Journal of Big Data

RESEARCH Open Access

A scalable association rule learning and recommendation algorithm for large-scale microarray datasets

Haosong Li and Phillip C.-Y. Sheu*

*Correspondence:
psheu@uci.edu
Department of Electrical Engineering and Computer Science, University of California, Irvine, USA

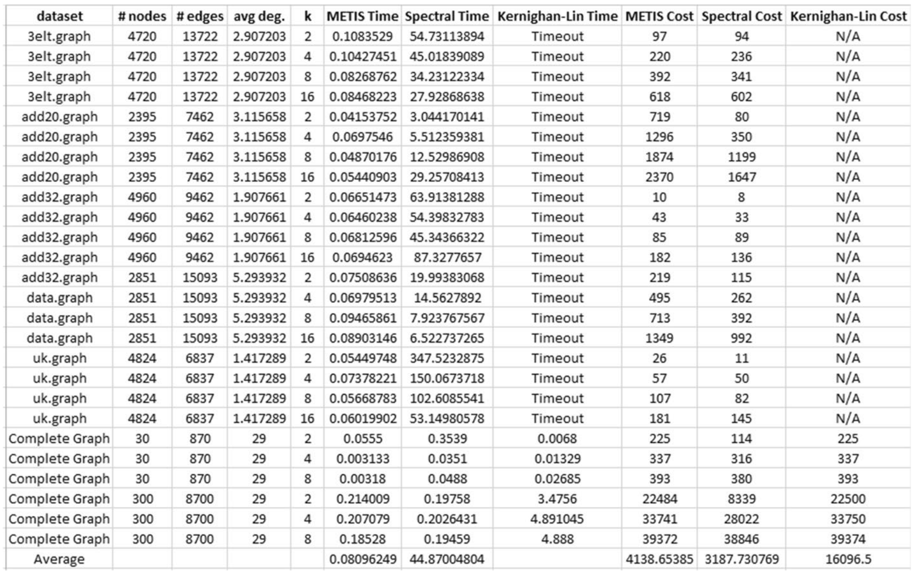
Abstract
Association rule learning algorithms have been applied to microarray datasets to find association rules among genes. With the development of microarray technology, larger datasets have been generated recently that challenge the current association rule learning algorithms. Specifically, the large number of items per transaction significantly increases the running time and memory consumption of such tasks. In this paper, we propose the Scalable Association Rule Learning (SARL) heuristic that 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 the SARL algorithm outperforms the Apriori algorithm by one to three orders of magnitude.

Keywords: Association rule learning, Microarray dataset, Frequent itemset mining, Scalability, Graph partitioning, Apriori algorithm

Introduction

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 (Fig. 1).

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 the Scalable Association Rule



