2006

Compact Representation of Association Rule

Mien K. Siao
San Jose State University

COMPACT REPRESENTATION OF ASSOCIATION RULE

A Thesis

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Mien K. Siao

December 2006

APPROVED FOR DEPARMENT OF COMPUTER SCIENCE

_____

Dr. John Avila

_____

Eric Louise

_____

Dr. T. Y. Lin

APPROVED FOR THE UNIVERSITY

_____

ABSTRACT

COMPACT REPRESENTATION OF ASSOCIATION RULE

by Mien K. Siao

Bitmap is an extremely efficient way of representing data, but the drawback is that the order of data is fixed in a bitmap. Granular computing is a new theory that frees the bitmap method from *fixed order of data* in the same manner as linear algebra frees the matrix theory from *a fixed basis*. To obtain meaningful information using data mining techniques has been a central idea in recent database applications [2]. One of the core techniques in data mining is to find associations (undirected association rules) between attribute values [4]. The complexity of finding associations is often very high.

In this work, a data warehouse is constructed utilizing bitmap fundamentals. Classic method of obtaining association rules is implemented along with the granular computing method of obtaining association rules. The performances from classic method and granular computing method is carefully analyzed and compared.

ACKNOWLEDGEMENTS

This work would not have been possible without Dr. T. Y. Lin's guidance. Committee members, Dr. John Avila and Eric Louise's support was essential to give this work an identity.

TABLE OF CONTENT

TABLE OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Data exists in various forms. Banks have data for each client to maintain bank accounts. Schools have data for its student's academic records. Data has an ever growing presence in this world. Obtaining meaningful information from data that show trends and patterns is the science of data mining. A data warehouse provides important organization and storage of data before data mining is performed. Data in a raw form is meaningless.

To facilitate better understanding, data can be analyzed through the use of "item sets." Item sets that occur frequently or show patterns lay the foundation of data mining and take the form of association rules. Large scale stores such as Safeway spreading across different locations possess humongous amounts of data due to their transaction volume. Prime reasons for chain stores to find association rules are: to adjust prices strategically, to promote new products effectively, and have the best decision support for better business.

A classical example suggests an association rule of the relationship between buying trends of diaper and beers. The item set formed in this example would be {diaper, beer}. Association rule "diaper => beer (60%)" means that three out of five customers buying diapers also bought beer. Suppose that the buying trend is observed over transactional data 'D'. So diaper => beer will have *confidence* 'c' and *support* 's' in D. In transactional data D, the number of transactions having diaper and beer together will

show support s in D. Confidence c in D will be indicated by transactions having either diaper or beer.

1.2 Related Work

The most profound impact in the realm of data mining world by any single algorithm has been achieved by the Apriori algorithm (proposed by Rakesh et al). Apriori algorithm approaches the problem of determining all association rules in transactional data D by sub-dividing the problem into two tasks. In the first task, all the frequent item sets are found. And in the second task, rules based on the frequent item sets that are found in the first task are generated. Apriori algorithm results in having multiple scans of the whole data set to find relationships between attribute values. This endeavor requires a lot of computational power.

In granular computing method, attribute values of a data set are represented by 1's and 0's in a bit map. Presence of a certain value is represented by 1 while the absence of that value is represented by 0. On parallel thoughts, 1 and 0 can be viewed as True and False. Computational power can be greatly reduced when using bit maps. In this way, using bit maps that need higher computational complexity, results are achieved with lesser computational power. From our earlier example of transactional data from a store such as Safeway, it will be a daunting task to find association rules using Apriori algorithm. Hypothetically, a set of transactional data having 100,000 attribute values and 1,000,000 rows of data will require 2^100,000 scans to find all possible combinations resulting in frequent item sets.

CHAPTER 2

REQUIREMENTS

2.1 Project Scope

Exploring the fundamental idea of having more efficient data mining techniques influenced by granular method entails two phases. First phase provides an avenue for storing data utilizing bit properties and requires a data warehouse that is built from ground up based on bit maps. Second phase encompasses the use of bit properties to find meaningful information in a faster manner.

This project covers the first phase by presenting a model for a bit map based data warehouse. The architecture and properties of a bit map based data warehouse are studied. A data mining centric data warehouse is constructed as part of the first phase in this project to facilitate the testing of bit association rules.

The second phase of exploring the fundamental idea of having a highly efficient data mining approach is implemented. Support for this implementation is demonstrated by providing the results from the bit association rules. Performance of the general method of obtaining association rules is compared with the bit algorithm for finding the same association rules.

CHAPTER 3

DESIGN

3.1 Architecture

3.1.1 Upload Initial Data

Data used in this project is initially present in a raw format as a text file (datasort.txt). Prior to uploading this data, a table INITIAL_DATA_TABLE (refer to Create Table SQL in Appendices) and a corresponding sequence are created in Oracle (refer to Create Sequence SQL in Appendices).

The data set used in this work has 136 attributes with 150,418 rows.

150,418 rows ↕

V0	**V1**	**V2**	**.**	**.**	**V133**	**V136**	**V137**
0	0	0	.	.	0	0	0
0	0	0	.	.	0	0	0
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
0	1	21	.	.	2	0	0
9	35	0	.	.	0	0	0
9	9	999	.	.	0	0	0

Fig. 1: INITIAL_DATA_TABLE

There are 137 columns representing 136 attributes in INITIAL_DATA_TABLE plus a column ‘ROWNO’ for keeping track of the rows. INITIAL_DATA_TABLE has 150,418 rows with each row having varying values. Each of the attribute column in INITIAL_DATA_TABLE has different number of distinct values.

3.1.2 Bit Map Table Creation

For the purposes of this project, the columns in INITIAL_DATA_TABLE are filtered based on 32 distinct values in a particular column. Any column having more than 32 distinct values is left out during the data conversion into bitmaps. The data set from the INITIAL_DATA_TABLE are converted into 5 bitmaps for using granular computing approach. The need to create 5 separate bit map tables arises from the fact that Oracle 10g has a limitation of 1000 columns. One can easily see that the 1000 columns limitation could be exceeded if only 40 columns were to have 30 distinct values in INITIAL_DATA_TABLE.

The new bit map table has 1000 columns. The first column is the 'NAME_VALUE', where NAME is the column name and VALUE is the distinct value in that column. The subsequent 998 columns are 'SET0' through 'SET997'. The last column is 'ROWNO'. Each SET has the 32 values corresponding to 32 rows in a column. SET0 will have the first 32 rows while SET1 will have the next 32 rows. 998 SETs will cover 31936 (998*32) rows from INITIAL_DATA_TABLE.

Each bit map table follows the scheme depicted in Fig. 2. NAME_VALUE has the format of column followed by the distinct value of that column. 'K' in Fig. 2 represents the column while 'i' through 'j' represents the distinct values found in column K.

NAME_VALUE	SET0	SET1	.	.	.	SET997	ROWNO
vK_i							
.							
vK_j							
vL_i							
.							
vL_j							

Fig. 2: Bit Map Table

The first 32 values (rows) in column V0 are taken and put in SET0. The next 32 values are put in SET1. Values in a SET are represented using 1s' and 0s'.

To help understand this architecture, let's take column V0 through the bit map conversion process. In the data set, column V0 has 10 distinct values. Therefore, column V0 is retained since it fits the criteria of having less than 32 distinct values. The distinct values for column V0 are: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Following the NAME_VALUE scheme (NAME => Column Name and VALUE => distinct value in the column) gives: v0_0, v0_1, v0_2, v0_3, v0_4, v0_5, v0_6, v0_7, v0_8, and v0_9. From an earlier description of the bit map table, the presence of that value in that particular position is translated to 1 (True). In the first constructed bit map table, v0_0 has 11111111111111111111111111111100. Looking at this bit representation, the first thirty 1 (True) bits denotes the occurrence of value '0' in column V0 and the 0 (False) $31^{st}/32^{nd}$ bits means that the value '0' is not present. In other words, column V0 in INITIAL_DATA_TABLE has the value '0' in its first 30 rows and some value other than '0' in rows 31 and 32. Fig. 3 shows the actual values in INITIAL_DATA_TABLE.

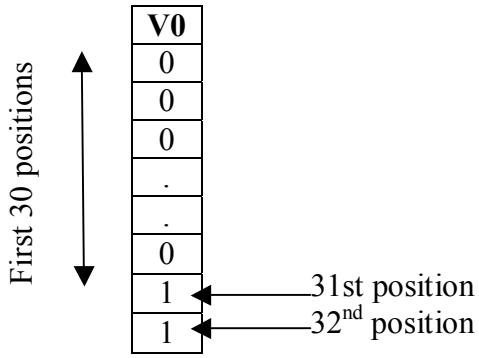

Fig. 3: Column V0 in INITIAL_DATA_TABLE

An interesting observation in this process is that if any of the distinct values hold true for a particular position (row), values for the remaining distinct values would be false automatically. Column V0 in INITIAL_DATA_TABLE has 10 distinct values (0 – 9). This means that any row from INITIAL_DATA_TABLE for column V0 will only have values from within 0 through 9.

Visualizing this information in a tabular format where the NAME_VALUE pair is mapped horizontally would be more helpful to further understand this concept. In the table below, the NAME_VALUE pair is represented in the first row by v0_0, v0_1, v0_2, v0_3, v0_4, v0_5, v0_6, v0_7, v0_8, and v0_9. Subsequent rows in the table below indicate the status of the rows in INITIAL_DATA_TABLE. First row in INITIAL_DATA_TABLE has the value of 0 in column V0. This row is represented in the table below by putting a 1 (True) under v0_0, and the remaining distinct values as 0 (False). The technique of bitwise logical operation (AND) is utilized in this project to enable efficient association rule mining.

v0_0	v0_1	v0_2	v0_3	v0_4	v0_5	V0_6	v0_7	v0_8	v0_9
1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0

Fig. 4: Property obtained from Distinct Values

CHAPTER 4

IMPLEMENTATION

4.1 Implementation issues

Challenges faced in the implementation of this endeavor can be broadly classified into two categories: issues encountered during the construction of the data warehouse and issues encountered during the implementation of algorithms for finding association rules.

4.1.1 Data warehouse construction issues

The primary implementation issue under this category was the absorption of the bit map concept and determining a way to implement this idea.

A major hurdle that influenced the implementation architecture of the data warehouse was Oracle 10g's thousand columns limitation. Existing values in a regular table are taken and represented in terms of bits for the granular approach. This results in more bit map tables depending on the number of distinct values in each column and the number of rows (representing transactional data) in a given table. In this project, a general bit map table consisting of 1000 columns (maximizing Oracle 10g's in-built feature) can represent 31936 rows from a table.

4.1.2 Association rule algorithm implementation issues

When implementing the computation of 1-itemset support, "Out of memory on Java Heap" errors were encountered. The 'BitmapRow' object that contains information related to a row in the bitmap table was causing the memory error. This object has a String array that is responsible for storing the counts of all values before converting to

binary format. The size of this String array object is larger than 150,000 as each row has over 150,000 values. Running the data file of size ‘n’ columns by ‘m’ distinct values results in n*m BitmapRow objects. With each BitmapRow object containing such a large-sized String array, space on the Java runtime heap is quickly exhausted with few iterations, causing out of memory errors.

This issue was handled by approaching the storage of the original value string from the bitmap into one single String value instead. The revised strategy uses getOrgString() method to break up into each individual value only prior to performing a bitwise ‘AND’ operation. The original count obtained from the BitmapRow objects are stored using String and each individual value is only processed when necessary. This methodology relieves excessive heap space utilization tremendously as each individual count is computed in a temporary String array instead of being stored with each row object.

4.2 Algorithms

Algorithms to generate association rules in the form of frequent item sets are implemented in two approaches for studies in this project. The two types of algorithms employed during the scope of this work are: classic algorithm to generate the association rules and algorithm utilizing bit map properties.

No major implementation issues were encountered with the classic algorithm for generating association rules. The classic algorithm was implemented using SQL in conjunction with JAVA.

4.2.1 Generating frequent 1-itemset(s) using the classic algorithm

The distinct values in a column are obtained using the SQL statement:

"SELECT COUNT (DISTINCT v" + f + ") FROM initial_data_table"

where 'f' is the column number ranging from 0 to 137. For this project, if a column has more than 32 distinct values, that column is not considered in the frequent itemset generation.

