Frequent Set Meta Mining: Towards Multi-Agent Data Mining

Kamal Ali Albashiri, Frans Coenen, Rob Sanderson, and Paul Leng
Department of Computer Science, The University of Liverpool
Ashton Building, Ashton Street, Liverpool L69 3BX, United Kingdom
{ali,frans,azaroth,phl}@csc.liv.ac.uk

Abstract

In this paper we describe the concept of Meta ARM in the context of its objectives and challenges and go on to describe and analyse a number of potential solutions. Meta ARM is defined as the process of combining the results of a number of individually obtained Associate Rule Mining (ARM) operations to produce a composite result. The typical scenario where this is desirable is in multi-agent data mining where individual agents wish to preserve the security and privacy of their raw data but are prepared to share data mining results. Four Meta ARM algorithms are described: a Brute Force approach, an Apriori approach and two hybrid techniques. A "bench mark" system is also described to allow for appropriate comparison. A complete analysis of the algorithms is included that considers the effect of: the number of data sources, the number of records in the data sets and the number of attributes represented.

Keywords: Meta Mining, Multi Agent Data Mining, Association Rule Mining.

1 Introduction

The term *meta mining* describes the process of combining the individually obtained results of N applications of a data mining activity. This is typically undertaken in the context of Multi-Agent Data Mining (MADM) where the individual owners of agents wish to preserve the privacy and security of their raw data but are prepared to share the results of data mining activities. The mining activities in question could be, for example, clustering, classification or Association Rule Mining (ARM); the study described here concentrates on the latter — frequent set meta mining (which in this paper we will refer to as Meta ARM). The Meta ARM problem is defined as follows: a given ARM algorithm is applied to N raw data sets producing N collections of frequent item sets. Note that each raw data set conforms to some globally agreed attribute schema, although each local schema will typically comprise some subset of this global schema. The objective is then to *merge* the different sets of results into a single *meta* set of frequent itemsets with the aim of generating a set of ARs or alternative a set of Classification Association Rules (CARS).

The most significant issue when combining groups of previously identified frequent sets is that wherever an itemset is frequent in a data source A but not in a data source B a check for any contribution from data source B is required

(so as to obtain a global support count). The challenge is thus to combine the results from N different data sources in the most computationally efficient manner. This in turn is influenced predominantly by the magnitude (in terms of data size) of returns to the source data that are required. There are a number of alternative mechanisms whereby ARM results can be combined to satisfy the requirements of Meta ARM. In this paper a study is presented comparing five different approaches (including a bench mark approach). The study is conducted using variations of the TFP set enumeration tree based ARM algorithm ([6, 4]), however the results are equally applicable to other algorithms (such as FP growth [7]) that use set enumeration tree style structures, the support-confidence framework and an Apriori methodology of processing/building the trees.

The paper is organised as follows. In Section 2 some related work is presented and discussed. A brief note on the data structures used by the Meta ARM algorithms is then presented in Section 3. The five different approaches that are to be compared are described in Section 4. This is followed, in Section 5, by an analysis of a sequence of experimental results used to evaluate the approaches introduced in Section 4. Finally some conclusions are presented in Section 6.

2 Previous Work

Multi-Agent Data Mining (MADM) is an emerging field concerned with the application and usage of Multi-Agent Systems (MAS) to perform data mining activities. MADM research encompasses many issues. In this paper the authors address the issue of collating data mining results produced by individual agents, we refer to this as *meta-mining*.

