl 2fpT	sarl 4apF	sarl 4apT	sarl 4fpF	sarl 4fpT	ap
20	79.2578	62.78268		54.88132	126.1449	29.18472		13.04025	94.26079	
10	379.6436			56.5663	305.3277	129.0632		19.01914	269.448	
7	354.9317			113.5315		165.1814		17.03973	368.4053	
4				161.2162				44.68621		
1				362.2856				39.26179		
0.7				165.8576				51.73652		
0.4				386.2269				46.14297		
0.1								100.1229		

Table 63 Experiments from the Mushroom Dataset

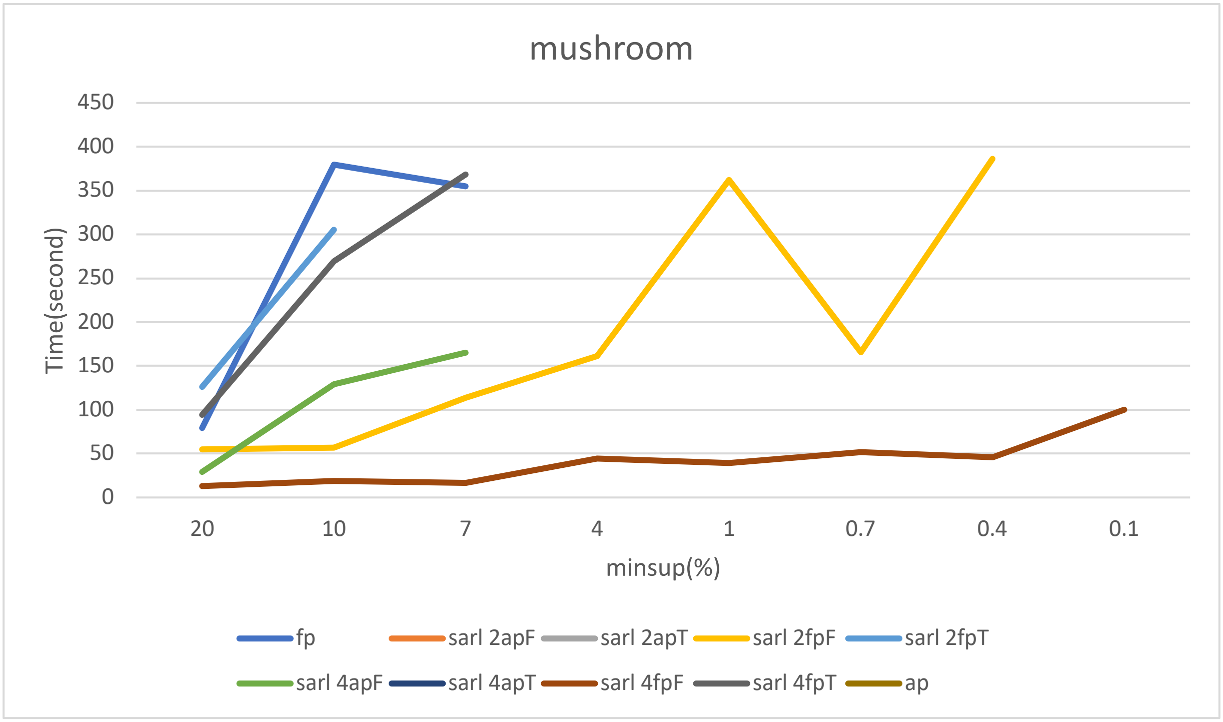


Figure 21 Mushroom Dataset Test Results

According to Figure 21, the results show that the four-partition, FP-Growth-based, heuristic SARL algorithm scales the best for this dataset regardless of the *minsup* value. It is 6 to 20.8 times faster than the FP-Growth algorithm when the FP-Growth algorithm does not exceed the time limit. Although it does not cover all frequent itemsets or association rules, depending on the application, this could be a great tradeoff. The two-partition, FP-Growth-based, heuristic SARL algorithm is the second-best in terms of running time. It has a higher accuracy with acceptable running time except for the test with 0.1% *minsup*. Two precise, FP-Growth based SARL algorithms mostly outperform the FP-Growth and the Apriori algorithms. The Apriori algorithm does not perform well on this dataset, and it does not terminate for any of the experiments. The average transaction width might have a high impact on the performance of the Apriori algorithm.

The second dataset we have tested is T10I4D100K. It has the following statistics:

Number of unique items: 870

The average size of transactions: 10

The average size of the maximal potentially large itemsets: 4

Number of transactions: 100000

File size: 4MB

The algorithms were tested on T10I4D100K for *minsup* of 10%, 4%, 1%, 0.7%, and 0.4%. This dataset has a medium size (for this environment), so the time limit is set to 300 seconds for each of the 50 experiments.

Table 64 shows the results for T10I4D100K:

	fp	sarl 2apF	sarl 2apT	sarl 2fpF	sarl 2fpT	sarl 4apF	sarl 4apT	sarl 4fpF	sarl 4fpT	ap
10	2.835414	19.57188	19.81187	19.11257	18.06593	18.71221	20.30663	19.44224	19.68889	5.622139
4	4.288454	18.61193	18.11847	18.25904	18.33747	18.43984	18.2938	18.39262	19.37539	18.49691
1	277.5966	32.34425	26.54587	21.23481	279.7529	21.47209	25.65941	20.94079	274.8818	
0.7	288.9096	24.09774	30.65488	23.6652		22.95777	32.24238	23.90843		
0.4		58.42791	89.43828	199.8875		56.80203	88.83656	207.7374		