dataset	# nodes	# edges	avg deg.	k	METIS Time	Spectral Time	Kernighan-Lin Time	METIS Cost	Spectral Cost	Kernighan-Lin Cost
3elt.graph	4720	13722	2.907203	2	0.1083529	54.73113894	Timeout	97	94	N/A
3elt.graph	4720	13722	2.907203	4	0.10427451	45.01839089	Timeout	220	236	N/A
3elt.graph	4720	13722	2.907203	8	0.08268762	34.23122334	Timeout	392	341	N/A
3elt.graph	4720	13722	2.907203	16	0.08468223	27.92868638	Timeout	618	602	N/A
add20.graph	2395	7462	3.115658	2	0.04153752	3.044170141	Timeout	719	80	N/A
add20.graph	2395	7462	3.115658	4	0.0697546	5.512359381	Timeout	1296	350	N/A
add20.graph	2395	7462	3.115658	8	0.04870176	12.52986908	Timeout	1874	1199	N/A
add20.graph	2395	7462	3.115658	16	0.05440903	29.25708413	Timeout	2370	1647	N/A
add32.graph	4960	9462	1.907661	2	0.06651473	63.91381288	Timeout	10	8	N/A
add32.graph	4960	9462	1.907661	4	0.06460238	54.39832783	Timeout	43	33	N/A
add32.graph	4960	9462	1.907661	8	0.06812596	45.34366322	Timeout	85	89	N/A
add32.graph	4960	9462	1.907661	16	0.0694623	87.3277657	Timeout	182	136	N/A
add32.graph	2851	15093	5.293932	2	0.07508636	19.99383068	Timeout	219	115	N/A
data.graph	2851	15093	5.293932	4	0.06979513	14.5627892	Timeout	495	262	N/A
data.graph	2851	15093	5.293932	8	0.09465861	7.923767567	Timeout	713	392	N/A
data.graph	2851	15093	5.293932	16	0.08903146	6.522737265	Timeout	1349	992	N/A
uk.graph	4824	6837	1.417289	2	0.05449748	347.5232875	Timeout	26	11	N/A
uk.graph	4824	6837	1.417289	4	0.07378221	150.0673718	Timeout	57	50	N/A
uk.graph	4824	6837	1.417289	8	0.05668783	102.6085541	Timeout	107	82	N/A
uk.graph	4824	6837	1.417289	16	0.06019902	53.14980578	Timeout	181	145	N/A
Complete Graph	30	870	29	2	0.0555	0.3539	0.0068	225	114	225
Complete Graph	30	870	29	4	0.003133	0.0351	0.01329	337	316	337
Complete Graph	30	870	29	8	0.00318	0.0488	0.02685	393	380	393
Complete Graph	300	8700	29	2	0.214009	0.19758	3.4756	22484	8339	22500
Complete Graph	300	8700	29	4	0.207079	0.2026431	4.891045	33741	28022	33750
Complete Graph	300	8700	29	8	0.18528	0.19459	4.888	39372	38846	39374
Average					0.08096249	44.87004804		4138.65385	3187.730769	16096.5

Fig. 1 Results of experiment that compare MLkP, Kernighan-Lin, and Spectral Partitioning algorithms

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.

Contributions

There are three main contributions in this paper:

1. The SARL heuristic with the divide and conquer methodology can increase the efficiency and scalability of association rule mining but still maintain reasonable accuracy.
2. The rule ranking algorithm calculates the importance and ranks the rules, so the investigator does not have to search through millions of rules.
3. We consider gene-disease associations. Each important association rule between genes and disease can be identified and highlighted in the result.

Related work

Several association rule learning algorithms have previously been applied to microarray datasets. The fundamental ones include the Apriori algorithm and the FP-Growth algorithm. Researchers have also created other algorithms or heuristics that find association rules with an approximation methodology.

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 $L_{k-1} \times L_{k-1}$ 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, L_{k-1}, and the candidates in level k, C_k, need to be stored in the memory. It generates the minimum number of candidates based on the $L_{k-1} \times L_{k-1}$ (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 [2]. 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 relatively 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. 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 7) into *k* balanced partitions. An efficient graph partitioning algorithm is crucial since the balanced graph partitioning problem is NP-complete [3]. 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 [4], the Multilevel k-way Partitioning Algorithm (MLkP) [5], and the recursive version of the Spectral Partitioning Algorithm [6]. Other graph partitioning algorithms include the Tabu search based MAGP algorithm [7] and the flow-based KaFFPa algorithm [8].

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 [9] are used. They are chosen because they are, on the one hand, large enough for us to study scalability and, on the other hand, manageable by our machine.

The datasets are desired to be 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.

As shown in Fig. 1, 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 that the MLkP algorithm has the best overall performance, we choose to use this algorithm for graph partitioning in our algorithm.

1) Applications Related to Association Rule Learning Algorithms on Microarray Data

   Both Apriori and FP-Growth algorithms have been applied to microarray datasets [10], according to [10]. 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 [11] 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 [12] 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 paper tolerate"" certain accuracy for better scalability so the investigator may navigate a dataset iteratively (We call it "iterative investigation" in this paper) to converge in the search process quickly.