Support for each of the distinct values in each of the columns in the table is calculated by the SQL statement:

SELECT COUNT (*) FROM initial_data_table WHERE v"f"="+ count ""

where 'count' is the distinct value in the table. Support for each distinct value in a 'valid' (meeting the criteria of having at most 32 distinct values) column is calculated by implementing the support formula for a single item set:

support = total distinct values / number of transactions

A sample table is given below as an example:

V0	**V1**	**V2**	**V3**	**V4**	**V5**	**V6**
1	1	0	1	1	4	0
2	2	0	2	2	5	0
7	3	0	7	2	7	0
9	1	0	2	1	9	0
.	.	.	.	.	.	.
1	2	0	0	3	1	1

Fig. 5: Sample Table

Column V0 through V6 from Fig. 5 are queried to determine the number of distinct values in that particular column. In this sample, columns V2 and V6 are not considered during frequent itemset generation because they exceed the threshold of 32 distinct values.

Column	V0	V1	V2	V3	V4	V5	V6
Distinct Values	10	8	142	10	10	10	61

Fig. 6: Distinct values for columns from sample table

Considering only the first 4 rows from the sample table in Fig. 5, the following support values are obtained for frequent 1-itemset:

Column/Distinct	v0_1	v0_2	v0_7	V0_9
Support %	25	25	25	25

Fig. 7: Support for column V0's distinct values

Column/Distinct	v1_1	v1_2	v1_3
Support %	50	25	25

Fig. 8: Support for column V1's distinct values

Column/Distinct	v3_1	v3_2
Support %	50	50

Fig. 9: Support for column V3's distinct values

Column/Distinct	V4_1	V4_2
Support %	50	50

Fig. 10: Support for column V4's distinct values

Column/Distinct	v5_4	v5_5	v5_7	v5_9
Support %	25	25	25	25

Fig. 11: Support for column V5's distinct values

4.2.2 Generating frequent 2-itemset(s) using the classic algorithm

For generating frequent 2-itemsets, the algorithm checks the distinct values for column V0. If the column has less than 33 distinct values, it becomes the first entity against which other subsequent distinct values of other columns are compared. The first column V0 has 10 distinct values, and column V1 has 8 distinct values. Both columns pass the threshold validity check and the support level is computed between these two columns. In the next column set comparison, V0 and V2, since V2 has 142 distinct values, therefore this column would not be considered in any future calculations. The algorithm continues to consider the next column as the second entity with V0 until it reached the last column in the table. The next set of iteration is performed against the next column number or V1 as the first entity. Proceeding in this fashion, this approach computes the support level for all distinct value pairs for all qualifying columns.

The common occurrences of distinct values in two columns are obtained by using the following SQL statement:

```
SELECT COUNT (*) FROM initial_data_table
WHERE V"first column"=distinct value AND V"second column"= distinct value
```

The detailed algorithm to generate all frequent 2-itemset can be abstracted in terms of two loops where the frequent 2-itemset will consists of a first entity and a second entity with the pair under consideration:

Outer loop

- Increments the column number of first entity till the last column is reached in the table

Inner loop

- Starts with initial iteration where column number of second entity equals to column number of first entity plus one
- Increments the column number of second entity after each iteration
- Generates support for each distinct value in first entity and second entity

Using the same sample table from frequent 1-itemset, all frequent 2-itemset obtained are:

Column/Distinct	**Support %**
v0_1, v1_1	25
v0_1, v3_1	25
v0_1, v4_1	25
v0_2, v1_2	25
v0_2, v3_2	25
v0_2, v4_2	25
v0_7, v5_7	25
v0_9, v5_9	25
v1_1, v3_1	25
v1_1, v4_1	50
v1_2, v3_2	25
v1_2, v4_2	25
v3_1, v4_1	25
v3_2, v4_2	25

Fig. 12: Support for all frequent 2-itemset

4.2.3 Generating frequent 1-itemset(s) using the bit algorithm

Bit algorithm uses the file ‘disk.txt’ for generating the frequent itemsets. This flat file is obtained by concatenating the 5 bit map tables previously constructed as described in section 3.1.2 (Bit Map Table Creation).

Consider the first line from disk.txt as a sample structure:

16777216 4294967292 0 0 0 . . .

These decimal representations are converted to binary for obtaining the column and distinct value information. Converting the header value ‘16777216’ to binary gives ‘100000000000000000000000’.

This binary representation is interpreted as –

0000000000000000 – the lowest sixteen bits represent the distinct value

00000000 – the next eight bits represent the column number

1 – the highest bit represents the disk number.

Thus, the value ‘16777216’ or ‘100000000000000000000000’ in binary, translates to V0_0 (column V0, distinct value 0).

The next value after ‘16777216’ is ‘4294967292’, which represents the presence or absence of distinct value ‘0’ at that particular position (first 32 rows in initial_data_table). Converting ‘4294967292’ to binary gives ‘11111111111111111111111111111100’. This indicates that in the first 30 rows of the original table (INITIAL_DATA_TABLE), ‘0’ is present followed by some other value on the thirty-first and thirty-second rows.

For generating frequent 1-itemsets, all the non-zero values are processed to get the total (presence or true) bit count. To calculate the associated support level, the following formula is used:

support = total bit count / number of transactions

4.2.4 Generating frequent 2-itemset(s) using the bit algorithm

In generating frequent 2-itemsets, the Apriori algorithm is utilized in the implementation. This approach reduces the total time taken to produce the output by 60% compared to the original implementation. Instead of simply computing for each possible combination of itemsets, validation of support requirement is performed prior to computation.

Suppose itemsets are calculated for support level of 30. The first entity in the comparison is checked against the specified support level. If the first entity does not meet the support level, subsequent computation of any possible permutations with the first entity is skipped. In the case that the first entity is at least the specified support level, the second entity is validated to ensure that it also fulfills the support level before the computation of common bits takes place.

Let's demonstrate this strategy with an example:

Recall the sample data from Fig. 5. In finding 2-itemsets that have at least 30% support level, the algorithm first considers the entity v0_1. Since v0_1 has a support of 25%, it fails the check against the desired support level and the remaining computations (v0_1, v1_1), (v0_1, v2_1), …, (v0_1…v137_1) are skipped. It then considers the next entity

v1_1 and checks against the support level. It validates that v1_1 with support of 50% meets the minimum support and the support of the second entity is checked. It finds that the support for v2_1 does in fact fulfill the specified support of 30% so the algorithm would process the itemset (v1_1, v2_1). The original bitmap strings stored in the v1_1 and v2_1 objects are broken up into String arrays, converted back to binary and performed bitwise AND operation to compute the count of common occurrences of bits in both columns. The total count is finally used to return the support level of this itemset. The algorithm continues in the same manner in computing for (v1_1, v3_1), …, (v1_1, v137_1) until all possible 2-itemset combinations have been considered.

The bit algorithms for generating frequent itemsets were implemented to give the same results as the classic algorithm.

CHAPTER 5

DEPLOYMENT

5.1 Performance Analysis

Computer utilized in this work had the following configurations:

- 2.16Mhz Intel Dual Core CPU
- 2GB PC2-5300 DDR2 SDRAM
- Windows XP Professional SP2

The original data that was uploaded in Oracle resulted in 150,418 rows and 136 attribute values. This data formed the base from which algorithms in this project obtain frequent itemsets.

Data	Rows	Attribute Values (Columns)
datasort	150,418	136

Fig. 13: Data tested

Association rules using the classic method and the granular computing method were used to generate frequent itemsets from the same data. Time was accounted for both the results.

	Frequent 1-itemset	Frequent 2-itemset
Classic algorithm	81 seconds	5331 seconds
Bit based algorithm	2 seconds	193 seconds

Fig. 14: Time comparison between algorithms

The classic method used SQL for getting the relevant information during its computation of frequent itemsets. An approach of having the data values from the

INITIAL_DATA_TABLE exported to a flat file and then using the classical method to obtain frequent itemsets was also examined. Time taken to obtain frequent itemsets from the flat file are not mentioned in this report as it was significantly higher than the classic algorithm using SQL and would not be meaningful to compare against the granular method.

5.2 Performance Data

Part of the data obtained from generating frequent itemsets is included in this section.

5.2.1 Data from frequent 1-itemset generation

Frequent 1-itemset using classic method:

```
v0_0    Support: 0
v0_1    Support: 27
v0_2    Support: 3
v0_3    Support: 4
v0_4    Support: 3
v0_5    Support: 8
v0_6    Support: 3
v0_7    Support: 38
v0_8    Support: 9
v0_9    Support: 5
v1_0    Support: 0
v1_1    Support: 68
v1_2    Support: 32
v1_3    Support: 0
v1_4    Support: 0
v1_5    Support: 0
v1_6    Support: 0
v1_9    Support: 1
.
.
v44_0   Support: 80
v44_1   Support: 8
v44_2   Support: 7
v44_3   Support: 4
v44_4   Support: 0
v44_5   Support: 0
v44_6   Support: 0
v44_9   Support: 0
.
.
****Time elapsed: 81
seconds
```

Frequent 1-itemset using granular method:

```
(V0_0)  Support: 0
(V0_1)  Support: 27
(V0_2)  Support: 3
(V0_3)  Support: 4
(V0_4)  Support: 3
(V0_5)  Support: 8
(V0_6)  Support: 3
(V0_7)  Support: 38
(V0_8)  Support: 9
(V0_9)  Support: 5
(V1_0)  Support: 0
(V1_1)  Support: 68
(V1_2)  Support: 32
(V1_3)  Support: 0
(V1_4)  Support: 0
(V1_5)  Support: 0
(V1_6)  Support: 0
(V1_9)  Support: 1
.
.
(V44_0)  Support: 80
(V44_1)  Support: 8
(V44_2)  Support: 7
(V44_3)  Support: 4
(V44_4)  Support: 0
(V44_5)  Support: 0
(V44_6)  Support: 0
(V44_9)  Support: 0
.
.
****Time elapsed: 2 seconds
```

5.2.2 Data from frequent 2-itemset generation

Note the ordering scheme of the following output varies for the two approaches although the two computed values are the same.

Frequent 2-itemset using classic method:

```
(v0_0, v1_0)    Support: 0
(v0_1, v1_1)    Support: 18
(v0_2, v1_2)    Support: 1
(v0_3, v1_3)    Support: 0
(v0_4, v1_4)    Support: 0
(v0_5, v1_5)    Support: 0
(v0_6, v1_6)    Support: 0
(v0_7, v1_7)    Support: 0
(v0_8, v1_8)    Support: 0
(v0_9, v1_9)    Support: 0
(v0_0, v3_0)    Support: 0
(v0_1, v3_1)    Support: 6
(v0_2, v3_2)    Support: 0
(v0_3, v3_3)    Support: 0
(v0_4, v3_4)    Support: 0
(v0_5, v3_5)    Support: 0
(v0_6, v3_6)    Support: 0
(v0_7, v3_7)    Support: 0
(v0_8, v3_8)    Support: 0
(v0_9, v3_9)    Support: 1
.
.
(v70_2, v111_2)    Support: 91
(v70_3, v111_3)    Support: 1
.
.

(v70_2, v116_2)    Support: 88
(v70_3, v116_3)    Support: 0

****Time elapsed: 5331 seconds
```

Frequent 2-itemset using granular method:

```
(V0_0, V1_0)  Support: 0
(V0_0, V3_0)  Support: 0
(V0_0, V4_0)  Support: 0
.
.
(V0_1, V1_1)  Support: 18
(V0_1, V3_1)  Support: 6
(V0_1, V4_1)  Support: 6
.
.
(V127_8, V133_8)  Support: 0
(V127_8, V136_8)  Support: 0
(V133_8, V136_8)  Support: 0
(V0_9, V1_9)  Support: 0
(V0_9, V3_9)  Support: 1
(V0_9, V4_9)  Support: 3
(V0_9, V5_9)  Support: 4
(V0_9, V14_9)  Support: 0
(V0_9, V15_9)  Support: 0
(V0_9, V16_9)  Support: 0
.
.
(V70_2, V111_2)  Support: 91
(V70_2, V112_2)  Support: 95
(V70_2, V113_2)  Support: 94
(V70_2, V114_2)  Support: 96
(V70_2, V115_2)  Support: 96
(V70_2, V116_2)  Support: 88
(V70_2, V117_2)  Support: 0
.
.
.
****Time elapsed: 193 seconds
```

CHAPTER 6

CONCLUSION

6.1 Project Achievements

None of the current compact methods of association mining utilize bitmaps in their implementation. Significant achievement of this project was to implement current compact methods of association mining by application of granular computing to reduce the complexity of dealing with a high number of associations.