Table 64 T10I4D100K Test Results

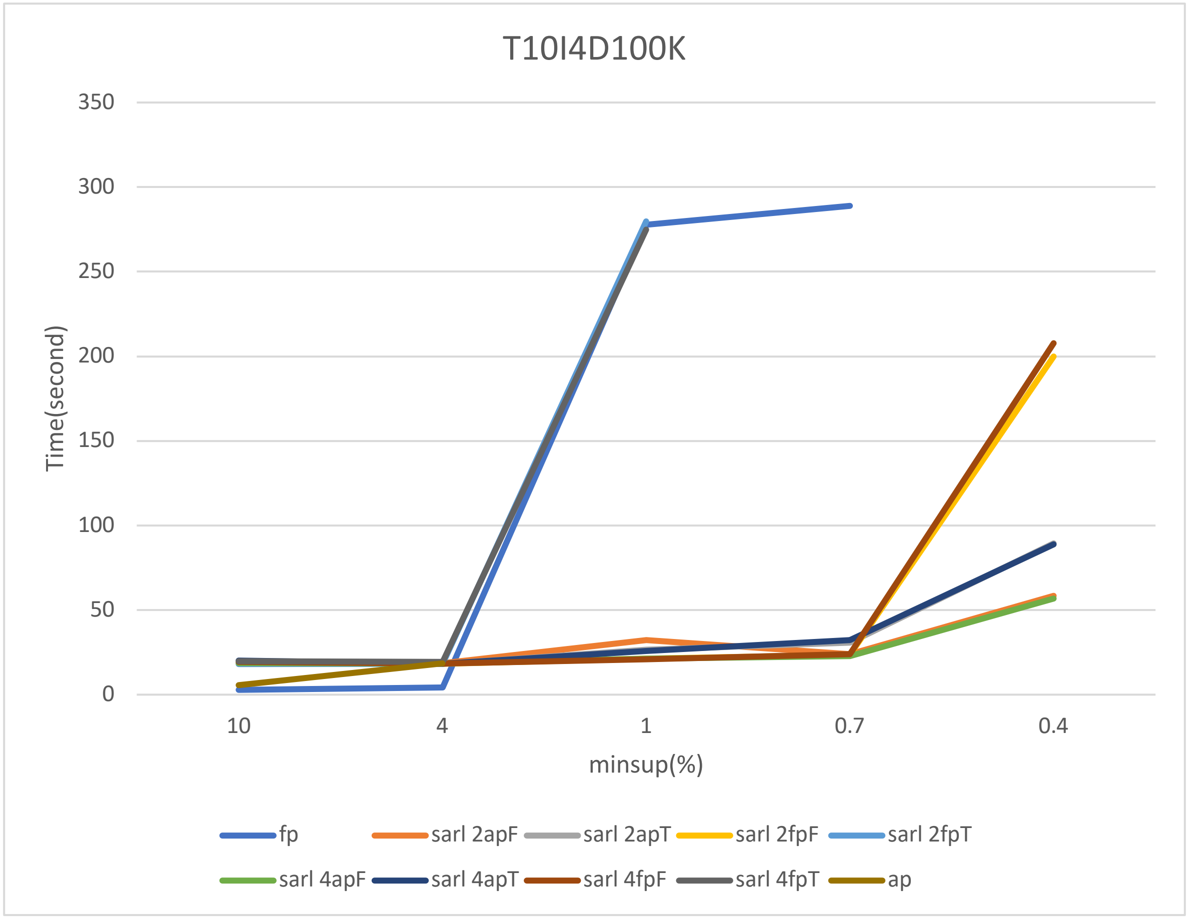


Figure 22 T10I4D100K Test Results

From Figure 22, the Apriori algorithm has an average performance for the initial *minsup* of 10% and 4%. However, it quickly reached the maximum running time after that and unable to finish the task in time for all subsequent *minsup*. The FP-Growth has a better performance. It was the fastest for higher *minsup* of 10% and 4%, but jumped to almost 300 seconds for 1% and 0.7%, before having timeout at 0.4%. All settings of the SARL algorithm outperform the Apriori and the FP-Growth algorithm for middle and low *minsup*. The SARL algorithm is slightly slower at a high *minsup* of 10%, and they were tied with the Apriori but was slightly slower than FP-Growth at *minsup* of 4%.

The dataset T40I10D100K has the following statistics:

Number of unique items: 942

The average size of transactions: 40

The average size of the maximal potentially large itemsets:10

Number of transactions:100000

File size: about 15 MB

This relatively large-size dataset was tested on *minsup* values of 20%, 10%, 7%, and 4%. The maximum running time was set to 300 seconds each for a total of 40 experiments. Table 65 shows the results of the experiments:

	fp	sarl 2apF	sarl 2apT	sarl 2fpF	sarl 2fpT	sarl 4apF	sarl 4apT	sarl 4fpF	sarl 4fpT	ap
20	8.736294	229.0365	267.8287	235.9384	235.0602	232.3693	229.7591	231.474	232.7452	23.49573
10		228.2672	226.3674	226.8315	226.7898	239.0484	237.0974	233.1036	230.0035	158.8888
7		236.5143	247.8587	233.9087		235.3071	253.1851	238.6583		
4		241.4238	256.5347	252.5584		242.9868	256.7242	241.8205		