The Meta ARM problem (as outlined in the above introduction) has similarities, and in some cases overlap, with incremental ARM (I-ARM) and distributed ARM. The distinction between I-ARM, as first proposed by Agrawal and Psaila [1], and Meta ARM is that in the case of I-ARM we typically have only two sets of results: (i) a large set of frequent itemsets D and (ii) a much smaller set of itemsets d that we wish to process in order to update D. In the case of Meta ARM there can be any number of sets of results which can be of any (or the same) size. Furthermore, in this case each contributing set has already been processed to obtain results in the form of locally frequent sets. I-ARM algorithms typically operate using a relative support threshold [8, 11, 15] as opposed to an absolute threshold; the use of relative thresholds has been adopted in the work described here. I-ARM algorithms are therefore supported by the observation that for an itemset to be *globally frequent* it must be *locally frequent* in at least one set of results regardless of the relative number of records at individual sources (note that this only works with relative support thresholds). When undertaking I-ARM four scenarios can be identified according to whether a given itemset i is: (i) frequent in d, and/or (ii) frequent in D; these are itemised in Table 1.

	Frequent in d	Not frequent in d
Frequent in D (retained itemsets)	Increment total count for i and recalculate support	i may be globally supported, increment total count for i and recalculate support
NotFrequent in D	May be globally supported, need to obtain total support count and recalculate support (Emerging itemset)	Do nothing

Table 1. I-ARM itemset comparison options (relative support)

From the literature, three fundamental approaches to I-ARM are described. These may be categorised as follows:

1. Maintain itemsets on the border with maximal frequent item sets (the *negative border* idea) and hope that this includes all those itemsets that may become frequent. See for example ULI [14].

2. Make use of a second (lower) support threshold above which items are retained (similar idea to negative border). Examples include AFPIM [8]) and EFPIM [11].

3. Acknowledge that at some time or other we will have to recompute counts for some itemsets and consequently maintain a data structure that (a) stores all support counts, (b) requires less space than the original structure and (c) facilitates fast look-up, to enable updating. See for example [9].

Using a reduced support threshold results in a significant additional storage overhead. For example if we assume that given a data set with 100 ($n = 100$) attributes where all the 1 and 2 item sets are frequent but none of the other itemsets are frequent, the negative border will comprise 161700 item sets ($\frac{n(n-1)(n-2)}{3!}$) compared to 4970 supported item sets ($n + \frac{n(n-1)}{2!}$). The Meta ARM ideas presented here therefore subscribe to this last of the above categories. The data at each source is held using a P-tree (**P**artial Support Tree) data structure, the nature of which is described in further detail in Section 3 below.

The distinction between distributed mining and MADM is one of control. Distributed ARM assumes some central control that allows for the global partitioning of either the raw data (*data distribution*) or the ARM task (*task distribution*), amongst a fixed number of processors. MADM, and by extension the Meta ARM mining described here, does not require this centralised control, instead the different sets of results are produced in an autonomous manner without any centralised control. MADM also offers the significant advantage that the privacy and security of raw data belonging to individual agents is preserved, an advantage that is desirable for both commercial and legal reasons.

In both I-ARM and distributed ARM, as well as Meta ARM, the raw data typically conforms to some agreed global schema.

Other research on meta mining that includes work on meta classification. Meta classification, also sometimes referred to as meta learning, is a technique for generating a *global* classifier from N distributed data sources by first computing N *base* classifiers which are then collated to build a single *meta* classifier [13] in much the same way that we are collating ARM results.

The term *merge mining* is used in [3] to describe a generalised form of incremental association rule mining which has some conceptual similarities to the ideas behind Meta ARM described here. However Aref et al. define merge mining in the context of time series analysis where additional data is to be merged with existing data as it becomes available.

3 Note on P and T Trees

The Meta ARM algorithms described here make use of two data structures, namely P-trees and T-trees. The nature of these structures is described in detail in [4, 6]; however, for completeness a brief overview is presented here.

The P-tree (**P**artial support tree) is a set enumeration tree style structure with two important differences: (i) more than one item may be stored at any individual node, and (ii) the tree includes partial support counts. The structure is sued to store a compressed version of the raw data set with partial support counts obtained during the reading of the input data. The best way of describing the P-tree is through an example such as that given in Figure 1. In the figure the data set given on the left is stored in the P-tree on the right. The advantages offered by the P-tree are of particular benefit if the raw data set contains many common leading sub-strings (prefixes). The number of such sub-strings can be increased if the data is ordered according to the frequency of the 1-itemsets contained in the raw data. The likelihood of common leading sub-strings also increases with the number of records in the raw data.