A bit map based data warehouse was constructed to lay the basic foundation for application of granular computing methods and obtaining association rules based on these methods. The performances between the classic method of obtaining association rules was compared with the bit based method of obtaining association rules. One of the primary purposes of this work was served by demonstrating time efficiency of granular computing methods.

6.2 Future work

Future work in this subject area can be directed towards two aspects: size of the data that needs to be analyzed and algorithms that are more efficient than the classical Apriori algorithm.

Data size is an important factor of data mining that should not be overlooked. The data used in this project consisted of approximately 150,000 rows with 137 attribute values. Time taken to obtain association rules using bit based algorithm was considerably less than the time taken to obtain the same set of association rules using the classic algorithm.

Suppose if a data of 5 attribute values and 5 rows was taken, would there be a significant time difference in the performance of granular method and classic method? No. On the contrary, as the data size increases, the need for efficient methods to find association rules efficiently becomes evident. Future direction can test these algorithms on larger data sets so that comparisons can be made effectively at that level as well.

Compact representations of association rules are one of the major efforts in reducing complexity of finding associations. Various methods have been reported, such as disjunction-free generators [7], constraints based mining [12], and mining closed itemsets (CLOSET, Charm) [13, 14]. Future work can test other algorithms and provide comparative data on those algorithms.

BIBLIOGRAPHY

[1] R. Agarwal, T. Imielinski, and A. N. Swami. *Mining Association Rules between Sets of Items in Large Databases.* In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., 1993, pp. 207-216.

[2] W. Chu and T. Y. Lin, *Foundation and Advances in Data Mining*, Springer, 2005.

[3] F. Bonchi, and C. Lucchese. *On Closed Constrained Frequent Pattern Mining.* In Proceedings of the Fourth IEEE International Conference on Data Mining, ICDM 2004, Brighton, UK, November 1-4, 2004, pp. 1-8.

[4] M. H. Dunham, *Data Mining: Introductory and Advanced Topics* Prentice Hall, 2002**.**

[5] H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database Systems: The Complete Book,* Prentice Hall. 2002.

[6] J. Hipp, U. Guntzer, and G. Nakhaezadeh. *Algorithms for Association Rule Mining-A general Survey and Comparison.* In SIGKDD Explorations, Volume 2, Issue 1, July 2000, pp. 58-64.

[7] M. Kryszkiewicz. *Concise Representation of Frequent Patterns based on Disjunction-free Generators.* In Proceedings of The 2001 IEEE International Conference on Data Mining, San Jose, California, November 29 – December 2, 2001, pp. 305-312.

[8] T. Y. Lin. *Data Mining and Machine Oriented Modeling: A Granular Computing Approach.* Journal of Applied Intelligence, Kluwer, Volume 13, No. 2, September/October, 2000, pp. 113-124.

[9] T. Y. Lin, Y. Y. Yao, and L. Zadeh (eds.). *Data Mining, Rough Sets and Granular Computing.* New York: Physica-Verlag. 2002.

[10] T. Y. Lin. *Attribute (Feature) Completion- The Theory of Attributes from Data Mining Prospect.* In Proceedings of IEEE International Conference on Data Mining, Maebashi, Japan, December 9-12, 2002, pp. 282-289.

[11] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. *Discovering frequent closed itemsets for association rules.* In Proceedings of the 7th International Conference Database Theory (ICDT'99), Jerusalem, Israel, January 1999, pp. 398-416.

[12] J. Pei, J, Han, and L. V. S. Lakshmanan. *Mining frequent item sets with convertible constraints.* In Proceedings International Conference on Data Engineering (ICDE'01), Heidelberg, Germany, April 2-6, 2001, pp. 433-442.

[13] J. Pei, J. Han, and R. Mao. *CLOSET: An efficient algorithm for mining frequent closed itemsets* . In ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Dallas, Texas, 2000, pp. 11-20.

[14] M. J. Zaki and C. J. Hsiao. *CHARM: An efficient algorithm for closed itemsets mining*. In 2nd SIAM International Conference on Data Mining, Arlington, Virginia, April 11-13, 2002, pp. 1-17.

APPENDIX A

JAVADOC APPENDIX

Package dw

Class Summary	
BitmapDM	BitmapDM class containing data mining operations for handling bitmap data
ColumnDM	ColumnDM class containing data mining operations for handling raw data
DataMining	DataMining class containing data mining operations for generating frequent 1-itemset using classical algorithm
DataMining2	DataMining2 Class containing data mining operations for generating frequent 2-itemset using classical algorithm
DB2File	DB2File class writes data retrieved from database onto a file
Item	Item class containing information of database item
ItemComparator	ItemComparator class which contains logic to compare Items
ItemSet	ItemSet class storing row items for support
ItemSetComparator	ItemSetComparator class which contains logic to compare ItemSet objects
Row	Row class to store information of a row in the bitmap.
RowComparator	RowComparator class which contains logic to compare Row object
RowTotal	RowTotal class that stores the total for a column
Utility	Utility class that contains common utility functions

---

dw
Class BitmapDM
java.lang.Object
  └ **dw.BitmapDM**

---

public class **BitmapDM**
extends java.lang.Object
BitmapDM class containing data mining operations for handling bitmap data

Field Summary	
private java.util.ArrayList	**bmRowList** List of Row objects for each Vi_n, Vj_n...
private RowComparator	**comp** RowComparator for sorting rows
private java.util.Date	**endTime** Timer stopper for each operation
private java.util.ArrayList	**itemSet2List** List of 2-itemSets under consideration
private java.util.ArrayList	**itemSetList** List of 1-itemSets under consideration
private int	**NUM_OF_ROWS** Number of rows in bitmap
private java.util.ArrayList	**rowTotalList** List of RowTotal objects for Total count of each Vi_n
private ItemSetComparator	**setComp** ItemSetComparator for sorting item sets
private java.util.Date	**startTime** Timer starter for each operation

Constructor Summary
BitmapDM() Constructor that initializes all member variables

Method Summary	
private void	**addRow**(Row row) Helper method to add Row to the ArrayList of Rows using binary search
private java.lang.String	**decodeHeader**(java.lang.Long lcode) Helper method that decode the header value of the row Note: bit [0] corresponds to disk number bit [1-9] corresponds to binary value of column number bit [9-25] corresponds to binary value of distinct value
void	**generate1_itemset**(java.lang.String filename) Generates all 1-itemsets from the input bitmap file
void	**generate2_itemset**(int minSup) Generates 2-itemsets that meets the minimum support level Operation is based on 1-itemset previously computed

java.util.ArrayList	**getItemSet1List**() Returns ArrayList of 1-itemsets
java.util.ArrayList	**getItemSet2List**() Returns ArrayList of 2-itemsets
int	**getSupport**(Row row) Getting support level for the single row
private int	**getTotalCount**(int col) Tracking total count of a given column
private boolean	**isMinSupport**(Row row, int min) Helper method to check if row meets minimum support
void	**printItemSetList**(java.lang.String file, java.util.ArrayList itemSetList) Prints list of n-itemsets to a file
void	**printRowList**(java.lang.String file) Prints row list to a file
private java.util.ArrayList	**setPrevSupport**(java.util.ArrayList itemsets, int rowTotal, int start, int end) Helper method for generate1_itemset() Sets support for previous rows in the list

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

bmRowList
private java.util.ArrayList **bmRowList**
List of Row objects for each Vi_n, Vj_n...

---

itemSetList
private java.util.ArrayList **itemSetList**
List of 1-itemSets under consideration

---

itemSet2List
private java.util.ArrayList **itemSet2List**
List of 2-itemSets under consideration

---

rowTotalList
private java.util.ArrayList **rowTotalList**
List of RowTotal objects for Total count of each Vi_n

---

comp
private RowComparator **comp**
RowComparator for sorting rows

---

setComp
private ItemSetComparator **setComp**
ItemSetComparator for sorting item sets

---

startTime
private java.util.Date **startTime**
Timer starter for each operation

---

endTime
private java.util.Date **endTime**
Timer stopper for each operation

NUM_OF_ROWS
private final int **NUM_OF_ROWS**
Number of rows in bitmap
See Also:
Constant Field Values

Constructor Detail

BitmapDM
public **BitmapDM**()
Constructor that initializes all member variables

Method Detail

generate1_itemset
public void **generate1_itemset**(java.lang.String filename)
Generates all 1-itemsets from the input bitmap file
Parameters:
filename - the name of bitmap file to be processed

---

setPrevSupport
private java.util.ArrayList **setPrevSupport**(java.util.ArrayList itemsets,
int rowTotal,
int start,
int end)
Helper method for generate1_itemset() Sets support for previous rows in the list
Parameters:
itemsets - ArrayList of itemsets
rowTotal - total number of rows
start - start index
end - end index

---

generate2_itemset
public void **generate2_itemset**(int minSup)
Generates 2-itemsets that meets the minimum support level Operation is based on 1-itemset previously computed
Parameters:
minSup - minimum support level to consider

---

isMinSupport
private boolean **isMinSupport**(Row row,
int min)
Helper method to check if row meets minimum support
Parameters:
row - Row object to consider
min - int that sets the minimum support
Returns:
true if row is at least the minimum support false otherwise

---

addRow
private void **addRow**(Row row)
Helper method to add Row to the ArrayList of Rows using binary search
Parameters:
row - Row object to be added

---

decodeHeader
private java.lang.String **decodeHeader**(java.lang.Long lcode)
Helper method that decode the header value of the row Note: bit [0] corresponds to disk number bit [1-9] corresponds to binary value of column number bit [9-25] corresponds to binary value of distinct value
Parameters:
lcode - the Long value to be decoded
Returns:
String representing the header of the row

---

getSupport
public int **getSupport**(Row row)
Getting support level for the single row
Parameters:
row - Row to calculate support for
Returns:
integer of support percentage * 100

---

getTotalCount
private int **getTotalCount**(int col)
Tracking total count of a given column
Parameters:
col - int of the column number to consider

Returns:
integer of total count

---

printRowList
public void **printRowList**(java.lang.String file)
Prints row list to a file
Parameters:
file - String of file name to print rows to

---

printItemSetList
public void **printItemSetList**(java.lang.String file,
java.util.ArrayList itemSetList)
Prints list of n-itemsets to a file
Parameters:
file - String of file name to print rows to
itemSetList - ArrayList of item-sets to be printed

---

getItemSet1List
public java.util.ArrayList **getItemSet1List**()
Returns ArrayList of 1-itemsets
Returns:
ArrayList of 1-itemsets

---

getItemSet2List
public java.util.ArrayList **getItemSet2List**()
Returns ArrayList of 2-itemsets
Returns:
ArrayList of 2-itemsets

---

dw
Class ColumnDM
java.lang.Object
└ **dw.ColumnDM**

---

public class **ColumnDM**
extends java.lang.Object
ColumnDM class containing data mining operations for handling raw data

Field Summary	
private java.util.ArrayList	**colSet** List of items for column
private ItemComparator	**comp** ItemComparator that carries logic to perform Item comparison in insertion
private java.io.PrintWriter	**out** PrintWriter to print result to

Constructor Summary
ColumnDM() Constructor that initializes all member variables

Method Summary	
private void	**addItem**(Item item) Adds an Item to the list of column items
void	**calcColData**(int col) Calculates column data
void	**clearSet**() Clear the list of column set
void	**closePrint**() Closes the opened printwriter
void	**extractColumn**(int col) Extracts column from initial_data_table, processes each value and creates an Item
int	**getSupport**(Item item) Returns the support level
int	**getTotalCount**() Returns the total count for the column set
private boolean	**isQualifiedColumn**(int col) Determines if a column is qualified to be considered
void	**printItemSet**(int min) Prints the set of items in the column that meets the given minimum support level
void	**printSet**(int col) Prints the set of column items for the given column number