Table 65 T40I10D100K Test Results

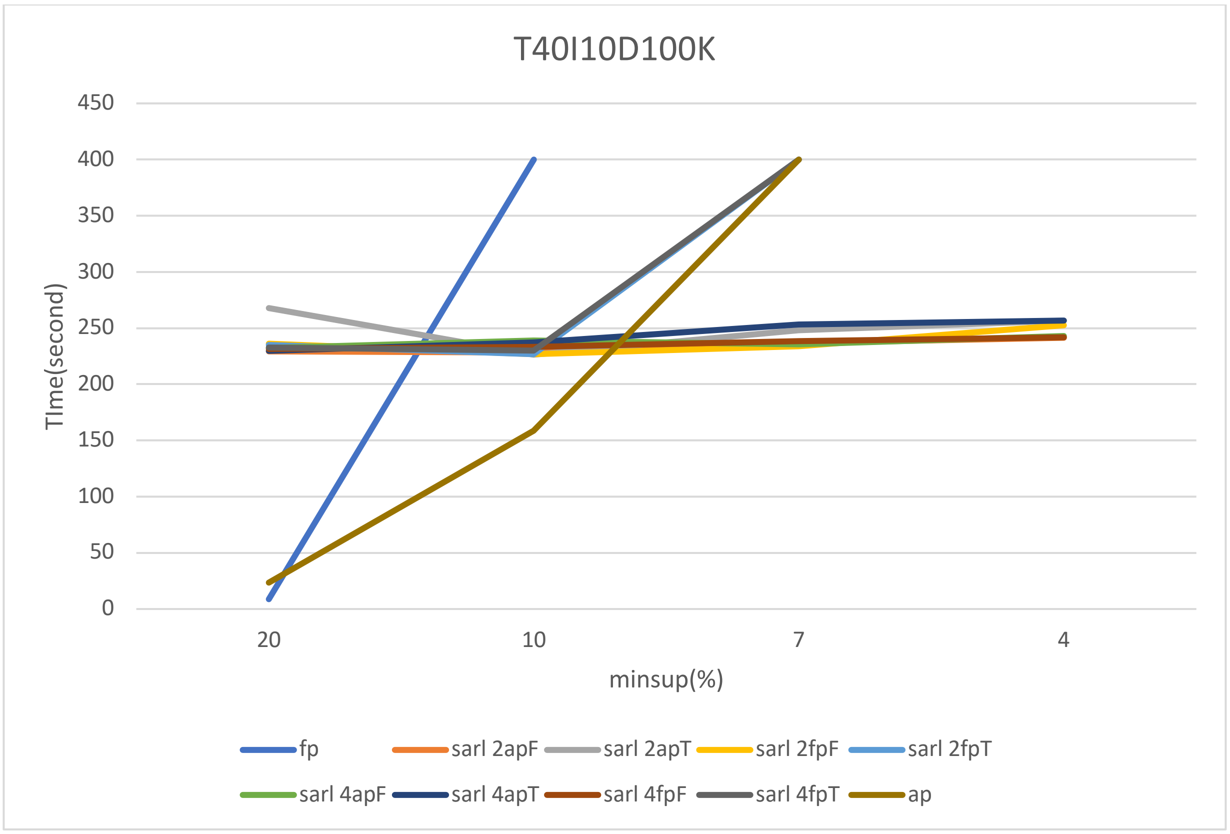


Figure 23 T40I10D100K Test Results

The results of the experiments(shown in Table 65 and Figure 23) show an obvious distinction between the scalability of different algorithms. All settings of the SARL algorithm demonstrates very high scalability. Almost all settings of the SARL algorithm had stable running time throughout the entire range of *minsup*. Surprisingly, the Apriori algorithm performs better than the FP-Growth algorithm with *minsup* between 20% and 7%. However, it is still unable to terminate within the time limit for *minsup* = 4%. Lastly, the FP-Growth algorithm does not scale very well on this dataset. It failed to terminate within the given time for both 7% and 4% of *minsup*.

We have designed and conducted experiments on small and large microarray datasets to demonstrate the scalability and accuracy of our algorithm. The experiments are based on the following configuration:

OS: macOS Big Sur

CPU: Apple M1

Memory: 8 GB

Disk: 256 GB, SSD

Programming Language: Python 3.7

All three datasets are downloaded from ArrayExpress [6]. We test the SARL algorithm on each of the datasets with various *minsup* configurations. Here, *minsup* refers to the minimum number of occurrences rather than the percentage of that.

1. E-MTAB-9030 - microRNA profiling of muscular dystrophies dataset

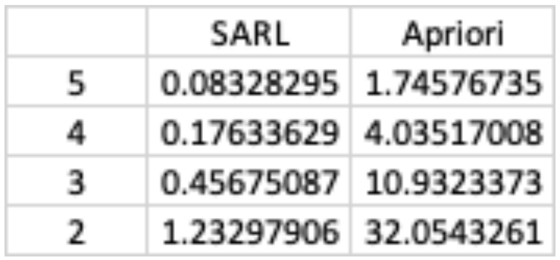
The dataset has the following metrics:

File size: 4 KB

Number of genes: 29

Number of assays: 15

This is a relatively small dataset. The experiments are done repeatedly for *minsup* of 5, 4, 3, The results are shown in Table 66.



	SARL	Apriori
5	0.08328295	1.74576735
4	0.17633629	4.03517008
3	0.45675087	10.9323373
2	1.23297906	32.0543261