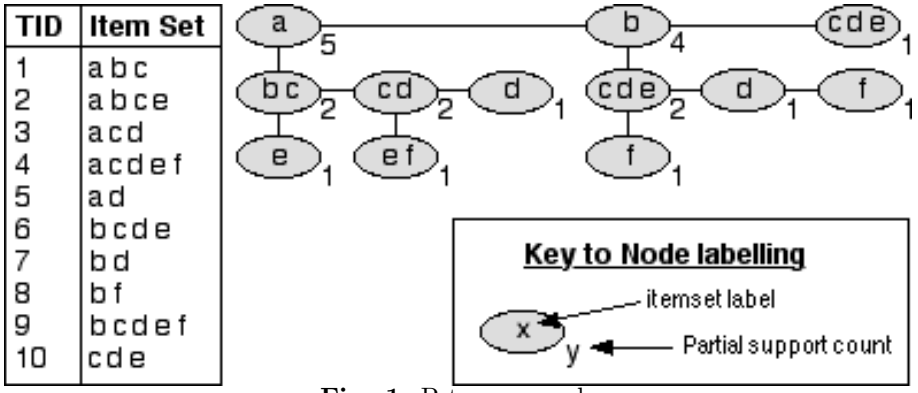


Fig. 1. P-tree example

The T-tree (**T**otal support tree) is a "reverse" set enumeration tree structure that inter-leaves node records with arrays. It is used to store frequent item

sets, in a compressed form, identified by processing the P-tree. An example, generated from the P-tree given Figure 1, is presented in Figure 2. ¿From the figure it can be seen that the top level comprises an array of references to node structures that hold the support count and reference to the next level (providing such a level exists). Indexes equate to itemset numbers although for ease of understanding in the figure letters have been used instead of numbers. The structure can be though of as a "reverse" set enumeration tree because child nodes only contain itemsets that are lexicographically before the parent itemsets. This offers the advantage that less array storage is required (especially if the data is ordered according to the frequency of individual items.

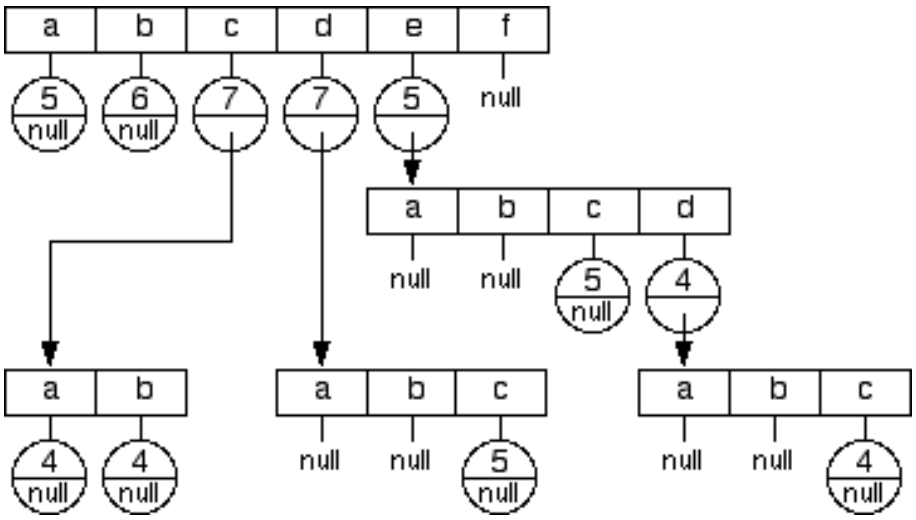


Fig. 2. T-tree example (support = 35%)

The T-tree is generated using an algorithm called Total From Partial (TFP) which is also described in [4, 6]. The TFP algorithm is essentially an Apriori style algorithm that proceeds in a level by level manner. At each level the P-tree is processed to generate appropriate support counts. Note that on completion of the TFP algorithm the T-tree contains details of all the supported itemsets, in a manner that provides for fast look up during AR generation, but no information about unsupported sets (other than that they are not supported). Referring to Figure 2 unsupported sets are indicated by a *null* reference.