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

colSet
private java.util.ArrayList **colSet**
List of items for column

---

out
private java.io.PrintWriter **out**
PrintWriter to print result to

---

comp
private ItemComparator **comp**
ItemComparator that carries logic to perform Item comparison in insertion

Constructor Detail

ColumnDM
public **ColumnDM**()
Constructor that initializes all member variables

Method Detail

extractColumn
public void **extractColumn**(int col)
Extracts column from initial_data_table, processes each value and creates an Item
Parameters:
col - column number to be extracted

---

isQualifiedColumn
private boolean **isQualifiedColumn**(int col)
Determines if a column is qualified to be considered
Parameters:
col - column number to be checked
Returns:
true if at most 32 distinct values found in given column false otherwise

---

addItem
private void **addItem**(Item item)
Adds an Item to the list of column items
Parameters:
item - Item to be added to the list

---

getSupport
public int **getSupport**(Item item)
Returns the support level

Parameters:
item - Item to get support level for
Returns:
int of support percentage * 100

---

getTotalCount
public int **getTotalCount**()
Returns the total count for the column set
Returns:
int of total count

---

printSet
public void **printSet**(int col)
throws java.io.IOException
Prints the set of column items for the given column number
Parameters:
col - int of column number
Throws:
java.io.IOException

---

closePrint
public void **closePrint**()
Closes the opened printwriter

---

clearSet
public void **clearSet**()
Clear the list of column set

---

calcColData
public void **calcColData**(int col)
throws java.io.IOException
Calculates column data
Parameters:
col - int of column number
Throws:
java.io.IOException

---

printItemSet
public void **printItemSet**(int min)
Prints the set of items in the column that meets the given minimum support level
Parameters:
min - minimum support level

dw
Class DataMining
java.lang.Object
  └ **dw.DataMining**

public class **DataMining**
extends java.lang.Object
DataMining class containing data mining operations for generating frequent 1-itemset using classical algorithm

Field Summary	
private int	**minSupport** Support level

Constructor Summary
DataMining()

Method Summary	
void	**support**() Calculates the support for 1-itemset that meets the minimum support level

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

minSupport
private int **minSupport**
Support level

Constructor Detail

DataMining
public **DataMining**()

Method Detail

support
public void **support**()
Calculates the support for 1-itemset that meets the minimum support level

dw
Class DataMining2
java.lang.Object
  └ **dw.DataMining2**

public class **DataMining2**
extends java.lang.Object
DataMining2 Class containing data mining operations for generating frequent 2-itemset using classical algorithm

Field Summary	
private int	**columnD** Distinct values for the first attribute
private int	**constantColumn** First attribute in the 2-itemset
private int	**constantDistinct** Distinct value of the first attribute in the 2-itemset
private int	**dv** Distinct values for the second attribute
private java.util.Date	**endTime** End timer for operation
private java.io.PrintWriter	**out** PrintWriter for output
private java.util.Date	**startTime** Start timer for operation
private int	**totalRows** Total rows

Constructor Summary
DataMining2() Constructor to initialize member variables

Method Summary	
int	**calcDistinct**(int f) Helper method to count distinct values
private	**numRows**()

int	Helper method for getting the total number of values in a column
void	**support2**() Calculates the support for 2-itemsets

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

totalRows
private int **totalRows**
Total rows

---

constantColumn
private int **constantColumn**
First attribute in the 2-itemset

---

constantDistinct
private int **constantDistinct**
Distinct value of the first attribute in the 2-itemset

---

columnD
private int **columnD**
Distinct values for the first attribute

---

dv
private int **dv**
Distinct values for the second attribute

---

out
private java.io.PrintWriter **out**
PrintWriter for output

---

startTime
private java.util.Date **startTime**
Start timer for operation

---

endTime
private java.util.Date **endTime**
End timer for operation

Constructor Detail

DataMining2
public **DataMining2**()
Constructor to initialize member variables

Method Detail

support2
public void **support2**()
Calculates the support for 2-itemsets

---

numRows
private int **numRows**()
Helper method for getting the total number of values in a column
Returns:
int number of values in a column

---

calcDistinct
public int **calcDistinct**(int f)
Helper method to count distinct values
Parameters:
f - int column number
Returns:
int of total number of distinct values

---

dw
Class DB2File
java.lang.Object
  └ **dw.DB2File**

---

public class **DB2File**
extends java.lang.Object
DB2File class writes data retrieved from database onto a file

---

Constructor Summary
DB2File()

Method Summary	
void	**read2File**() Reads data from tables and writes

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DB2File
public **DB2File**()

Method Detail

read2File
public void **read2File**()
Reads data from tables and writes

---

dw
Class Item
java.lang.Object
  └ **dw.Item**

---

public class **Item**
extends java.lang.Object
Item class containing information of database item

Field Summary	
private int	**count** Count of the item
private int	**value** Value of the item

Constructor Summary
Item() Default constructor that initializes an empty item
Item(int val) Constructor that initializes all member variables

Method Summary	
int	**getCount**() Getter of count
int	**getValue**() Getter of support
void	**incCount**() Increments count by 1

void	**setValue**(int i) Setter of value
java.lang.String	**toString**() Returns the String representation of the ItemSet

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

value
private int **value**
Value of the item

---

count
private int **count**
Count of the item

Constructor Detail

Item
public **Item**()
Default constructor that initializes an empty item

Item
public **Item**(int val)
Constructor that initializes all member variables
Parameters:
val - int of value to set

Method Detail

setValue
public void **setValue**(int i)
Setter of value
Parameters:
i - value of item

---

getValue
public int **getValue**()
Getter of support
Returns:
int value of item

---

incCount
public void **incCount**()
Increments count by 1

---

getCount
public int **getCount**()
Getter of count
Returns:
int count of item

---

toString
public java.lang.String **toString**()
Returns the String representation of the ItemSet
Overrides:
toString in class java.lang.Object
Returns:
String representation of the ItemSet object

---

dw
Class ItemComparator
java.lang.Object
  └ **dw.ItemComparator**
All Implemented Interfaces:
java.util.Comparator

---

public class **ItemComparator**
extends java.lang.Object
implements java.util.Comparator
ItemComparator class which contains logic to compare Items

Constructor Summary
ItemComparator() Default constructor

Method Summary	
int	**compare**(java.lang.Object o1, java.lang.Object o2) Compares 2 items and returns the lexicographical order

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface java.util.Comparator
equals

Constructor Detail

ItemComparator
public **ItemComparator**()
Default constructor

Method Detail

compare
public int **compare**(java.lang.Object o1,
java.lang.Object o2)
Compares 2 items and returns the lexicographical order
Specified by:
compare in interface java.util.Comparator
Parameters:
o1 - Item object1
o2 - Item object2
Returns:
int value of the comparison result -1 if o1 comes before o2 0 if order of o1 and o2 are equal 1 if o1 comes after o2

---

dw
Class ItemSet
java.lang.Object
└ **dw.ItemSet**

---

public class **ItemSet**
extends java.lang.Object
ItemSet class storing row items for support

Field Summary	
private int	**commonCount** Common occurrence of the itemset
private int	**NUM_OF_ROWS** Total number of rows in bitmap
private java.util.ArrayList	**rows** ArrayList of Row objects
private int	**support** Support level

Constructor Summary
ItemSet() Constructor that initializes all member variables

Method Summary	
void	**addtoItemSet**(Row r) Adds a row to the itemset
void	**calcCommon2ItemSet**() Getter of support level
void	**calcSupport**() Computation of support level of the itemset and sets the support value
int	**getCommonCount**() Getter of common count
java.util.ArrayList	**getItems**() Getter of the list of rows in ItemSet
int	**getSupport**() Getter of support level
void	**setCommonCount**(int i) Setter of common count
void	**setSupport**(int i) Setter of support
java.lang.String	**toString**() Returns the String representation of the ItemSet

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

rows
private java.util.ArrayList **rows**
ArrayList of Row objects

---

commonCount
private int **commonCount**
Common occurrence of the itemset

---

support
private int **support**
Support level

NUM_OF_ROWS
private final int **NUM_OF_ROWS**

Total number of rows in bitmap
See Also:
Constant Field Values

Constructor Detail

ItemSet
public **ItemSet**()
Constructor that initializes all member variables

Method Detail

addtoItemSet
public void **addtoItemSet**(Row r)
Adds a row to the itemset
Parameters:
r - Row to be added

---

getItems
public java.util.ArrayList **getItems**()
Getter of the list of rows in ItemSet
Returns:
ArrayList of Rows in ItemSet

---

setCommonCount
public void **setCommonCount**(int i)
Setter of common count
Parameters:
i - number of common occurrences of the value

---

setSupport
public void **setSupport**(int i)
Setter of support
Parameters:
i - support level of the itemset

---

calcSupport
public void **calcSupport**()
Computation of support level of the itemset and sets the support value

---

getCommonCount
public int **getCommonCount**()
Getter of common count
Returns:
int of common count

---

getSupport
public int **getSupport**()
Getter of support level

Returns:
int of support level

---

calcCommon2ItemSet
public void **calcCommon2ItemSet**()
Getter of support level

---

toString
public java.lang.String **toString**()
Returns the String representation of the ItemSet
Overrides:
toString in class java.lang.Object
Returns:
String representation of the ItemSet object

---

dw
Class ItemSetComparator
java.lang.Object
  └ **dw.ItemSetComparator**
All Implemented Interfaces:
java.util.Comparator

---

public class **ItemSetComparator**
extends java.lang.Object
implements java.util.Comparator
ItemSetComparator class which contains logic to compare ItemSet objects

---

Constructor Summary
ItemSetComparator()

Method Summary	
int	**compare**(java.lang.Object o1, java.lang.Object o2) Compares 2 ItemSet objects and returns the lexicographical order.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface java.util.Comparator
equals

Constructor Detail

ItemSetComparator
public **ItemSetComparator**()

Method Detail

compare
public int **compare**(java.lang.Object o1,
java.lang.Object o2)
Compares 2 ItemSet objects and returns the lexicographical order. Comparison is performed on the Row items within each ItemSet
Specified by:
compare in interface java.util.Comparator
Parameters:
o1 - Item object1
o2 - Item object2
Returns:
int value of the comparison result -1 if o1 comes before o2 0 if order of o1 and o2 are equal 1 if o1 comes after o2

---

dw
Class Row
java.lang.Object
└ **dw.Row**

---

public class **Row**
extends java.lang.Object
Row class to store information of a row in the bitmap. This is the bitmap conversion of a column of the original table

---

Field Summary	
private int	**column** Column number of bitmap row
private int	**count** Number of occurrence for the value in the column
private java.lang.String	**header** String of the decoded header value
private java.lang.String	**orgString** String containing original row of the bitmap file
private int	**value** Distinct value of bitmap row

Constructor Summary
Row() Constructor that initializes all member variables

Method Summary	
int	**getColumn**() Getter for the column number
int	**getCount**() Getter for the count
java.lang.String	**getHeader**() Getter for the header string
java.lang.String	**getString**() Getter for the original string of the row
java.lang.String[]	**getStringArr**() Returns the original bitmap row value without the header as a String array
int	**getValue**() Getter for the distinct value
void	**setColumn**(int i) Setter for the column number
void	**setCount**(int i) Setter for the count
void	**setHeader**(java.lang.Long lcode) Setter for the header string of the bitmap row
void	**setString**(java.lang.String s) Setter for the original string of the bitmap row
void	**setValue**(int i) Setter for the distinct value
java.lang.String	**toString**() Returns the String representation of the Row

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

column
private int **column**
Column number of bitmap row

---

value
private int **value**
Distinct value of bitmap row

---

count
private int **count**
Number of occurrence for the value in the column

---

header
private java.lang.String **header**
String of the decoded header value

---

orgString
private java.lang.String **orgString**
String containing original row of the bitmap file

Constructor Detail

Row
public **Row**()
Constructor that initializes all member variables

Method Detail

setColumn
public void **setColumn**(int i)
Setter for the column number
Parameters:
i - the column number of the row