2) 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 [13]. Many algorithms have been applied to solve classification problems. The most studied algorithms include the Bayesian network [14], Support Vector Machine [15], and k-Nearest Neighbor [16].

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.

Our solution

Data preprocessing

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 [17].

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.

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 [12]. 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.

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"}.

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:

$$E_g = \left| \frac{\sum_{j=1}^{m} K_j}{m} - \frac{\sum_{i=1}^{t} K_g}{t} \right|$$

In the above, K_j is the gene expression level of gene j, K_g is the gene expression level for gene g. The first part, $\frac{\sum_{j=1}^{m} K_j}{m}$, calculates the average expression level of all genes, and the second part, $\frac{\sum_{i=1}^{t} K_g}{t}$, finds the average expression level of gene g. The difference between the two is the deviation for gene g. 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.

The SARL-heuristic

Definitions

The following definitions are used in this paper:

1) *K-itemset*: an itemset with k items
2) *Support*: number of occurrences of an itemset in the dataset
3) *Minsup*: the minimum requirement of support. The user usually provides this. Itemsets with support < *minsup* are eliminated.
4) *Confidence*: the indication of robustness of a rule in terms of percentage. Confidence(XY) = support($X \cup Y$)/support(X)
5) *Minconf*: the minimum requirement of confidence. The user usually provides this. Rules with confidence < minconf are eliminated.
6) *Item-Association Graph:* a graph structure that stores the frequent associations between pairs of items.
7) *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 the edge weights that are cut off.
8) E_g: importance of gene g.
9) I_r : Importance of rule r.

A scalable heuristic algorithm—SARL-heuristic

The following is an outline of our scalable heuristic [18].

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.

An example

Suppose the microarray dataset in Table 1 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 2.

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 3.

Table 1 Microarray dataset

	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	1.12	0.13	0.46	0.89	1.88	0.3	0.98

Table 2 Relative expression levels

	Assay 1	Assay 2	Assay 3	Assay 4	Assay 5	Assay 6	Assay 7
Gene 1	12.09091	44.33333	0.880795	3.911765	0.130264	133	4.75
Gene 2	0.244742	0.221453	0.934307	0.888889	0.16732	0.059953	0.646465
Gene 3	1.356061	0.366053	1.704762	1.306569	0.211834	0.101936	1
Gene 4	0.717949	1.154639	22.4	9.333333	1.098039	0.835821	5.894737
Gene 5	0.154818	0.875	7.538462	2.130435	1.101124	0.521277	3.266667

Table 3 Log-scale expression levels

	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	− 2.03067	− 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

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 4.

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 5.

Finally, we convert the transposed matrix to a transactional dataset, shown in Table 6, each expression level that equals −1 or 1 is transformed into an item. In Table 6, 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).

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 7):

The frequent two-itemsets are found afterward (shown in Table 8):

Table 4 Normalized expression levels

	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 5 Transposed matrix

	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 6 Transactional dataset

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 7 Frequent one itemsets

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

Table 8 Frequent two itemsets

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

Next, we transform the above frequent two-itemsets into an item association graph (IAG), shown in Fig. 2:

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 Figs. 3 and 4.

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 9 and 10. 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 11 and 12 show the transaction partitions:

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 11:

Since the modified Apriori algorithm starts with three-itemsets, there are no additional frequent itemsets in the first partition. Table 12 shows the results found in transaction partition 2:

The final results (shown in Table 13) of frequent itemsets are simply the union of Tables 7, 8, and 11:

After running the Apriori-ap-genrules algorithm, the association rules can be found in Table 14.