4 Proposed Meta ARM Algorithms

In this section a number of Meta ARM algorithms are described, an analysis of which is presented in section 5. It is assumed that each data source will maintain the data set in either its raw form or a compressed form. For the experiments reported here the data has been stored in a compressed form using a P-tree (see above).

The first algorithm developed was a bench mark algorithm, against which the identified Meta ARM algorithms were to be compared. This is described in 4.1. Four Meta ARM algorithms were then constructed. For the Meta ARM algorithm it was assumed that each data source would produce a set of frequent sets using the TFP algorithm with the results stored in a T-tree. These T-trees would then be merged in some manner.

Each of the Meta ARM algorithms described below makes use of *return to data* (RTD) lists, one per data set, to contain lists of itemsets whose support was not included in the current T-tree and for which the count is to be obtained by a return to the raw data. RTD lists comprise zero, one or more tuples of the form $< I, sup >$, where I is an item set for which a count is required and sup is the desired count. RTD lists are constructed as a Meta ARM algorithm progresses. During RTD list construction the sup value will be 0, it is not until the RTD list is processed that actual values are assigned to sup. The processing of RTD lists may occur during, and/or at the end of, the Meta ARM process depending on the nature of the algorithm.

4.1 Bench Mark Algorithm

For Meta ARM to make sense the process of merging the distinct sets of discovered frequent itemsets must be faster than starting from the beginning (otherwise there is no benefit from undertaking the merging). The first algorithm developed was therefore a bench mark algorithm. This was essentially an Apriori style algorithm (see Table 2) that used a T-tree as a storage structure to support the generation process.

4.2 Brute Force Meta ARM Algorithm

The philosophy behind the Brute Force Meta ARM algorithm was that we simply fuze the collection of T-trees together adding items to the appropriate RTD lists as required. The algorithm comprises three phases: (i) merge, (ii) inclusion of additional counts and (iii) final prune. The merge phase commences by selecting one of the T-trees as the initial merged T-tree (from a computational perspective it is desirable that this is the largest T-tree). Each additional T-tree is then combined with the merged T-tree "sofar" in turn. The combining is undertaken by comparing each element in the top level of the merged T-tree sofar with the corresponding element in the "current" T-tree and then updating or extending the merged T-tree sofar and/or adding to the appropriate RTD lists as indicated in Table 3 (remember that we are working with relative support thresholds). Note that the algorithm only proceeds to the next branch in the merged T-tree sofar if an element represents a supported node in both the merged and current T-trees. At the end of the merge phase the RTD lists are processed and any additional counts included (the *inclusion of additional counts* phase). The final merged T-tree is then pruned in phase three to remove any unsupported frequent sets according to the user supplied support threshold (expressed as a percentage of the total number of records under consideration).

```
K = 1
Generate candidate K-itemsets
Start Loop
    if (K-itemsets == null break)
    forall N data sets get counts for K-itemsets
    Prune K-itemsets according to support threshold
    K ⇐ K+1
    Generate K-itemsets
End Loop
```

Table 2. Bench Mark Algorithm

	Frequent in T-tree N	Not frequent in T-tree N
Frequent in merged T-tree sofar	Update support count for i in merged T-tree sofar and proceed to child branch in merged T-tree sofar	Add labels for all supported nodes in merge T-tree branch, starting with current node, to RTD list N
Not Frequent in merged T-tree sofar	Process current branch in T-tree N, starting with current node, adding nodes with their support to the merged T-tree sofar and recording labels for each to RTD lists 1 to $N-1$	Do nothing