Table 66 Experiment Results of Dataset E-MTAB-9030

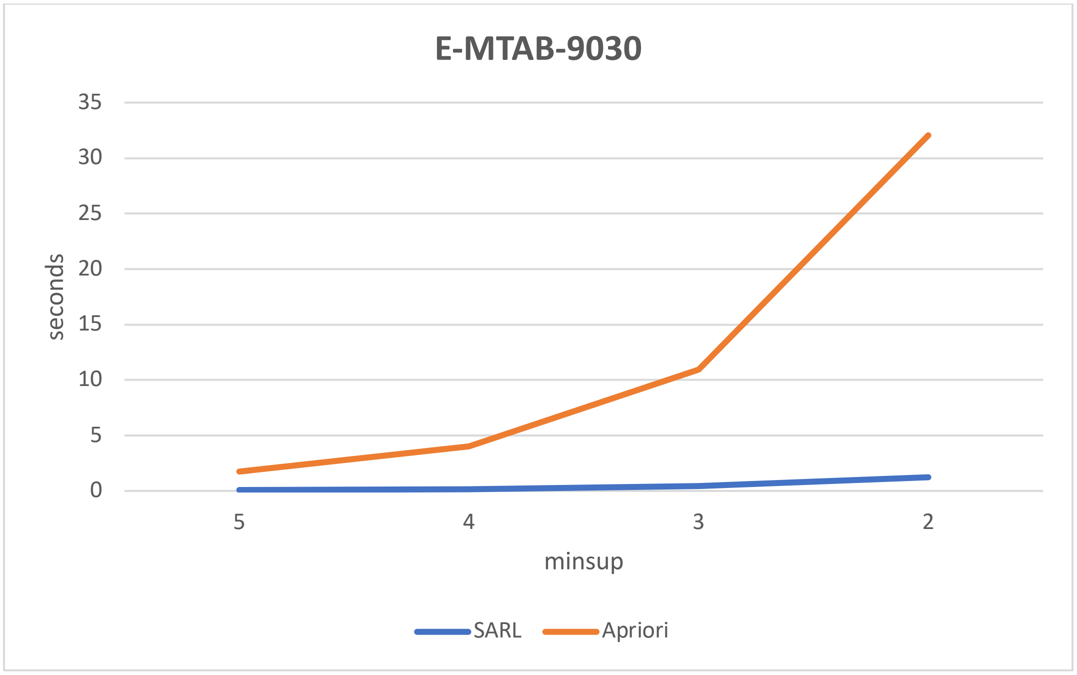


Figure 24 Experiment Results of Dataset E-MTAB-9030 as a Chart

According to Figure 24, the SARL heuristic runs faster than the Apriori algorithm on all *minsup* configurations. We can see the running time becomes larger as the *minsup* goes down. For the test case where *minsup* is 2, the SARL algorithm performs 26 times faster than the Apriori algorithm.

Figure 25 shows the accuracy of the SARL heuristic is between 0.62 and 0.67. The accuracy is calculated based on the 100 most important frequent itemsets. The results are 62% to 67% accurate based on the association rules derived by the Apriori algorithm with the 100 most important frequent itemsets. It seems the accuracy may be acceptable considering the purpose of this research and the speedup, i.e., to have a computational tool that can more quickly derive the important associations among genes for iterative investigation.

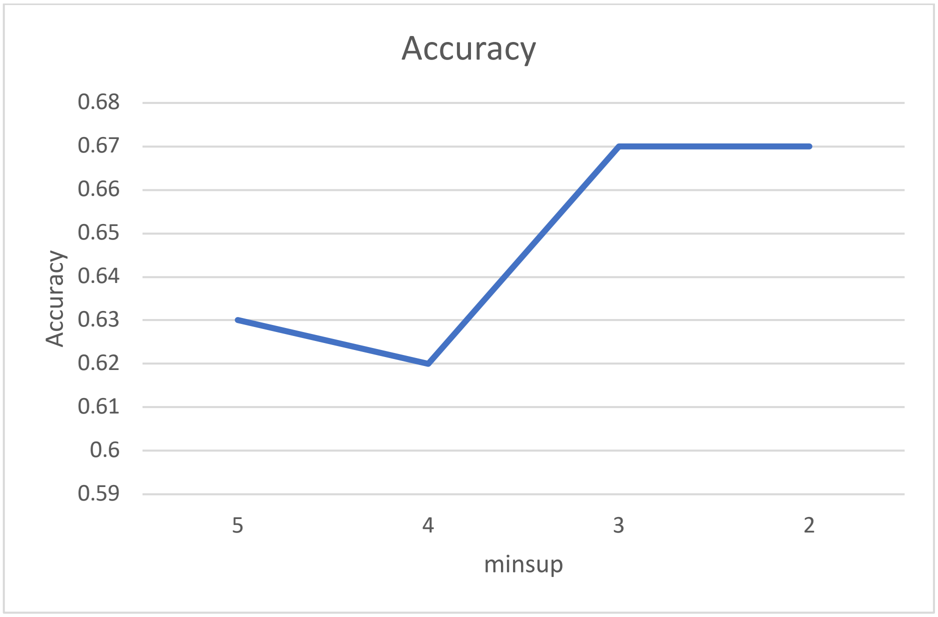


Figure 25 SARL Heuristic Accuracy on Dataset E-MTAB-9030

2. E-MTAB-8615

- Molecular characterisation of TP53 mutated squamous cell carcinomas of the lung identifies BIRC5 as a putative target for therapy

The dataset has the following metrics:

File size: 73.4 MB

Number of genes: 58202

Number of assays: 209

This dataset is larger than the previous one. Traditionally, finding association rules with the Apriori algorithm on the full dataset will take an extremely long time.

The result of the experiment is shown in Table 67.

	SARL	Apriori
10	1.929843903	0.01297688
9	1.779034853	0.01288104
8	1.654714823	0.01302099
7	1.796648979	0.01289988
6	1.666769028	0.01285505
5	1.721930981	0.01291895
4	1.836429834	0.02120876
3	1.690389156	0.77273917
2	44.13863969	278.983532
1	Interrupted	Interrupted