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

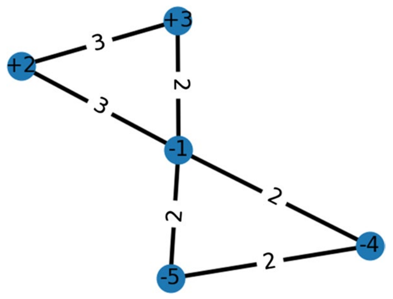


Fig. 2 An item association graph

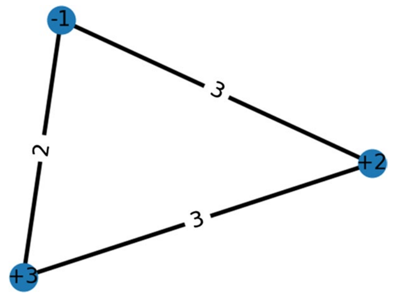


Fig. 3 Item association graph partition 1

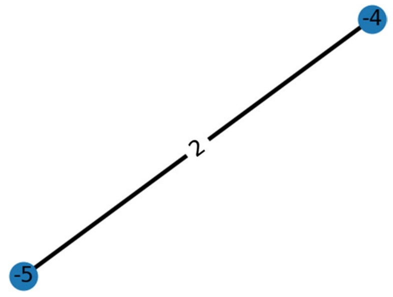


Fig. 4 Item association graph partition 2

Table 9 Transaction partition 1

TID	Items
T001	$-1, +2, +3$
T005	$-1, +2, +3$

Table 10 Transaction partition 2

TID	Items
None	None

Table 11 Frequent itemsets from transaction partition 1

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

Table 12 Frequent itemsets from transaction partition 2

Frequent Itemsets	Support
None	N/A

Table 13 Frequent itemset final results

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

will be discussed shortly. In this example, the SARL heuristic loses one frequent itemset $\{-1, -4, -5\}$ and two related rules generated from $\{-1, -4, -5\}$.

The SARL (scalable association rule learning) heuristic

We introduced the SARL heuristic in our previous paper [18]. SARL is a highly scalable heuristic algorithm for association rule learning problems on horizontal transactional datasets. In this paper, 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 paper that introduces SARL.

Table 14 Association rules generated

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

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.

We also conducted a thorough time and space complexity analysis and an error-bound analysis that can be found in the Appendix section of this paper.

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 [19]. 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.

Experiments and results

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 [20]. 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.

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, 2. The results are shown in Table 15.

According to Fig. 5, 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 6 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.

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: 58,202
- 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 16.