Table 3. Brute Force Meta ARM itemset comparison options

4.3 Apriori Meta ARM Algorithm

In the Brute Force approach the RTD lists are not processed until the end of the merge phase. This means that many itemsets may be included in the merged T-tree sofar and/or the RTD lists that are in fact not supported. The objective of the Aprori Meta ARM algorithm is to identify such unsupported itemsets much earlier on in the process. The algorithm proceeds in a similar manner to the standard Apriori algorithm (Table 2) as shown in Table 4. Note that items are added to the RTD list for data source n if a candidate itemset is not included in T-tree n.

```
K = 1
Generate candidate K-itemsets
Start Loop
    if (K-itemsets == null break)
    Add supports for level K from N T-trees or add to RTD list
    Prune K-itemsets according to support threshold
    K ⇐ K+1
    Generate K-itemsets
End Loop
```

Table 4. Apriori Meta ARM Algorithm

4.4 Hybrid Meta ARM Algorithm 1 and 2

The Apriori Meta ARM algorithm requires less itemsets to be included in the RTD list than is the case with the Brute Force Meta ARM algorithm (as demonstrated in Section 5). However, the Apriori approach requires the RTD lists to be processed at the end of each level generation, while in the case of the Brute Force approach this is only done once. A hybrid approach, that combines the advantages offered by both the Brute Force and Apriori meta ARM algorithms therefore suggests itself. Experiments were conducted using two different version of the hybrid approach.

The Hybrid 1 algorithm commences by generating the top level of the merged T-tree in the Apriori manner described above (including processing of the RTD list); and then adds the appropriate branches, according to which top level nodes are supported, using a Brute Force approach.

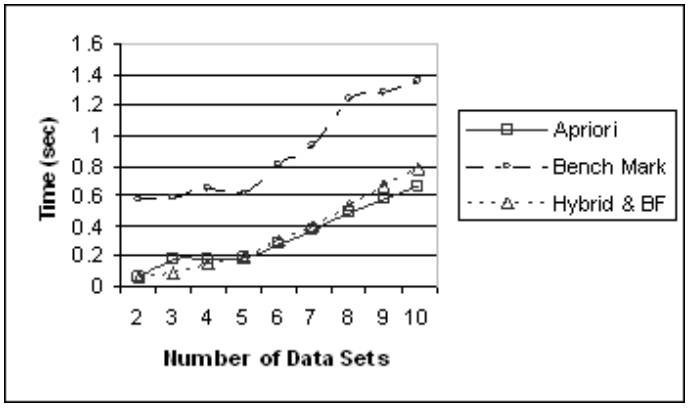
The Hybrid 2 algorithm commences by generating the top two levels of the merged T-tree, instead of only the first level, as in the Hybrid 1 approach. Additional support counts are obtained by processing the RTD lists. The remaining branches are added to the supported level 2-nodes in the merged T-tree sofar (again) using the Brute Force mechanism. The philosophy behind the hybrid 2 algorithm was that we might expect all the one itemsets to be supported and included in the component T-trees therefore we might as well commence by building the top two layers of the merged T-tree.

5 Experimentation and Analysis

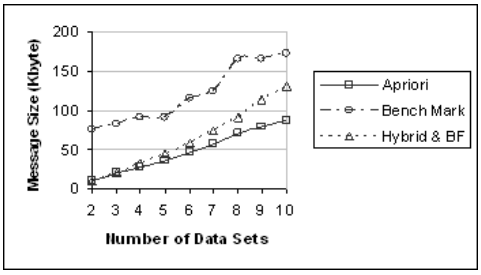
To evaluate the five algorithms outlined above a number of experiments were conducted. These are described and analysed in this section. The experiments were designed to analyse the effect of the following:

1. The number of data sources.
2. The size of the datasets in terms of number of records .
3. The size of the datasets in terms of number of attributes.