Table 67 Experiment Results on E-MTAB-8615 Dataset

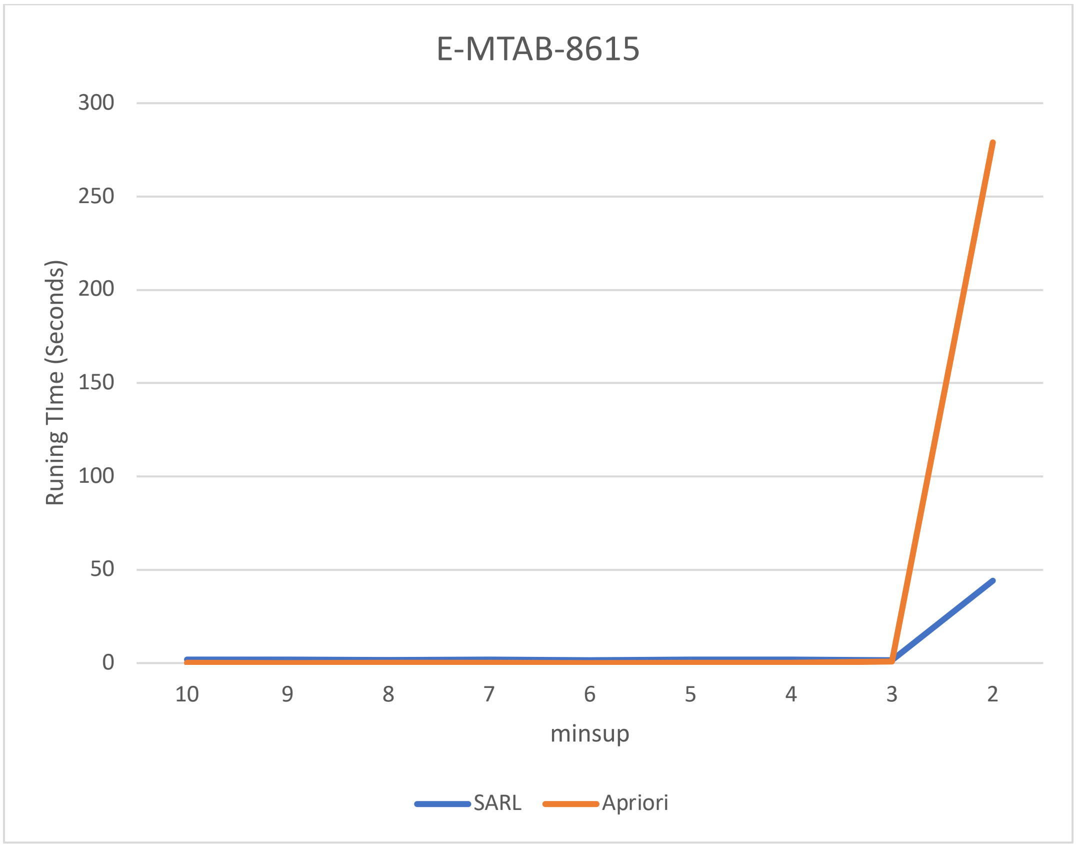


Figure 26 Experiment Results as a Graph on E-MTAB-8615 Dataset

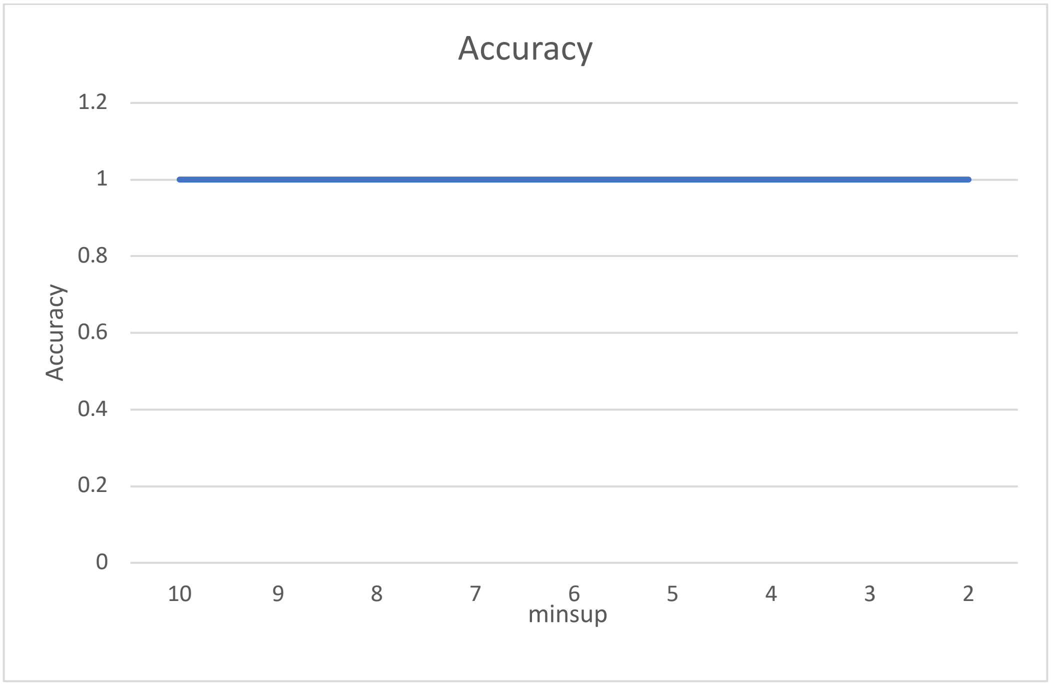


Figure 27 SARL Heuristic Accuracy

According to Figure 26, the SARL heuristic performs similarly comparing to the Apriori algorithm on a minsup range between 10 to 3, and both algorithms can finish within 2 seconds. This is because the SARL heuristic has a small overhead, and the size of the processed data is very small on these minsup configurations. However, when it comes to minsup = 2, the SARL heuristic outperforms the Apriori algorithm by a large margin. The SARL heuristic finished the task in less than 45 seconds comparing to 279 seconds for Apriori. Figure 27 shows that the SARL heuristic has 100% accuracy across all minsup configurations. We believe the SARL heuristic performs better than the Apriori algorithm overall on this dataset because it achieves the same goal with a fraction of time.

3. E-MTAB-6703 - A microarray meta-dataset of breast cancer

The dataset has the following metrics:

File size: 780.2 MB

Number of genes: 20546

Number of assays: 2302

This dataset is about ten times larger than the second dataset. However, based on the purpose of this research, we believe the total number of rules generated from the previous dataset is already overwhelmingly large. Therefore, we also reduced this dataset based on the method mentioned in this dissertation to speed up the calculation.

The experiment results are shown in Table 68.