Table 15 Experiment results (in seconds) of dataset E-MTAB-9030

	SARL	Apriori
5	0.083	1.746
4	0.176	4.035
3	0.457	10.932
2	1.233	32.054

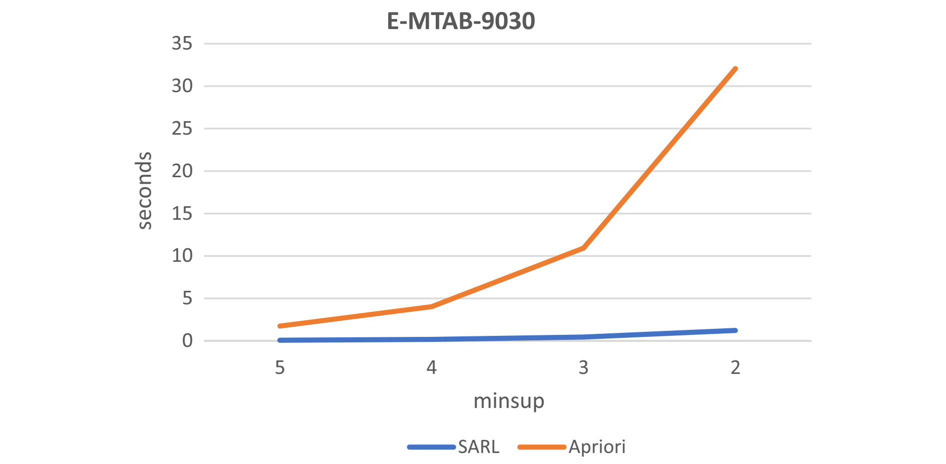


Fig. 5 Experiment results of dataset E-MTAB-9030 as a chart

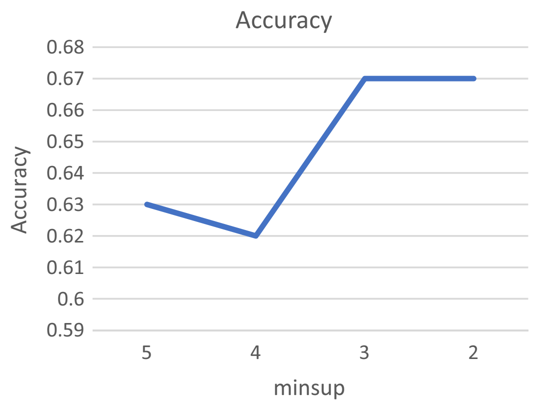


Fig. 6 SARL heuristic accuracy on dataset E-MTAB-9030

According to Fig. 7, 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 s. 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 s comparing to 279 s for Apriori. Figure 8 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.

E-MTAB-6703—A microarray meta-dataset of breast cancer

The dataset has the following metrics:

- File size: 780.2 MB

Table 16 Experiment results (in seconds) on E-MTAB-8615 dataset

	SARL	Apriori
10	1.93	0.01
9	1.78	0.01
8	1.65	0.01
7	1.8	0.01
6	1.67	0.01
5	1.72	0.01
4	1.84	0.02
3	1.69	0.77
2	44.14	278.98
1	Interrupted	Inter-rupted

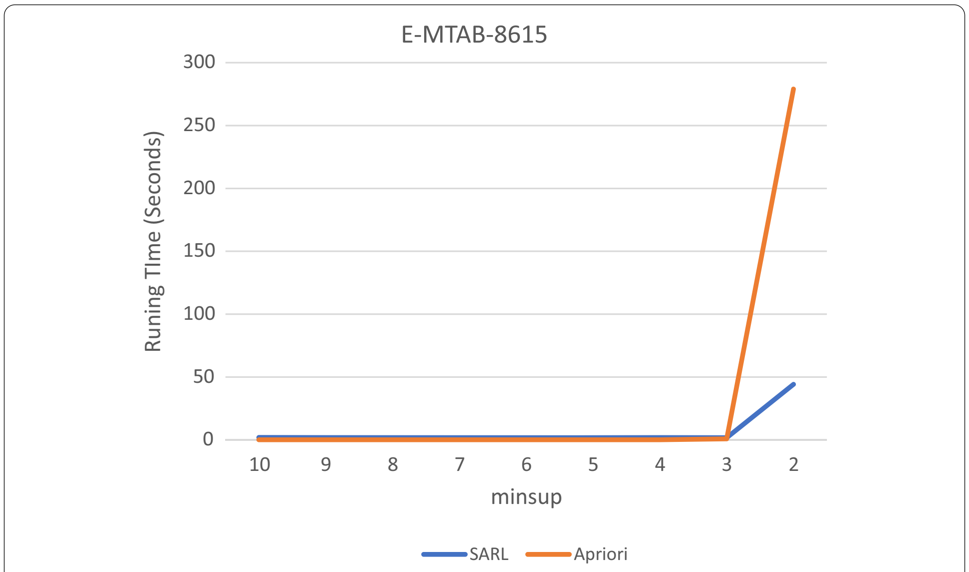


Fig. 7 Experiment results as a graph on E-MTAB-8615 dataset

- Number of genes: 20,546
- 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 paper to speed up the calculation.

The experiment results are shown in Table 17.

From Fig. 9, 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 Fig. 10, SARL is accurate on all minsup configurations.