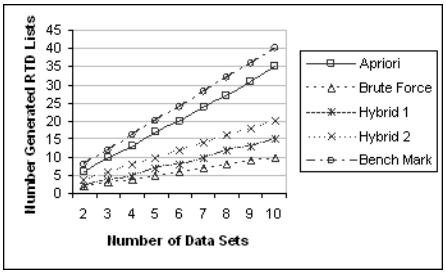
All experiments were run using a Intel Core 2 Duo E6400 CPU (2.13GHz) with 3GB of main memory (DDR2 800MHz), Fedora Core 6, Kernel version 2.6.18 running under Linux. For each of the experiments we measured: (i) processing time (seconds / mseconds), (ii) the size of the RTD lists (Kbytes) and (iii) the number of RTD lists generated. The authors did not use the IBM QUEST generator [2] because many different data sets (with the same input parameters) were required and the quest generator always generated the same data given the same input parameters. Instead the authors used the LUCS KDD data generator [1].



(a) Processing Time



(b) Total size of RTD lists



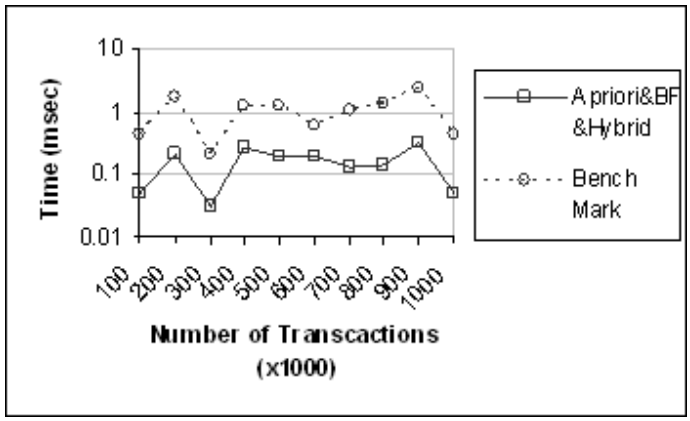
(c) Number of RTD lists

Fig. 3. Effect of number of data sources

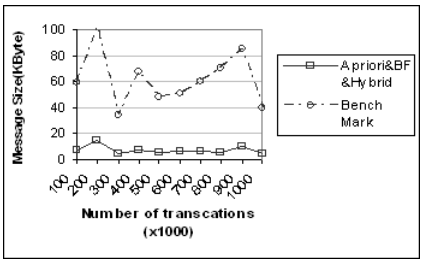
Figure 3 shows the effect of adding additional data sources. For this experiment ten different artificial data sets were generated using $T = 4$ (average number of items per transactions), $N = 20$ (Number of attributes), $D = 100k$ (Number of transactions). Note that the selection of a relatively low value for N ensured that there were some common frequent itemsets shared across the T-trees. Experiments using $N = 100$ and above tended to produced very flat T-trees with many frequent 1-itemsets, only a few isolated frequent 2-itemsets and no frequent sets with cardinality greater than 2. For the experiments a support

[1] $http://www.csc.liv.ac.uk/\ frans/KDD/Software//LUCS-KDD-DataGen/$

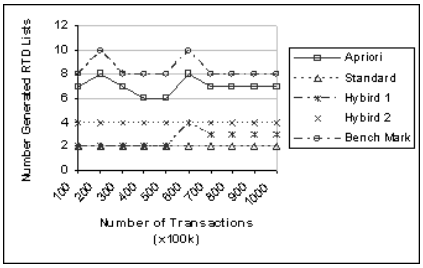
threshold of 1% was selected. Graph 3(a) demonstrates that all of the proposed Meta ARM algorithms worked better then the bench mark (start all over again) approach. The graph also shows that the Apriori Meta ARM algorithm, which invokes the "return to data procedure" many more times than the other algorithms, at first takes longer; however as the number of data sources increases the approach starts to produce some advantages as T-tree branches that do not include frequent sets are identified and eliminated early in the process. The amount of data passed to and from sources, shown in graph 3(b), correlates directly with the execution times in graph 3(a). Graph 3(c) shows the number of RTD lists generated in each case. The Brute Force algorithm produces one (very large) RTD list per data source. The Bench Mark algorithm produces the most RTD lists as it is constantly returning to the data sets, while the Apriori approach produces the second most (although the content is significantly less).