---

setValue
public void **setValue**(int i)
Setter for the distinct value
Parameters:
i - the distinct value of the row

---

setCount
public void **setCount**(int i)
Setter for the count
Parameters:
i - the number of times this distinct value appeared in the column

---

setString
public void **setString**(java.lang.String s)
Setter for the original string of the bitmap row
Parameters:
s - original String of the bitmap row

---

setHeader
public void **setHeader**(java.lang.Long lcode)
Setter for the header string of the bitmap row
Parameters:
lcode - Long value to be converted into header

---

getColumn
public int **getColumn**()
Getter for the column number
Returns:
int column number

---

getValue
public int **getValue**()
Getter for the distinct value
Returns:
int distinct value

---

getCount
public int **getCount**()
Getter for the count
Returns:
int number of times this distinct value appeared in the row

---

getHeader
public java.lang.String **getHeader**()
Getter for the header string
Returns:
String of the decoded header of the bitmap row

---

getString
public java.lang.String **getString**()
Getter for the original string of the row
Returns:
String of the original text of the bitmap row

---

getStringArr
public java.lang.String[] **getStringArr**()
Returns the original bitmap row value without the header as a String array
Returns:
String[] containing each broken-down value of the original string

---

toString
public java.lang.String **toString**()
Returns the String representation of the Row
Overrides:

toString in class java.lang.Object
Returns:
String representation of the Row object

---

dw
Class RowComparator
java.lang.Object
  └ **dw.RowComparator**
All Implemented Interfaces:
java.util.Comparator

---

public class **RowComparator**
extends java.lang.Object
implements java.util.Comparator
RowComparator class which contains logic to compare Row object

---

Constructor Summary
RowComparator()

Method Summary	
int	**compare**(java.lang.Object o1, java.lang.Object o2) Compares 2 Row objects and returns the lexicographical order

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface java.util.Comparator
equals

Constructor Detail

RowComparator
public **RowComparator**()

Method Detail

compare
public int **compare**(java.lang.Object o1,
java.lang.Object o2)
Compares 2 Row objects and returns the lexicographical order

Specified by:
compare in interface java.util.Comparator
Parameters:
o1 - Item object1
o2 - Item object2
Returns:
int value of the comparison result -1 if o1 comes before o2 0 if order of o1 and o2 are equal 1 if o1 comes after o2

---

dw
Class RowTotal
java.lang.Object
└ **dw.RowTotal**

---

public class **RowTotal**
extends java.lang.Object
RowTotal class that stores the total for a column

Field Summary	
private int[]	**column** Int array that stores the column numbers
private int	**total** Total for the column

Constructor Summary
RowTotal() Constructor that initializes all member variables

Method Summary	
int[]	**getColumn**() Getter of column numbers in itemset
int	**getTotal**() Getter of total row count
void	**setColumn**(int i) Setter of column for 1-itemset
void	**setColumn**(int i, int j) Setter of column for 2-itemset

void	**setColumn**(int i, int j, int k) Setter of column for 3-itemset
void	**setTotal**(int i) Setter of row Total

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

column
private int[] **column**
Int array that stores the column numbers

---

total
private int **total**
Total for the column

Constructor Detail

RowTotal
public **RowTotal**()
Constructor that initializes all member variables

Method Detail

setColumn
public void **setColumn**(int i)
Setter of column for 1-itemset
Parameters:
i - column number in the 1-item

---

setColumn
public void **setColumn**(int i,
int j)
Setter of column for 2-itemset
Parameters:
i - first column number in the 2-item
j - second column number in the 2-item

---

setColumn
public void **setColumn**(int i,
int j,
int k)
Setter of column for 3-itemset
Parameters:
i - first column number in the 2-item
j - second column number in the 2-item

k - third column number in the 3-item

---

setTotal
public void **setTotal**(int i)
Setter of row Total
Parameters:
i - total of the row count

---

getColumn
public int[] **getColumn**()
Getter of column numbers in itemset
Returns:
array of column numbers of items

---

getTotal
public int **getTotal**()
Getter of total row count
Returns:
int of total row count

---

dw
Class Utility
java.lang.Object
  └ **dw.Utility**

---

public class **Utility**
extends java.lang.Object
Utility class that contains common utility functions

Constructor Summary
Utility()

Method Summary	
static int	**binary2int**(java.lang.String binary) Static method that convert from based-2 value String into binary value
static int	**countBits**(java.lang.String str) Helper methods that countsthe numbers of bits after converting based-2 number into binary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Utility
public **Utility**()

Method Detail

binary2int
public static int **binary2int**(java.lang.String binary)
Static method that convert from based-2 value String into binary value
Parameters:
binary - the String that holds binary value
Returns:
int of equivalent value in based-2

---

countBits
public static int **countBits**(java.lang.String str)
Helper methods that countsthe numbers of bits after converting based-2 number into binary
Parameters:
str - the String that holds the based-2 value
Returns:
number of 1 bits in the binary representation of the input string

---

APPENDIX B

UML DIAGRAMS APPENDIX

UML Class Diagrams

1. Original Data Management

   ColumnDM contains methods that reads the original data from the INITIAL_DATA_TABLE and keeps a list of Items of data retrieved. ItemComparator provides the rules of sorting the Item list. ColumnDM uses DB2File class to produce a flat file of the original data.

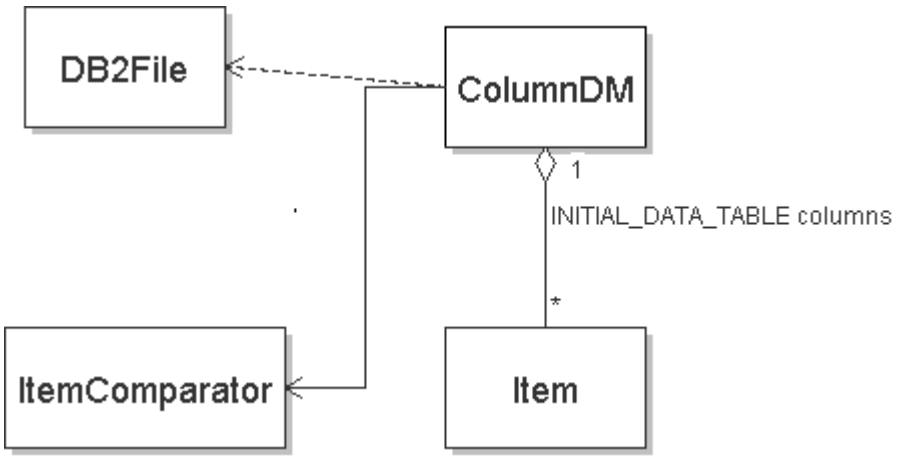

Fig. B 1: UML Class Diagram for the Original Data Management Scheme

2. Classic Algorithm

   This diagram below describes the classes involved in performing datamining operations using the classic algorithm. DataMining class provides operations to compute the support level for 1-itemset, while DataMining2 conducts computation for 2-itemsets.

Fig. B 2: UML Class Diagram for the Itemsets Generation using Classic Algorithm

3. Bitmap Datamining

BitmapDM class contains a list of Rows, RowTotals, ItemSets for datamining operations using the bitmaps approach. The list of Row objects maintains the information of each row of the bitmap, sorted by the rules defined within the RowComparator class. N ItemSet lists are kept to track the N-itemsets that meets the specified support level. Each of the itemset list is in the lexicographical order noted in the ItemSetComparator class to ensure the speediness of insertion/retrieval of items within the list. The list of RowTotal objects are used to find the support level for 1-itemsets. The Utility class contains common methods that aids in the operations for BitmapDM and Row processing.