	SARL	Apriori
5	0.023450851	0.011446238
4	0.01734972	0.019985914
3	0.0338943	1.446111679
2	0.142143965	99.75626707

Table 68 Experiment Results for E-MTAB-6703 Dataset

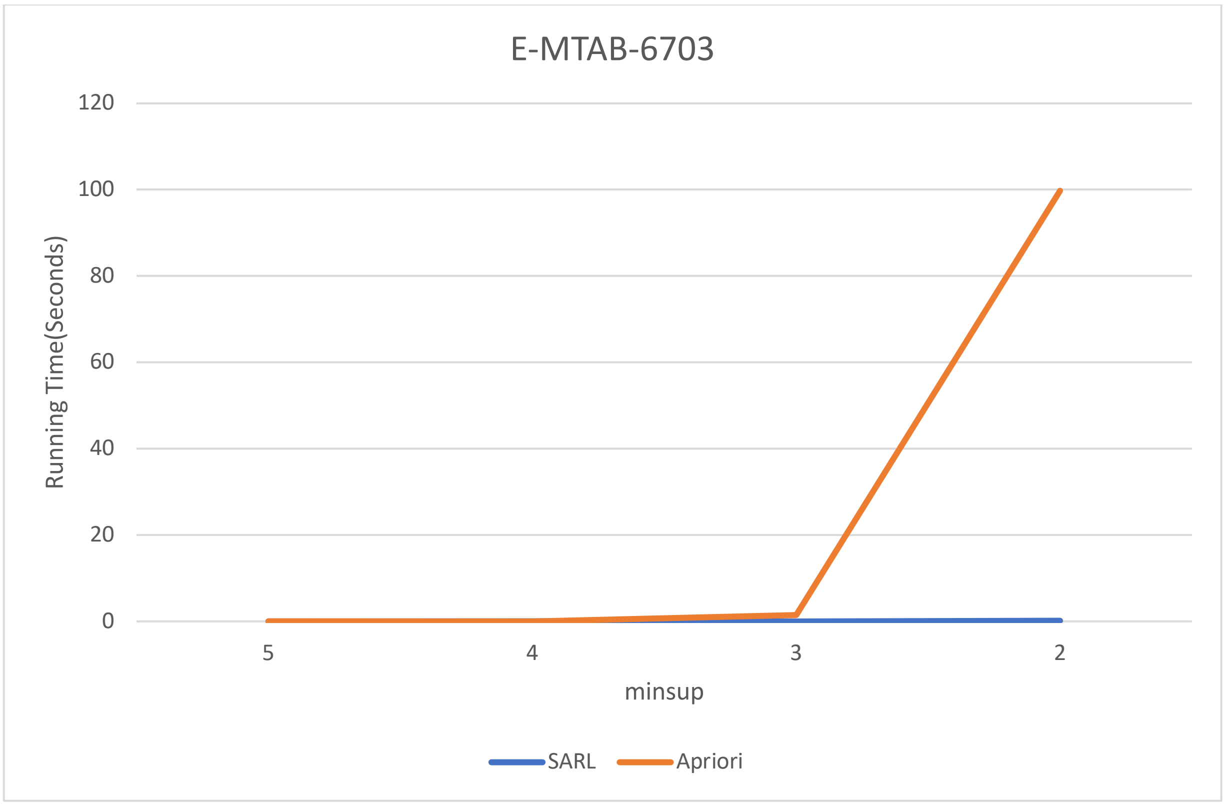


Figure 28 Experiment Results as a Graph for E-MTAB-6703

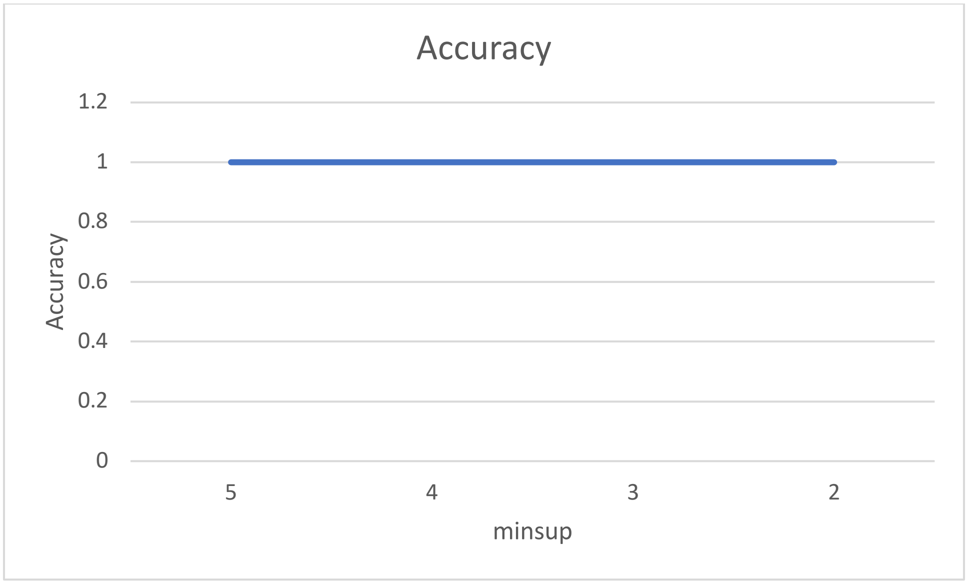


Figure 29 SARL Accuracy for E-MTAB-6703

From Figure 28, we can find a similar performance result as the previous datasets, but the SARL performs better for the larger dataset. The SARL heuristic is 700 times faster than the Apriori algorithm on minsup = 2. More surprisingly, according to Figure 29, SARL is accurate on all minsup configurations.

Chapter VI: CONCLUSIONS AND FUTURE WORK

In this research, we have proposed a scalable, highly parallelizable association rule mining heuristic algorithm (the SARL algorithm). The contributions include the use of the divide-and-conquer method to speed up complex computations, the use of an item association graph that provides an efficient estimation of potential frequent itemsets, and the use of the MLkP algorithm to divide the items into partitions while minimizing the loss of information. We have shown the scalability of the SARL heuristic through a series of experiments. The results indicate that the SARL heuristic has better scalability, with high accuracy, than both the Apriori and the FP-Growth algorithms in most cases.

We also covered an application of SARL for microarray datasets. The SARL heuristic algorithm utilizes the ternary discretization method, divide and conquer paradigm, graph theory, and graph partitioning algorithm to significantly speed up the association rule learning process compared to traditional algorithms. The algorithm also shows space efficiency. The rule ranking algorithm based on the importance saves time for researchers by showing the most important rules first. The rules found and ranked by the SARL heuristic cover both inter-genes rules and gene-disease rules. We compared our algorithm with Apriori, the most commonly used association rule learning algorithm, through a series of experiments. The results show that our algorithm has a significant speedup while still maintains high accuracy.