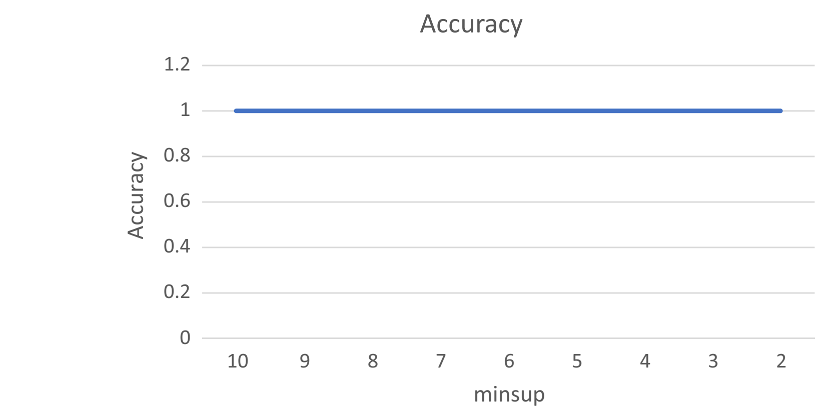


Fig. 8 SARL heuristic accuracy

Table 17 Experiment results (in seconds) for E-MTAB-6703 dataset

	SARL	Apriori
5	0.023	0.011
4	0.017	0.02
3	0.034	1.446
2	0.142	99.756

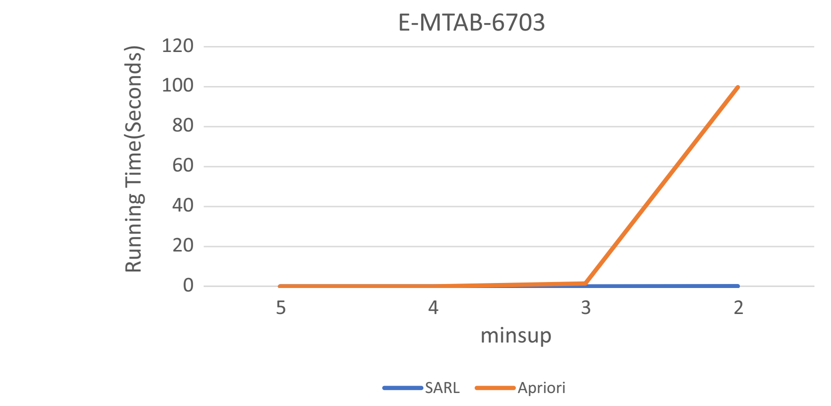


Fig. 9 Experiment results as a graph for E-MTAB-6703

Discussion and conclusions

In this paper, we proposed a new algorithm for association rule learning specifically designed 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

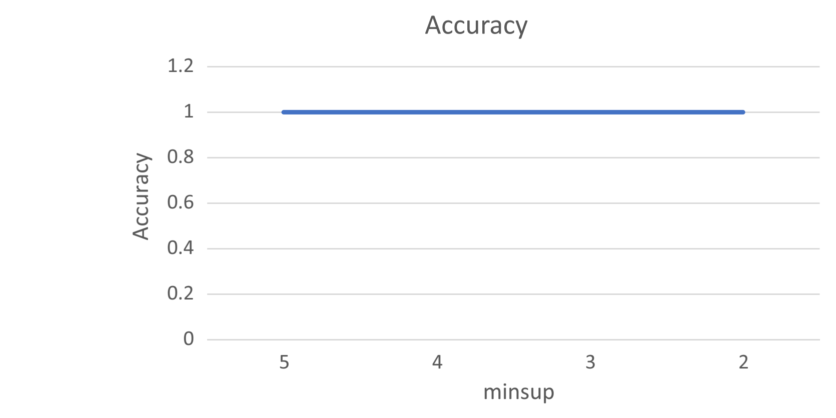


Fig. 10 SARL accuracy for E-MTAB-6703

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.

Some potential drawbacks of our algorithm include: 1. There is a small probability that some non-trivial rules are lost in the dataset partitioning stage. 2. For small datasets or high support, the performance of our algorithm may be similar or slightly higher than the Apriori algorithm due to the overhead.