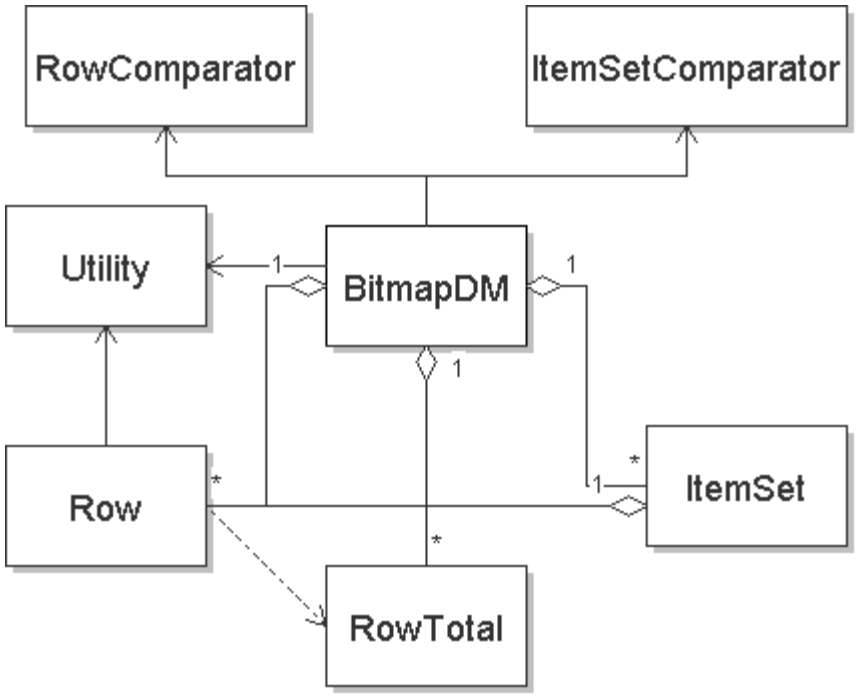

Fig. B 3: UML Class Diagram for the Itemsets Generation using Bitmap Approach

APPENDIX C

CODE LISTING

```
/**
 * BitmapDM.java
 */

package dw;
import java.io.*;
import java.sql.*;
import java.util.*;

/**
 * BitmapDM class containing data mining operations for handling bitmap data
 * @author Mien Siao
 */
public class BitmapDM
{
   /**
    *  List of Row objects for each Vi_n, Vj_n...
    */
   private ArrayList bmRowList;

   /**
    *  List of 1-itemSets under consideration
    */
   private ArrayList itemSetList;

   /**
    *  List of 2-itemSets under consideration
    */
   private ArrayList itemSet2List;

   /**
    *  List of RowTotal objects for Total count of each Vi_n
    */
   private ArrayList rowTotalList;

   /**
    *  RowComparator for sorting rows
    */
   private RowComparator comp;
```

```
/**
 *  ItemSetComparator for sorting item sets
 */
private ItemSetComparator setComp;

/**
 *  Timer starter for each operation
 */
private java.util.Date startTime;

/**
 *  Timer stopper for each operation
 */
private java.util.Date endTime;

/**
 *  Number of rows in bitmap
 */
private final int NUM_OF_ROWS = 137;

/**
 *  Constructor that initializes all member variables
 */
public BitmapDM()
{
   bmRowList = new ArrayList();
   itemSetList = new ArrayList();
   itemSet2List = new ArrayList();
   rowTotalList = new ArrayList();
   comp = new RowComparator();
   setComp = new ItemSetComparator();
}

/**
 *  Generates all 1-itemsets from the input bitmap file
 *  @param filename    the name of bitmap file to be processed
 */
public void generate1_itemset(String filename)
{
   // Start timer
   startTime = new java.util.Date();

   BufferedReader reader = null;
   try
   {
      reader = new BufferedReader(new FileReader(filename));
```

```
StringBuffer safeBuffer = new StringBuffer();
String line;
int totalBits = 0;
int rowTotCount = 0;
int col1 = 0;
int col2 = 0;
int start = 0;
int end = 0;
int i = 0;

// Process each line of the bitmap file
while( ( line = reader.readLine() ) != null )
{
   Row row = new Row();
   ItemSet set = new ItemSet();
   row.setString(line);
   totalBits = 0;  // reset total count

   StringTokenizer st = new StringTokenizer(line);

   // Store the first token of each line as header
   if (st.hasMoreTokens())
   {
      String s = st.nextToken();
      row.setHeader(new Long(s));
   }

   while (st.hasMoreTokens())
      totalBits += Utility.countBits(st.nextToken().toString());

   row.setCount(totalBits);
   addRow(row);    // adding row to member list
   set.addtoItemSet(row);
   set.setCommonCount(totalBits);
   itemSetList.add(set);

   // Reached next column, store last column count and restart
   if (row.getColumn() != col1)
   {
      RowTotal rowTot = new RowTotal();
      rowTot.setColumn(col1);
      rowTot.setTotal(rowTotCount);
      rowTotalList.add(rowTot);
      itemSetList = setPrevSupport(itemSetList, rowTotCount, start, end);
      rowTotCount = 0;
      col1 = row.getColumn();
```

```
            start = end;
          }
          end++;
          rowTotCount += totalBits;
        }
        reader.close();
        Collections.sort(itemSetList, setComp);
      }
      catch (IOException ex)
      {
        ex.printStackTrace();
      }
      endTime = new java.util.Date(); // stop timer
}

/**
 *  Helper method for generate1_itemset()
 *  Sets support for previous rows in the list
 *  @param itemsets     ArrayList of itemsets
 *  @param rowTotal     total number of rows
 *  @param start        start index
 *  @param end          end index
 */
private ArrayList setPrevSupport(ArrayList itemsets, int rowTotal, int start, int end)
{
    ArrayList list = itemsets;
    ItemSet set = new ItemSet();
    for (int i = start; i < end; i++)
    {
      set = (ItemSet)list.get(i);
      int sup = Math.round(((float)set.getCommonCount())/((float)rowTotal) * 100);
      set.setSupport(sup);
      list.set(i, set);
    }
    return list;
}

/**
 *  Generates 2-itemsets that meets the minimum support level
 *  Operation is based on 1-itemset previously computed
 *  @param minSup      minimum support level to consider
 */
public void generate2_itemset(int minSup)
{
    startTime = new java.util.Date();       // start timer
    ItemSet set = new ItemSet();
```

```
Row temp1, temp2;

// Looping through each Value: Va_i ... Va_n
for (int i=0; i < NUM_OF_ROWS; i++)
{
   // Looping through each column: Vj_n ... Vm_n
   for (int j = 0; j < NUM_OF_ROWS; j++)
   {
      temp1 = new Row();
      temp1.setColumn(j);
      temp1.setValue(i);
      int rowIndex1 = Collections.binarySearch(bmRowList, temp1, comp);
      if(rowIndex1 > -1)
      {
         temp1 = (Row)bmRowList.get(rowIndex1);
         // Looping for the 2nd row; process only if support of first row >= requested
         if (isMinSupport(temp1, minSup))
         {
            for (int k = j+1; k < NUM_OF_ROWS; k++)
            {
               temp2 = new Row();
               temp2.setColumn(k);
               temp2.setValue(i);
               int rowIndex2 = Collections.binarySearch(bmRowList, temp2, comp);
               if(rowIndex2 > -1)
               {
                  temp2 = (Row)bmRowList.get(rowIndex2);
                  // process only if support of second row >= requested
                  if (isMinSupport(temp2, minSup))
                  {
                     set = new ItemSet();
                     set.addtoItemSet(temp1);
                     set.addtoItemSet(temp2);
                     set.calcCommon2ItemSet();
                     set.calcSupport();

                     // Add to 2-itemset list only if it meet minimum support
                     if (set.getSupport() >= minSup)
                        itemSet2List.add(set);
                  }
               }
            }
         }
      }
   }
}
```

```
   endTime = new java.util.Date();     // stop timer
}

/**
 *  Helper method to check if row meets minimum support
 *  @param row  Row object to consider
 *  @param min  int that sets the minimum support
 *  @return     true if row is at least the minimum support
 *              false otherwise
 */
private boolean isMinSupport(Row row, int min)
{
   ItemSet s = new ItemSet();
   s.addtoItemSet(row);
   int index = Collections.binarySearch(itemSetList, s, setComp);
   int sup = 0;
   if(index > -1)
   {
      ItemSet temp = (ItemSet)itemSetList.get(index);
      sup = temp.getSupport();
   }
   return (sup >= min);
}

/**
 *  Helper method to add Row to the ArrayList of Rows using binary search
 *  @param row  Row object to be added
 */
private void addRow(Row row)
{
   int rowIndex = Collections.binarySearch(bmRowList, row, comp);
   if(rowIndex < 0)
   {
      bmRowList.add(row);
      Collections.sort(bmRowList, comp);
   }
}


/**
 *  Helper method that decode the header value of the row
 *  Note:   bit [0] corresponds to disk number
 *          bit [1-9] corresponds to binary value of column number
 *          bit [9-25] corresponds to binary value of distinct value
 *  @param lcode   the Long value to be decoded
 *  @return        String representing the header of the row
```

```
 */
private String decodeHeader(Long lcode)
{
   String headerStr = Long.toBinaryString(lcode.longValue());
   StringBuffer sb = new StringBuffer("V");
   int iTemp = Utility.binary2int(headerStr.substring(1,9));
   sb.append(iTemp + "_");
   iTemp = Utility.binary2int(headerStr.substring(9,25));
   sb.append(iTemp);
   return sb.toString();
}

/**
 *  Getting support level for the single row
 *  @param row      Row to calculate support for
 *  @return         integer of support percentage * 100
 */
public int getSupport(Row row)
{
   int rowTotal = getTotalCount(row.getColumn());
   return Math.round(((float)row.getCount())/((float)rowTotal) * 100);
}

/**
 *  Tracking total count of a given column
 *  @param col   int of the column number to consider
 *  @return      integer of total count
 */
private int getTotalCount(int col)
{
   RowTotal tot = null;
   for(int i = 0; i < rowTotalList.size(); i++)
   {
      tot = (RowTotal) rowTotalList.get(i);
      int temp[] = tot.getColumn();
      if(temp[0] == col)
      {
         return tot.getTotal();      // return total count when column number matched
      }
   }
   return 0;
}

/**
 *  Prints row list to a file
 *  @param file  String of file name to print rows to
```

```
 */
public void printRowList(String file)
{
   PrintWriter writer = null;
   try
   {
      writer = new PrintWriter(new FileWriter(file));
   }
   catch (IOException ex)
   {
      ex.printStackTrace();
   }

   Row row;
   for(int i = 0; i<bmRowList.size(); i++)
   {
      row = (Row)bmRowList.get(i);
      writer.print(row.toString());
      writer.print("\t  Support: " + getSupport(row));
      writer.println();
   }
   writer.close();
}

/**
 *  Prints list of n-itemsets to a file
 *  @param file         String of file name to print rows to
 *  @param itemSetList  ArrayList of item-sets to be printed
 */
public void printItemSetList(String file, ArrayList itemSetList)
{
   PrintWriter writer = null;
   try
   {
      writer = new PrintWriter(new FileWriter(file));
   }
   catch (IOException ex)
   {
      ex.printStackTrace();
   }
   ItemSet set;
   for(int i = 0; i<itemSetList.size(); i++)
   {
      set = (ItemSet)itemSetList.get(i);
      writer.print(set.toString());
      writer.println();
```

```
        }

        long sec = (endTime.getTime() - startTime.getTime());
        writer.println("****Time elapsed: " + (sec/1000) + " seconds");
        writer.close();
    }

    /**
     *  Returns ArrayList of 1-itemsets
     *  @return     ArrayList of 1-itemsets
     */
    public ArrayList getItemSet1List()
    {
        return itemSetList;
    }

    /**
     *  Returns ArrayList of 2-itemsets
     *  @return     ArrayList of 2-itemsets
     */
    public ArrayList getItemSet2List()
    {
        return itemSet2List;
    }

}
```

```
/**
 * ColumnDM.java
 */
package dw;

import java.io.*;
import java.sql.*;
import java.util.*;

/**
 * ColumnDM class containing data mining operations for handling raw data
 * @author Mien Siao
 */
public class ColumnDM
{
   /**
    * List of items for column
    */
   private ArrayList colSet;

   /**
    * PrintWriter to print result to
    */
   private PrintWriter out;

   /**
    * ItemComparator that carries logic to perform Item comparison in insertion
    */
   private ItemComparator comp;

   /**
    *  Constructor that initializes all member variables
    */
   public ColumnDM()
   {
      colSet = new ArrayList();
      comp = new ItemComparator();
      try
      {
         out = new PrintWriter(new FileWriter("outColn.txt"));
      }
      catch (IOException ex)
      {
         ex.printStackTrace();
      }
   }
```

```
   /**
    * Extracts column from initial_data_table, processes each value and creates an Item
    * @param col   column number to be extracted
    */
   public void extractColumn(int col)
   {
      try
      {
         Class.forName("oracle.jdbc.driver.OracleDriver");

         Connection conn = DriverManager.getConnection(
               "jdbc:oracle:thin:@bodog:1521:orcl","system","ronnielt");

         Statement stmt = conn.createStatement();
         String sqlStr = "select V" + col + " from initial_data_table where ROWNO <=
451315";
         ResultSet rs = stmt.executeQuery(sqlStr);

         Item item;
         while (rs.next())
         {
            item = new Item(rs.getInt(1));
            System.out.println(item.toString());
            addItem(item);
         }

         stmt.close();
         rs.close();
         conn.close();

         System.out.println("!!Number of items in colset: " + colSet.size());
      }
      catch(Exception e)
      {
         System.out.println(e);
      }
   }

   /**
    * Determines if a column is qualified to be considered
    * @param col   column number to be checked
    * @return true if at most 32 distinct values found in given column
    *         false otherwise
    */
   private boolean isQualifiedColumn(int col)
```

```
    {
        int distinctVal = 0;
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");

            Connection conn = DriverManager.getConnection(
                  "jdbc:oracle:thin:@bodog:1521:orcl","system","ronnielt");

            Statement stmt = conn.createStatement();
            String sqlStr = "select count (distinct(V" + col + ")) from initial_data_table where
ROWNO <= 451315";
            ResultSet rs = stmt.executeQuery(sqlStr);
            rs.next();
            distinctVal = rs.getInt(1);

            stmt.close();
            rs.close();
            conn.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
        return distinctVal <= 32;
    }

    /**
     * Adds an Item to the list of column items
     * @param item  Item to be added to the list
     */
    private void addItem(Item item)
    {
        int itemIndex = Collections.binarySearch(colSet, item, comp);

        if(itemIndex < 0)
        {
            System.out.println("New Item added!");
            colSet.add(item);
            Collections.sort(colSet, comp);
        }
        else
        {
            System.out.println("Dup found!");
            Item foundItem = (Item)colSet.get(itemIndex);
            foundItem.incCount();
```

```
    }
}

/**
 * Returns the support level
 * @param item  Item to get support level for
 * @return      int of support percentage * 100
 */
public int getSupport(Item item)
{
    return Math.round(((float)item.getCount())/((float)getTotalCount()) * 100);
}

/**
 * Returns the total count for the column set
 * @return      int of total count
 */
public int getTotalCount()
{
    int total = 0;
    for (Iterator it = colSet.iterator(); it.hasNext(); )
        total += ((Item)it.next()).getCount();
    return total;
}

/**
 * Prints the set of column items for the given column number
 * @param col   int of column number
 */
public void printSet(int col) throws IOException
{

    if (col >= 0)
    {