As discussed, the proposed heuristic is limited by the space requirement that the memory should be large enough to accommodate the IAG (proportional to d^2 where d is the number of unique items in the transactions) which we think may be a reasonable assumption in practice.

In the future, we plan to extend our work with the following tasks:

Develop a parallel version of the SARL heuristic and its implementation: The transaction partitions can be considered as independent datasets, and we can easily run the modified Apriori algorithm or FP-Growth algorithm on each of the transaction partitions in parallel and then merge the results (frequent three or higher itemsets) together along with the frequent one and two itemsets to obtain the total frequent itemsets. Each parallel processor does not need to communicate with others during the computation since all the information needed is already included in the local dataset. This would result in maximum utilization of each processor.

Study how different characteristics of the datasets influence the performance of the SARL heuristic: Although we know that the SARL heuristic has excellent performance for most datasets, the exact speed and accuracy of the SARL heuristic are still unpredictable. We think by applying some statistical measurements on the dataset, it is possible to estimate the accuracy and speed of the SARL heuristic roughly. This will help the user to determine if using the SARL heuristic is beneficial enough compared to other accurate algorithms.

Incremental Learning on Multiple Datasets: Nowadays, new microarray datasets are added to databases around the world on a daily basis. Among them, some of them focus on the same sets of genes, and others may have overlapping gene components. This brings an interesting question: can we learn association rules across multiple microarray datasets to get a larger number of more convincing rules? The answer is yes. It is possible and quite useful to learn association rules from multiple datasets. In fact, a prominent advantage of the SARL algorithm is the ability to do incremental learning across multiple datasets.

Assume we already examined and ran the SARL algorithm to learn association rules on datasets A, which includes genes G1, G2, and G3. Now, a dataset B is added with genes G1, G2, G3, and G4. We can extend the association rules from dataset A on G1, G2, and G3 in dataset B. G4 is removed from B since we cannot associate G4 to dataset A. Firstly, we compare the reference conditions (assays) between datasets A and B and find a coefficient for each gene expression level:

$$c1 = \frac{A1}{B1}$$

$$c2 = \frac{A2}{B2}$$

$$c3 = \frac{A3}{B3}$$

A1 through *A3* are expression levels of reference condition in dataset A, *B1* through *B3* are expression levels of reference condition in dataset B. *c1*, *c2*, and *c3* are coefficients we want to find.

Next, all expression levels in dataset B are divided by the corresponding coefficient:

$$\frac{E_i}{c_i} \rightarrow E_i$$

Where E_i is gene expression level with gene number i, and c_i is the coefficient found in the previous step for gene i. Now, expression levels in dataset B are adjusted for the differences in experimental conditions, we then are ready to run the SARL algorithm on dataset B.

The following is an example of combining two datasets:

	Assay 1	Assay 2
G1	5	7
G2	6	2
G3	3	1

Table 69 Dataset A

	Assay 1	Assay 2
G1	5	2
G2	3	5
G3	8	7
G4	3	5

Table 70 Dataset B

According to Table 69 and Table 70, assuming Assay2 is selected to be the reference condition in both datasets. We may calculate c1, c2, and c3 as:

c1 = 7/2 = 3.5

c2 = 2/5 = 0.4

c3 = 1/7 = 0.14

After dividing all the expression levels in dataset B by the corresponding coefficient we have a combined dataset (shown in Table 71):

	Assay 1	Assay 2	Assay 3
G1	5	7	1.43
G2	6	2	7.5
G3	3	1	57.14

Table 71 The Combined Dataset

The process of learning association rules on datasets A and B combined is simple. We run the SARL algorithm on the normalized dataset B until all support values are found. We can then merge the support values found for dataset B with the support values found for dataset A. After eliminating infrequent itemsets based on the new *minsup* value, we can generate the association rules.

REFERENCES

1. Agrawal, R., & Srikant, R. (1994, September). Fast algorithms for mining association rules. In Proc. 20th int. conf. very large data bases, VLDB (Vol. 1215, pp. 487-499).
2. Agrawal, R., Imieliński, T., & Swami, A. (1993, June). Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD international conference on Management of data (pp. 207-216).
3. Agarwal, R. C., Aggarwal, C. C., & Prasad, V. V. V. (2001). A tree projection algorithm for generation of frequent item sets. Journal of parallel and Distributed Computing, 61(3), 350-371.
4. A. J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning. J. Global Optimization, 29(2):225-241, 2004.
5. Alagukumar, S., & Lawrance, R. (2015). A selective analysis of microarray data using association rule mining. Procedia Computer Science, 47, 3-12.
6. Athar A. et al., 2019. ArrayExpress update - from bulk to single-cell expression data. Nucleic Acids Res, doi: 10.1093/nar/gky964. Pubmed ID 30357387
7. Baralis, E., Cerquitelli, T., Chiusano, S., & Grand, A. (2013, April). P-Mine: Parallel itemset mining on large datasets. In 2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW) (pp. 266-271). IEEE.
8. Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., & Schulz, C. (2016). Recent advances in graph partitioning. In Algorithm Engineering (pp. 117-158). Springer, Cham.