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 partition 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.
- A better algorithm may be used to predict the proper thresholds of the ternary discretization. The current ternary discretization is based on empirical methods and may need several tests to find the best thresholds that reduce the dataset to a smaller size while keeping enough information. A statistical analysis of the dataset may help to decide the boundaries. Furthermore, we should consider incorporating deep neural network approach in this process to predict the best threshold.
- 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:

According to Tables 18 and 19, 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 20):

Table 18 Dataset A

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

Table 19 Dataset B

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

Table 20 The combined dataset

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

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.

Appendix

The appendix section is intended to provide a brief overview of the SARL heuristic [18].

Theorems The followings are the theorems we proposed in our SARL heuristic paper [18]. The proofs are available in the same refenced paper.

Theorem 1: Soundness—all frequent itemsets and association rules generated by the SARL heuristic are correct.
Theorem 2: Computing the frequent two itemsets is considered relatively trivial compared to computing the frequent 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.

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.

Another method to find frequent one and two itemsets is 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 many 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.

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.

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

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:

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

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

The Multilevel k-way partitioning (MLkP) algorithm [17] is an efficient graph partitioning algorithm. The time complexity is *O(E)*, where E is the number of edges in the graph.

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 [18]. It includes an implementation of the MLkP algorithm that takes a graph as the input and outputs groups of nodes separated after the partition.

Transaction partitioning

Based on the results of the MLkP algorithm that divide the items into groups P_1, P_2,… ,P_m, we can partition the transactions into the same number of groups, where each group D_i contains only the items in partition P_i. This guarantees that each partition fits into the memory. Refer to paper [18] for more details.

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.

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 minsup is set to a high value so that the expected number of itemsets is limited 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. The SARL heuristic choose between the modified Apriori algorithm and FP-Growth algorithm to complete the computation of frequent itemsets.

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.

Time complexity

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 $O\left(k * 2^{\frac{1.03d}{k}}\right) = O(2^{\frac{1.03d}{k}})$, and the total time complexity of the SARL algorithm, when the modified Apriori algorithm is selected, is $O\left(d^2T + n + d^2 + d^2 + n + 2^{\frac{1.03d}{k}}\right) = O(d^2T + n + 2^{\frac{1.03d}{k}})$. Assume $n"d$, and $2^{\frac{1.03d}{k}}"n$, 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 $O\left(\frac{2^d}{2^{\frac{1.03d}{k}}}\right) = O(2^{\frac{k-1.03}{k}d})$ times faster than the Apriori algorithm. The exponential speed up comes from the smaller number of unique items in each transaction partition. The algorithm chosen to mine frequent itemsets from the transaction partitions only needs to consider a portion of all the items for each partition.

Space complexity

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 $O\left(k * 2^{\frac{1.03d}{k}}\right) = O(2^{\frac{1.03d}{k}})$, and the total space complexity of the SARL heuristic, when the modified Apriori algorithm is selected, is $O\left((3-1) * d^2 + \frac{n}{k} + 2^{\frac{1.03d}{k}}\right) = O(d^2 + \frac{n}{k} + 2^{\frac{1.03d}{k}})$. Assume $\frac{n}{k}"d$, and $2^{\frac{1.03d}{k}}"\frac{n}{k}$, 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 $O\left(\frac{2^{\frac{1.03d}{k}}}{2^d}\right) = O\left(2^{\frac{1.03-k}{k}d}\right) = o(\frac{1}{2^{\frac{k-1.03}{k}d}})$ 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.

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.

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, $P_m = \frac{\#frequent2+itemsets}{\#totalfrequentitemsets}$, then the total error bound can be computed as $Error_{total} = P_m * P_{out}$.

Benefits of having datasets fit into the memory

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.