        out.println("\n\n< 1-ITEMSET FOR COLUMN V" + col + " >");
        for(int i = 0; i<colSet.size(); i++)
        {
            Item item = (Item)colSet.get(i);
            out.print(item.toString());
            out.print("  Support: " + getSupport(item));
            out.println();
        }
    }
    else
        out.println("\nSKIPPING COLUMN!!");
```

```
    }
    /**
     * Closes the opened printwriter
     */
    public void closePrint()
    {
       out.close();
    }

    /**
     * Clear the list of column set
     */
    public void clearSet()
    {
       colSet = new ArrayList();
    }

    /**
     * Calculates column data
     * @param col   int of column number
     */
    public void calcColData(int col) throws IOException
    {
       clearSet();
       extractColumn(col);
    }

    /**
     * Prints the set of items in the column that meets the given minimum support level
     * @param min   minimum support level
     */
    public void printItemSet(int min)
    {
       Item item;
       for(int i = 0; i<colSet.size(); i++)
       {
          item = (Item)colSet.get(i);
          if(getSupport(item) >= min)
          {
             out.print(item.toString());
             out.print("  Support: " + getSupport(item));
             out.println();
          }
       }
    }
}
```

```
/*
 * DataMining.java
 */
package dw;

import java.sql.*;
import java.io.*;
import java.lang.*;
import java.util.*;

/**
 * DataMining class containing data mining operations for generating frequent 1-itemset
 * using classical algorithm
 * @author Mien Siao
 */
public class DataMining
{

   /**
    * Support level
    */
   private int minSupport;

   public DataMining()
   {
      minSupport = 0;
   }

   /**
    * Calculates the support for 1-itemset that meets the minimum support level
    */
   public void support()
   {
      int totalDistinctValues = 0;    // Distinct values in a column
      int dv = 0;                     // distinct value
      float percentageValue = 0;      // Calculated the support percentage
      int countDV = 0;                // Count of distinct values
      int totalRows = 0;              // Total rows
      PrintWriter out = null;         // Printwriter for outputting result
      java.util.Date startTime = new java.util.Date();

      try
      {
         out = new PrintWriter(new FileWriter("outDataMining.txt"));
      }
      catch (IOException ex)
```

```
        {
            ex.printStackTrace();
        }
        try
        {
            //Setting connections for Oracle
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@IBM-
2A24156C376:1521:orcl","system","cs257");

            Statement stmt = conn.createStatement();
            Statement numRows = conn.createStatement();
            Statement createTableStmt = conn.createStatement();

            //Getting the total number of rows in Initial_Data_Table
            ResultSet tempRows = numRows.executeQuery("SELECT COUNT(*) FROM
initial_data_table");
            tempRows.next();

            String rows = tempRows.getString(1);

            //Converting to integer value
            totalRows = java.lang.Math.round(Integer.parseInt(rows));

            out.print("Total # of values in column is " +totalRows);
            out.print("\n");

            for(Integer f=0; f<=137; f++)
            {
                // For the case when 134th row is reached, skip over the 2 empty rows
                if(f==134)
                    f = 136;

                //Getting the total number of distinct values in column V?? in
initial_Data_Table
                ResultSet temp = numRows.executeQuery("SELECT COUNT (DISTINCT v"
+ f.toString() + ") FROM initial_data_table");
                temp.next();

                String distinctValues = temp.getString(1);

                //Converting to integer value
                totalDistinctValues = Integer.parseInt(distinctValues);

                out.println("Total distinct values in v" + f.toString() + " : " +
totalDistinctValues);
```

```
            // Process only if column qualifies the 32 maximum distinct value criteria
            if(totalDistinctValues <= 32)
            {

               ResultSet rs = stmt.executeQuery(
                     "SELECT DISTINCT(v" + f.toString() + ") FROM initial_data_table
ORDER BY v" + f.toString() + "");

               while(rs.next(  ))
               {

                  String distinctV = rs.getString(1);

                  //converting to integer value
                  dv = Integer.parseInt(distinctV);

                  ResultSet tem = numRows.executeQuery("SELECT COUNT (*) FROM
initial_data_table WHERE v" + f.toString() + "="+ dv +"");
                  tem.next();

                  String temCount = tem.getString(1);
                  countDV = java.lang.Math.round(Integer.parseInt(temCount));
                  percentageValue =
java.lang.Math.round((((float)countDV/(float)totalRows)*100.0));

                  //Checking if a distinct value in a column is greater than the support
threshold
                  if (percentageValue >= minSupport)
                  {
                     out.println("v" + f + "_"+ dv);
                     out.println("support is " + (int)percentageValue);
                  }
               }
            }
         }
         conn.close();
      }
      catch(Exception e)
      {
         out.println(e);
      }
      java.util.Date endTime = new java.util.Date();

      long sec = (endTime.getTime() - startTime.getTime());
      out.print("****Time elapsed: " + sec/1000 + " seconds");
```

```
        out.close();
    }
}
```

```
/*
 * DataMining2.java
 */

package dw;

import java.sql.*;
import java.io.*;
import java.lang.*;
import java.util.*;

/**
 * DataMining2 Class containing data mining operations for generating frequent 2-
itemset
 * using classical algorithm
 * @author Mien Siao
 */
public class DataMining2 {

   /**
    *  Total rows
    */
   private int totalRows;

   /**
    *  First attribute in the 2-itemset
    */
   private int constantColumn;

   /**
    *  Distinct value of the first attribute in the 2-itemset
    */
   private int constantDistinct;

   /**
    * Distinct values for the first attribute
    */
   private int columnD;

   /**
    *  Distinct values for the second attribute
    */
   private int dv;

   /**
    *  PrintWriter for output
```

```
 */
private PrintWriter out;

/**
 *  Start timer for operation
 */
private java.util.Date startTime;

/**
 *  End timer for operation
 */
private java.util.Date endTime;

/**
 *  Constructor to initialize member variables
 */
public DataMining2(){
   totalRows = 0;
   constantColumn = 0;
   constantDistinct = 0;
   columnD = 0;
   dv = 0;

   try {
      out = new PrintWriter(new FileWriter("outDataMining2.txt"));
   } catch (IOException ex) {
      ex.printStackTrace();
   }
}

/**
 *  Calculates the support for 2-itemsets
 */
public void support2() {

   startTime = new java.util.Date();

   // Getting total rows
   totalRows = numRows();

   // Outer loop for moving through the columns and keeping the current column as the
first attribute value
   for (Integer i=0; i<=136; i++) {
      if(i==134)
         i = 136;
      constantColumn = i;
```

```
        try{

            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@IBM-
2A24156C376:1521:orcl","system","cs257");

            Statement numRows = conn.createStatement();

            ResultSet temp2 = numRows.executeQuery("SELECT COUNT (DISTINCT v"
+ constantColumn + ") FROM initial_data_table");
            temp2.next();

            String distinctCValues = temp2.getString(1);
            columnD = Integer.parseInt(distinctCValues);

            out.print("\n");
            out.print("Total distinct values in v " + constantColumn + " : " + columnD);

            // Inner loop to move throught the columns and keep the current column as the
second attribute value
            if(columnD <= 32){

               for(int f=(constantColumn+1); f<=137; f++) {
                  if(f==134)      // to skip over 134th and 135th rows which are empty
                     f = 136;

                  calcDistinct(f);
               }

            }
            conn.close();
         }catch(Exception e) {
            out.println(e);
         }
      }

      endTime = new java.util.Date();
      long sec = (endTime.getTime() - startTime.getTime());
      out.print("\n****Time elapsed: " + sec/1000 + " seconds");

      out.close();
   }

   /**
    *  Helper method for getting the total number of values in a column
```

```
     *  @return     int number of values in a column
     */
    private int numRows() {

        try{
            //Setting connections for Oracle
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@IBM-
2A24156C376:1521:orcl","system","cs257");

            Statement stmt = conn.createStatement();
            Statement numRows = conn.createStatement();
            Statement createTableStmt = conn.createStatement();

            //Getting the total number of rows in Initial_Data_Table
            ResultSet tempRows = numRows.executeQuery("SELECT COUNT(*) FROM
initial_data_table");
            tempRows.next();

            String rows = tempRows.getString(1);

            //Converting to integer value
            totalRows = Integer.parseInt(rows);

            out.println("Total # of values in column is " + totalRows);

            conn.close();
        }catch(Exception e) {
            out.println(e);
        }
        return totalRows;
    }


     /**
     *  Helper method to count distinct values
     *  @param f    int column number
      * @return     int of total number of distinct values
     */

    public int calcDistinct(int f) {

        int totalDistinctValues = 0;    // Total distinct values in a column
        int countDV = 0;                // Count of distinct values
        float percentageValue = 0;      // Percentage of support level
```

```
      try{
         // Setting connection for Oracle
         Class.forName("oracle.jdbc.driver.OracleDriver");
         Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@IBM-
2A24156C376:1521:orcl","system","cs257");

         Statement numRows = conn.createStatement(); // Getting number of rows

         ResultSet temp = numRows.executeQuery("SELECT COUNT (DISTINCT v" + f
+ ") FROM initial_data_table");
         temp.next(); // Getting number of distinct values in the column

         String distinctValues = temp.getString(1);

         //Converting to integer value
         totalDistinctValues = Integer.parseInt(distinctValues);

         out.print("\n");
         out.print("Total distinct values in v " + f + " : " + totalDistinctValues);
         out.print("\n");
         out.print("\n");

         // Check the number of distinct values in current column

         if(totalDistinctValues <=32) {

            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(
                  "SELECT DISTINCT(v" + f + ") FROM initial_data_table ORDER BY
v" + f + "");

            // loop through the distinct values
            while(rs.next(  )) {
                String distinctV = rs.getString(1);

               //converting to integer value
               dv = Integer.parseInt(distinctV);

               Statement stmtC = conn.createStatement();
               ResultSet rsC = stmt.executeQuery(
                     "SELECT DISTINCT(v" + constantColumn + ") FROM
initial_data_table ORDER BY v" + constantColumn + "");

               while(rsC.next())
               {
                  String distinctC = rsC.getString(1);
```

```
                constantDistinct = Integer.parseInt(distinctC);

                ResultSet tem = numRows.executeQuery("SELECT COUNT (*) FROM
initial_data_table " +
                        "WHERE v" + constantColumn + "=" + constantDistinct +" AND
v"+ f +"=" + constantDistinct +"");

                tem.next();
                String temCount = tem.getString(1);
                countDV = java.lang.Math.round(Integer.parseInt(temCount));

                out.println("v" + constantColumn + "_"+ constantDistinct +" = v" + f +
"_"+ constantDistinct +"");

                percentageValue = java.lang.Math.round(((countDV/totalRows)*100));

                out.println("support is " + (int)percentageValue);
              }
            }
          }

          conn.close();
        }catch(Exception e) {
          out.println(e);
        }
        return totalDistinctValues;
    }
}
```

```
/*
 * DB2File.java
 */

package dw;

import java.io.*;
import java.sql.*;

/**
 * DB2File class writes data retrieved from database onto a file
 * @author Mien Siao
 */
public class DB2File
{

   /**
    * Reads data from tables and writes
    */
   public void read2File()
   {
      try
      {
         Class.forName("oracle.jdbc.driver.OracleDriver");

         Connection conn = DriverManager.getConnection(
               "jdbc:oracle:thin:@bodog:1521:orcl","system","ronnielt");

         Statement stmt = conn.createStatement();
         Statement numRows = conn.createStatement();

         //ResultSet rs1 = stmt.executeQuery("select * from initial_data_table where
rowno <= 451315");
         ResultSet rs1 = stmt.executeQuery("select * from initial_data_table where rowno
<= 100000");

         PrintWriter out = new PrintWriter(new FileWriter("out1.txt"));

         while (rs1.next())
         {

            for(int i=1; i<137; i++ )
               out.print(rs1.getString(i) + " ");

            out.println();
         }
```

```
            out.close();
            conn.close();

        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }

 /*   public static void main(String args[])
    {
        DB2File file = new DB2File();
        file.read2File();
    }*/
}
```

```
/**
 * Item.java
 */

package dw;

import java.util.ArrayList;

/**
 * Item class containing information of database item
 * @author Mien Siao
 */
public class Item
{
   /**
    *  Value of the item
    */
   private int value;

   /**
    *  Count of the item