(a) Processing Time



(b) Total size of RTD lists

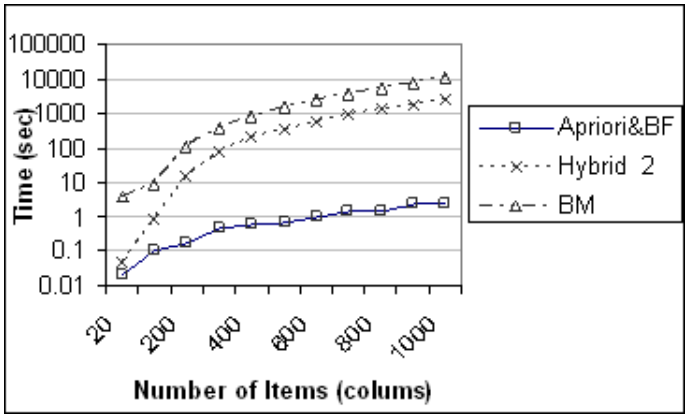


(c) Number of RTD lists

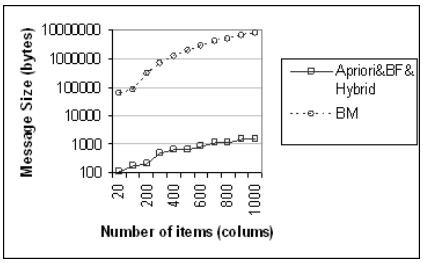
Fig. 4. Effect of increasing number of records

Figure 4 demonstrates the effect of increasing the number of records. The input data for this experiment was generated by producing a sequence of ten pairs of data sets (with $T = 4$, $N = 20$) representing two sources. From graph 4(a) it can be seen that the all algorithms work outperformed the bench

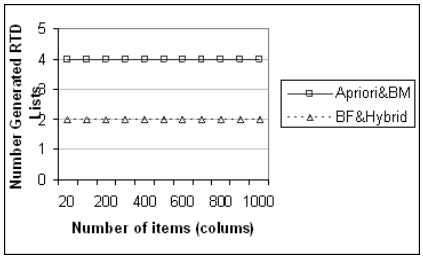
mark algorithm because the size of the return to data lists are limited as no unnecessary candidate sets are generated. This is illustrated in graph 4(b). Graph 4(b) also shows that the increase in processing time in all case is due to the increase in the number of records only, the size of the RTD lists remains constant through as does the number of RTD lists generated (graph 4(c)). The few "bumps" at the results are simply from a vagary of the random nature of the test data generation.



(a) Processing Time



(b) Total size of RTD lists



(c) Number of RTD lists

Fig. 5. Effect of increasing number of items (attributes)

Figure 5 shows the effect of increasing the global pool of potential attributes (remember that each data set will include some subset of this global set of attributes). For this experiment another sequence of pairs of data sets (representing two sources) was generated with $T = 4$, $D = 100K$ and N ranging from 100 to 1000. As in the case of experiment 2 the Apriori, Brute Force and Hybrid 1 algorithms work best (for similar reasons) as can be seen from graph 5(a,b). However in this case (compared to the previous experiment), the Hybrid 2 algorithm did not work as good. The reasoning behind the Hybrid 2 algorithm slower perfromance is that all the 1-itemsets tended not to be all supported and becuase there were not eliminated and included in 2-itemsets generation (graph 5(a)). For completeness graph 5(c) indicates the number of RTD lists sent with respect to the different algorithms.

All the Meta ARM algorithms outperformed the bench mark algorithm. The Hybrid 2 algorithm also performed in an unsatisfactory manner largely because of the size of the RTD lists sent. Of the remainder the Apriori approach coped best with a large number of data sources, while the Brute Force and Hybrid 1 approaches coped best with increases data sizes (in terms of column/rows) again largely because of the relatively smaller RTD list sizes.