Abbreviations

K-itemset: An itemset with *k* items; Minsup: The minimum requirement of support. The user usually provides this. Itemsets with support < *minsup* are eliminated.; Minconf: The minimum requirement of confidence. The user usually provides this. Rules with confidence < minconf are eliminated.; IAG: Item-Association Graph; MLkP: Multilevel k-way Partitioning algorithm; SARL: Scalable Association Rule Learning; E_g: Importance of gene g.; I_r: Importance of rule r.

Acknowledgements
Not applicable

Authors' contributions

Both HL and PS are major contributors to writing the manuscript. HL and PS have made substantial contributions to the conception and design of work. HL has created the software used in this paper. Both HL and PS have approved the submitted version and have agreed on both to be personally accountable for the author's own contributions and to ensure that questions related to the accuracy or integrity of any part of the work, even ones in which the author was not personally involved, are appropriately investigated, resolved, and the resolution documented in the literature. Both authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
The datasets analyzed during the current study are available in ArrayExpress [https://www.ebi.ac.uk/arrayexpress/] and The Graph Partitioning Archive[https://chriswalshaw.co.uk/partition].

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 31 August 2021 Accepted: 15 February 2022
Published online: 28 March 2022

References
1. Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large data bases, VLDB, Vol. 1215; 1994, p. 487–99.
2. Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. ACM SIGMOD Rec. 2000;29(2):1–12.
3. Buluç A, Meyerhenke H, Safro I, Sanders P, Schulz C. Recent advances in graph partitioning. In: Algorithm engineering. Cham: Springer; 2016, p. 117–58.
4. Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. Bell Syst Tech J. 1970;49(2):291–307.
5. Karypis G, Kumar V. Multilevelk-way partitioning scheme for irregular graphs. J Parallel Distrib Comput. 1998;48(1):96–129.
6. McSherry F. Spectral partitioning of random graphs. In: Proceedings 42nd IEEE symposium on foundations of computer science. IEEE; 2001, p. 529–37.
7. Galinier P, Boujbel Z, Fernandes MC. An efficient memetic algorithm for the graph partitioning problem. Ann Oper Res. 2011;191(1):1–22.
8. Sanders P, Schulz C. Engineering multilevel graph partitioning algorithms. In European symposium on algorithms. Berlin, Heidelberg: Springer; 2011, p. 469–80.
9. Walshal C. The graph partitioning archive; 2020. https://chriswalshaw.co.uk/partition/.
10. Alagukumar S, Lawrance R. A selective analysis of microarray data using association rule mining. Procedia Comput Sci. 2015;47:3–12.
11. 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).
12. Huang Z, Li J, Su H, Watts GS, Chen H. Large-scale regulatory network analysis from microarray data: modified Bayesian network learning and association rule mining. Decis Support Syst. 2007;43(4):1207–25.
13. Dudoit S, Fridly J. Introduction to classification in microarray experiments. In: A practical approach to microarray data analysis. Boston: Springer; 2003, p. 132–49.
14. 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.
15. Mukherjee S. Classifying microarray data using support vector machines. In: A practical approach to microarray data analysis. Boston: Springer; 2003, p. 166–85.
16. Li L, Weinberg CR. Gene selection and sample classification using a genetic algorithm and k-nearest neighbor method. In: A practical approach to microarray data analysis. Boston: Springer; 2003, p. 216–29.
17. Quackenbush J. Microarray data normalization and transformation. Nat Genet. 2002;32(4):496–501.
18. Li H, Sheu PCY. A scalable association rule learning heuristic for large datasets. J Big Data. 2021;8(1):1–32.
19. McNicholas PD, Murphy TB, O'Regan M. Standardising the lift of an association rule. Comput Stat Data Anal. 2008;52(10):4712–21.
20. Athar A, et al. ArrayExpress update—from bulk to single-cell expression data. 2019. Nucleic Acids Res. https://doi.org/10.1093/nar/gky964.PubmedID30357387.

Publisher's Note