    */
   private int count;


   /**
    *  Default constructor that initializes an empty item
    */
   public Item()
   {
      count = 0;
   }

   /**
    *  Constructor that initializes all member variables
    *  @param val  int of value to set
    */
   public Item(int val)
   {
      value = val;
      count = 1;
   }

   /**
```

```
    * Setter of value
    *  @param i    value of item
    */
    public void setValue(int i)
    {
       value = i;
    }

    /**
    * Getter of support
    *  @return     int value of item
    */
    public int getValue()
    {
       return value;
    }

    /**
    * Increments count by 1
    */
    public void incCount()
    {
       count++;
    }

    /**
    * Getter of count
    *  @return     int count of item
    */
    public int getCount()
    {
       return count;
    }

    /**
    *  Returns the String representation of the ItemSet
    *  @return     String representation of the ItemSet object
    */
    public String toString()
    {
       return "Value: " + value + "\t  Count: " + count;
    }
}
```

```
/**
 * ItemComparator.java
 */
package dw;

import java.util.Comparator;

/**
 * ItemComparator class which contains logic to compare Items
 * @author Mien Siao
 */
public class ItemComparator implements Comparator
{
   /**
    *  Default constructor
    */
   public ItemComparator()
   {

   }

   /**
    *  Compares 2 items and returns the lexicographical order
    *  @param o1  Item object1
    *  @param o2  Item object2
    *  @return    int value of the comparison result
    *             -1 if o1 comes before o2
    *             0 if order of o1 and o2 are equal
    *             1 if o1 comes after o2
    */
   public int compare(Object o1, Object o2)
   {
      Item i1 = (Item)o1;
      Item i2 = (Item)o2;

      if(i1.getValue() < i2.getValue())
         return -1;
      else if (i1.getValue() == i2.getValue())
         return 0;
      else
         return 1;
   }

}
```

```
/**
 * ItemSet.java
 */

package dw;

import java.util.*;

/**
 * ItemSet class storing row items for support
 * @author Mien Siao
 */
public class ItemSet
{
   /**
    * ArrayList of Row objects
    */
   private ArrayList rows;

   /**
    * Common occurrence of the itemset
    */
   private int commonCount;

   /**
    * Support level
    */
   private int support;

   /**
    * Total number of rows in bitmap
    */
   private final int NUM_OF_ROWS = 150400;

   /**
    *  Constructor that initializes all member variables
    */
   public ItemSet()
   {
      rows = new ArrayList();
      commonCount = 0;
      support = 0;
   }

   /**
```

```
 * Adds a row to the itemset
 *  @param r    Row to be added
 */
public void addtoItemSet(Row r)
{
   rows.add(r);
}

/**
 * Getter of the list of rows in ItemSet
 *  @return     ArrayList of Rows in ItemSet
 */
public ArrayList getItems()
{
   return rows;
}

/**
 * Setter of common count
 *  @param i    number of common occurrences of the value
 */
public void setCommonCount(int i)
{
   commonCount = i;
}

/**
 * Setter of support
 *  @param i    support level of the itemset
 */
public void setSupport(int i)
{
   support = i;
}

/**
 * Computation of support level of the itemset and sets the support value
 */
public void calcSupport()
{
   long sup = Math.round( (float)commonCount/(float)NUM_OF_ROWS * 100.0);
   setSupport( (int)sup);
}

/**
 * Getter of common count
```

```
     *  @return     int of common count
     */
    public int getCommonCount()
    {
       return commonCount;
    }

    /**
     * Getter of support level
     *  @return     int of support level
     */
    public int getSupport()
    {
       return support;
    }

    /**
     * Getter of support level
     */
    public void calcCommon2ItemSet()
    {
       Row r1 = (Row)rows.get(0);
       Row r2 = (Row)rows.get(1);
       String[] rString1 = r1.getStringArr();
       String[] rString2 = r2.getStringArr();
       Long int1;
       Long int2;
       int i;
       long n1, n2, res;

       // Compute using the r1 as reference if it has the least non-zero values
       if(r1.getCount() < r2.getCount())
       {
          for(int b = 0; b < rString1.length; b++)
          {
             int1 = new Long(rString1[b]);
             n1 = int1.longValue();
             if(n1 > 0)
             {
                int2 = new Long(rString2[b]);
                n2 = int2.longValue();
                res = n1 & n2;
                commonCount += Long.bitCount((long)res); // convert res to binary and add
bitcount to count.
             }
          }
```

```
        }
        else    // Compute using the r2 as reference if it has the least non-zero values
        {
           for(int b = 0; b < rString2.length; b++)
           {
              int2 = new Long(rString2[b]);
              n2 = int2.longValue();
              if(n2 > 0)
              {
                 int1 = new Long(rString1[b]);
                 n1 = int1.longValue();
                 res = n1 & n2;
                 commonCount += Long.bitCount((long)res); // convert res to binary and add
bitcount to count.
              }
           }
        }
     }

     /**
      *  Returns the String representation of the ItemSet
      *  @return     String representation of the ItemSet object
      */
     public String toString()
     {
        StringBuffer sb = new StringBuffer("(");
        int num = rows.size();
        for (int i = 0; i < num; i++)
        {
           sb.append(((Row)rows.get(i)).getHeader());
           if(i != num-1)
              sb.append(", ");
        }
        sb.append(")");

        sb.append("\t  Common count: " + commonCount);
        sb.append("\t  Support: " + support);
        return sb.toString();
     }
}
```

```
/**
 * ItemSetComparator.java
 */

package dw;

import java.util.*;

/**
 * ItemSetComparator class which contains logic to compare ItemSet objects
 * @author Mien Siao
 */
public class ItemSetComparator implements Comparator
{

   public ItemSetComparator()
   {
   }

   /**
    *  Compares 2 ItemSet objects and returns the lexicographical order.
    *  Comparison is performed on the Row items within each ItemSet
    *  @param o1  Item object1
    *  @param o2  Item object2
    *  @return    int value of the comparison result
    *           -1 if o1 comes before o2
    *           0 if order of o1 and o2 are equal
    *           1 if o1 comes after o2
    */
   public int compare(Object o1, Object o2)
   {
      ItemSet b1 = (ItemSet)o1;
      ItemSet b2 = (ItemSet)o2;

      ArrayList list1 = (b1.getItems());
      ArrayList list2 = (b2.getItems());

      int result = 0;
      RowComparator comp = new RowComparator();
      for (int i = 0; i < list1.size(); i++)
      {
         result = comp.compare((Row)list1.get(i), (Row)list2.get(i));
         if (result != 0)
            return result;
      }
      return result;}}
```

```
/**
 * Row.java
 */

package dw;

import java.util.*;

 /**
  * Row class to store information of a row in the bitmap.
  * This is the bitmap conversion of a column of the original table
  * @author Mien Siao
  */
public class Row
{
   /**
    * Column number of bitmap row
    */
   private int column;

   /**
    * Distinct value of bitmap row
    */
   private int value;

   /**
    * Number of occurrence for the value in the column
    */
   private int count;

   /**
    * String of the decoded header value
    */
   private String header;

   /**
    * String containing original row of the bitmap file
    */
   private String orgString;

   /**
    *  Constructor that initializes all member variables
    */
   public Row()
   {
```

```
    column = 0;
    value = 0;
    count = 0;
    header = new String();
    orgString = new String();
}

/**
 *  Setter for the column number
 *  @param i    the column number of the row
 */
public void setColumn(int i)
{
    column = i;
}

/**
 *  Setter for the distinct value
 *  @param i    the distinct value of the row
 */
public void setValue(int i)
{
    value = i;
}

/**
 *  Setter for the count
 *  @param i    the number of times this distinct value appeared in the column
 */
public void setCount(int i)
{
    count = i;
}

/**
 *  Setter for the original string of the bitmap row
 *  @param s    original String of the bitmap row
 */
public void setString(String s)
{
    orgString = s;
}

/**
 *  Setter for the header string of the bitmap row
 *  @param lcode    Long value to be converted into header
```

```
 */
public void setHeader(Long lcode)
{
   // bits 1-8-16 represents disk-column-distinctValue
   String headerStr = Long.toBinaryString(lcode.longValue());

   StringBuffer sb = new StringBuffer("V");
   int iTemp = Utility.binary2int(headerStr.substring(1,9));
   setColumn(iTemp);
   sb.append(iTemp + "_");
   iTemp = Utility.binary2int(headerStr.substring(9,25));
   setValue(iTemp);
   sb.append(iTemp);
   header = sb.toString();
}

/**
 *  Getter for the column number
 *  @return     int column number
 */
public int getColumn()
{
   return column;
}

/**
 *  Getter for the distinct value
 *  @return     int distinct value
 */
public int getValue()
{
   return value;
}

/**
 *  Getter for the count
 *  @return     int number of times this distinct value appeared in the row
 */
public int getCount()
{
   return count;
}

/**
 *  Getter for the header string
 *  @return     String of the decoded header of the bitmap row
```

```
     */
    public String getHeader()
    {
       return "V"+column+"_"+value;
    }

    /**
     *  Getter for the original string of the row
     *  @return     String of the original text of the bitmap row
     */
    public String getString()
    {
       return orgString;
    }

    /**
     *  Returns the original bitmap row value without the header as a String array
     *  @return     String[] containing each broken-down value of the original string
     */
    public String[] getStringArr()
    {
       String s = orgString.substring(9);
       return s.split("\\s");
    }

    /**
     *  Returns the String representation of the Row
     *  @return     String representation of the Row object
     */
    public String toString()
    {
       return "Header: " + header + "  Count: " + count;
    }
}
```

```
/**
 * RowComparator.java
 */

package dw;

import java.util.Comparator;

/**
 * RowComparator class which contains logic to compare Row object
 * @author Mien Siao
 */
public class RowComparator implements Comparator
{

   public RowComparator()
   {
   }

   /**
    *  Compares 2 Row objects and returns the lexicographical order
    *  @param o1  Item object1
    *  @param o2  Item object2
    *  @return    int value of the comparison result
    *            -1 if o1 comes before o2
    *            0 if order of o1 and o2 are equal
    *            1 if o1 comes after o2
    */
   public int compare(Object o1, Object o2)
   {
      Row b1 = (Row)o1;
      Row b2 = (Row)o2;

      if(b1.getValue() < b2.getValue())
         return -1;
      else if (b1.getValue() == b2.getValue())
      {
         if(b1.getColumn() < b2.getColumn())
            return -1;
         else if(b1.getColumn() > b2.getColumn())
            return 1;
         else
            return 0;
      }
```

```
        else
            return 1;
    }

}
```

```
/**
 * RowTotal.java
 */

package dw;

/**
 * RowTotal class that stores the total for a column
 * @author Mien Siao
 */
public class RowTotal
{
   /**
    * Int array that stores the column numbers
    */
   private int[] column;

   /**
    * Total for the column
    */
   private int total;

   /**
    *  Constructor that initializes all member variables
    */
   public RowTotal()
   {
      column = null;
      total = 0;
   }

   /**
    * Setter of column for 1-itemset
    *  @param i   column number in the 1-item
    */
   public void setColumn(int i)
   {
      column = new int[] {i};
   }

   /**
    * Setter of column for 2-itemset
    *  @param i   first column number in the 2-item
    *  @param j   second column number in the 2-item
    */
   public void setColumn(int i, int j)
```

```
    {
        column = new int[] {i, j};
    }

    /**
     * Setter of column for 3-itemset
     *  @param i   first column number in the 2-item
     *  @param j   second column number in the 2-item
     *  @param k   third column number in the 3-item
     */
    public void setColumn(int i, int j, int k)
    {
        column = new int[] {i, j, k};
    }

    /**
     * Setter of row Total
     *  @param i   total of the row count
     */
    public void setTotal(int i)
    {
        total = i;
    }

    /**
     * Getter of column numbers in itemset
     *  @return     array of column numbers of items
     */
    public int[] getColumn()
    {
        return column;
    }

    /**
     * Getter of total row count
     *  @return     int of total row count
     */
    public int getTotal()
    {
        return total;
    }

}
```

```
/**
 * Utility.java
 */

package dw;

/**
 * Utility class that contains common utility functions
 * @author Mien Siao
 */
public class Utility
{

   public Utility()
   {
   }

   /**
    *  Static method that convert from based-2 value String into binary value
    *  @param binary   the String that holds binary value
    *  @return         int of equivalent value in based-2
    */
   public static int binary2int(String binary)
   {
      int ans = 0;
      for (int i = 0; i < binary.length(); i++)
      {
         char c = binary.charAt(i);
         if(c == '0')
            ans = 2 * ans;
         else if (c == '1')
            ans = 2 * ans + 1;
      }
      return ans;
   }

   /**
    *  Helper methods that countsthe numbers of bits after converting based-2 number into
binary
    *  @param str  the String that holds the based-2 value
    *  @return     number of 1 bits in the binary representation of the input string
    */
   public static int countBits(String str)
   {
      Long ll = new Long(str);
      long l = ll.longValue();
```

```
        return Long.bitCount(l);
    }

}
```