9. C. Borgelt, “Frequent item set mining,” Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 2, no. 6, pp. 437–456, 2012.
10. Chee, C. H., Jaafar, J., Aziz, I. A., Hasan, M. H., & Yeoh, W. (2019). Algorithms for frequent itemset mining: a literature review. Artificial Intelligence Review, 52(4), 2603-2621.
11. Croushore, D. (2011). Frontiers of real-time data analysis. Journal of economic literature, 49(1), 72-100.
12. Cong, G., Tung, A. K., Xu, X., Pan, F., & Yang, J. (2004, June). Farmer: Finding interesting rule groups in microarray datasets. In Proceedings of the 2004 ACM SIGMOD international conference on Management of data (pp. 143-154).
13. Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
14. Dubois, E., Blättler, C., Camachon, C., & Hurter, C. (2017, June). Eye movements data processing for ab initio military pilot training. In International Conference on Intelligent Decision Technologies (pp. 125-135). Springer, Cham.
15. Dudoit, S., & Fridly, J. (2003). Introduction to classification in microarray experiments. In A practical approach to microarray data analysis (pp. 132-149). Springer, Boston, MA.
16. Feddaoui, I., Felhi, F., & Akaichi, J. (2016, August). EXTRACT: New extraction algorithm of association rules from frequent itemsets. In 2016 IEEE/ACM

International Conference on Advances in Social Networks Analysis and Mining (ASONAM) (pp. 752-756). IEEE.

17. Fournier-Viger, P., Lin, C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H. T. (2016). The SPMF Open-Source Data Mining Library Version 2. Proc. 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III, Springer LNCS 9853, pp. 36-40.
18. Galinier, P., Boujbel, Z., & Fernandes, M. C. (2011). An efficient memetic algorithm for the graph partitioning problem. Annals of Operations Research, 191(1), 1-22.
19. Goethals, B. (n.d.). Frequent Itemset Mining Dataset Repository. Retrieved 2020, from http://fimi.uantwerpen.be/data/
20. Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. ACM sigmod record, 29(2), 1-12.
21. Hoseini, M. S., Shahraki, M. N., & Neysiani, B. S. (2015, November). A new algorithm for mining frequent patterns in can tree. In 2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI) (pp. 843-846). IEEE.
22. Houtsma, M., & Swami, A. (1993). Set-oriented mining for association rules. IBM Almaden research center. research report RJ 9567, San Jose.
23. Huang, Z., Li, J., Su, H., Watts, G. S., & Chen, H. (2007). Large-scale regulatory network analysis from microarray data: modified Bayesian network learning and association rule mining. Decision Support Systems, 43(4), 1207-1225.

24. J. Heaton, “Comparing dataset characteristics that favor the Apriori, Eclat or FP-Growth frequent itemset mining algorithms,” SoutheastCon 2016, Norfolk, VA, 2016, pp. 1-7, doi: 10.1109/SECON.2016.7506659.
25. Karypis, G., & Kumar, V. (1998). Multilevelk-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed computing, 48(1), 96-129.
26. Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on scientific Computing, 20(1), 359-392.
27. Kaur, M., & Kang, S. (2016). Market Basket Analysis: Identify the changing trends of market data using association rule mining. Procedia computer science, 85(Cms), 78-85.
28. Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, *49*(2), 291-307.
29. Li, H., & Sheu, P. C. Y. (2021). A scalable association rule learning heuristic for large datasets. Journal of Big Data, 8(1), 1-32.
30. Li, L., & Weinberg, C. R. (2003). Gene selection and sample classification using a genetic algorithm and k-nearest neighbor method. In A Practical Approach to Microarray Data Analysis (pp. 216-229). Springer, Boston, MA.
31. Lin, X. (2014, June). Mr-apriori: Association rules algorithm based on mapreduce. In 2014 IEEE 5th international conference on software engineering and service science (pp. 141-144). IEEE.

32. McNicholas, P. D., Murphy, T. B., & O'Regan, M. (2008). Standardising the lift of an association rule. Computational Statistics & Data Analysis, 52(10), 4712-4721.

33. McSherry, F. (2001, October). Spectral partitioning of random graphs. In Proceedings 42nd IEEE Symposium on Foundations of Computer Science (pp. 529-537). IEEE.

34. Mukherjee, S. (2003). Classifying microarray data using support vector machines. In A practical approach to microarray data analysis (pp. 166-185). Springer, Boston, MA.

35. Nadimi-Shahraki, M. H., & Mansouri, M. (2017, March). Hp-Apriori: Horizontal parallel-apriori algorithm for frequent itemset mining from big data. In 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)((pp. 286-290). IEEE.

36. Naulaerts, S., Meysman, P., Bittremieux, W., Vu, T. N., Vanden Berghe, W., Goethals, B., & Laukens, K. (2015). A primer to frequent itemset mining for bioinformatics. Briefings in bioinformatics, 16(2), 216-231.

37. Park, J. S., Chen, M. S., & Yu, P. S. (1995). An effective hash-based algorithm for mining association rules. Acm sigmod record, 24(2), 175-186.

38. Pyun, G., Yun, U., & Ryu, K. H. (2014). Efficient frequent pattern mining based on linear prefix tree. Knowledge-Based Systems, 55, 125-139.

39. Quackenbush, J. (2002). Microarray data normalization and transformation. Nature genetics, 32(4), 496-501.

40. Song, M., & Rajasekaran, S. (2006). A transaction mapping algorithm for frequent itemsets mining. IEEE transactions on knowledge and data engineering, 18(4), 472-481.

41. Stone, A., Shiffman, S., Atienza, A., & Nebeling, L. (2007). The science of real-time data capture: Self-reports in health research. Oxford University Press.

42. Walshal, C. (2020). The Graph Partitioning Archive. https://chriswalshaw.co.uk/partition/.

43. X. Y. Yang, Z. Liu and Y. Fu, "MapReduce as a programming model for association rules algorithm on Hadoop," The 3rd International Conference on Information Sciences and Interaction Sciences, Chengdu, 2010, pp. 99-102, doi: 10.1109/ICICIS.2010.5534718.

44. Y. Shiau and S. Liang, "Real-Time Network Virtual Military Simulation System," 2007 11th International Conference Information Visualization (IV '07), Zurich, 2007, pp. 807-812, doi: 10.1109/IV.2007.93.

45. Zaki, M. J. (2000). Scalable algorithms for association mining. IEEE transactions on knowledge and data engineering, 12(3), 372-390.

46. Zhang, B. T., & Hwang, K. B. (2003). Bayesian network classifiers for gene expression analysis. In A practical approach to microarray data analysis (pp. 150-165). Springer, Boston, MA.