It should also be noted that the algorithms are all complete and correct, i.e. the end result produced by all the algorithms is identical to that obtained from mining the union of all the raw data sets using some established ARM algorithm. In practice of course the MADM scenario, which assumes that data cannot be combined in this centralised manner, would not permit this.

6 Conclusions

In this paper we have described a novel extension of ARM where we build a meta set of frequent itemsets from a collection of component sets which have been generated in an autonomous manner without centralised control. We have termed this type of conglomerate Meta ARM so as to distinguish it from a number of other related data mining research areas such as incremental and distributed ARM. We have described and compared a number of meta ARM algorithms: (i) Bench Mark, (ii) Apriori, (iii) Brute Force, (iv) Hybrid 1 and (v) Hybrid 2. The results of this analysis indicates that Apriori Meta ARM approach coped best with a large number of data sources, while the Brute Force and Hybrid 1 approaches coped best with increases data sizes (in terms of column/rows). The work represents an important "milestone" towards a multi-agent approach to ARM that the authors are currently investigating.

References

[1] Agrawal, R. and Psaila, G. (1995). Active Data Mining. Proc. 1st Int. Conf. Knowledge Discovery in Data Mining, AAAI, pp 3-8.

[2] Agrawal, R., Mehta, M., Shafer, J., Srikant, R., Arning, A. and Bollinger, T. (1996). The Quest Data Mining System. Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining, (KDD1996).

[3] Aref, W.G., Elfeky, M.G., Elmagarmid, A.K. (2004). Incremental, Online, and Merge Mining of Partial Periodic Patterns in Time-Series Databases. IEEE Transaction in Knowledge and Data Engineering, Vol 16, No 3, pp. 332-342

[4] Coenen, F.P. Leng, P., and Goulbourne, G. (2004). Tree Structures for Mining Association Rules. Journal of Data Mining and Knowledge Discovery, Vol 8, No 1, pp25-51.

[5] Dietterich, T.G. (2002). Ensemble Methods in Machine Learning. In Kittler J. and Roli, F. (Ed.), First International Workshop on Multiple Classifier Systems, LNCS pp1-15.

[6] Goulbourne, G., Coenen, F.P. and Leng, P. (1999). Algorithms for Computing Association Rules Using A Partial-Support Tree. Proc. ES99, Springer, London, pp132-147.

[7] Han,J., Pei, J. and Yiwen, Y. (2000). Mining Frequent Patterns Without Candidate Generation. Proc. ACM-SIGMOD International Conference on Management of Data, ACM Press, pp1-12.

[8] Koh, J.L. and Shieh, S.F. (2004). An efficient approach to maintaining association rules based on adjusting FP-tree structures. Proc. DASFAA 2004, pp417-424.

[9] Leung, C.K-S., Khan, Q.I, and Hoque, T, (2005). CanTree: A tree structure for efficient incremental mining of Frequent Patterns. Proc. ICDM 2005. pp274-281.

[10] Li W., Han, J. and Pei, J. (2001). CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules. Proc. ICDM 2001, pp369-376.

[11] Li, X., Deng, Z-H. and Tang S-W. (2006). A Fast Algorithm for Maintenance of Association Rules in Incremental Databases. Proc. ADMA 2006, Springer-Verlag LNAI 4093, pp 56-63.

[12] Liu, B. Hsu, W. and Ma, Y (1998). Integrating Classification and Association Rule Mining. Proc. KDD-98, New York, 27-31 August. AAAI. pp80-86.

[13] Prodromides, A., Chan, P. and Stolfo, S. (2000). Meta-Learning in Distributed Data Mining Systems: Issues and Approaches. In Kargupta, H. and Chan, P. (Eds), Advances in Distributed and Parallel Knowledge Discovery. AAAI Press/The MIT Press, 2000, pp81-114.

[14] Thomas, S., Bodagala, S., Alsabti, K. and Ranka, S. (1997). An efficient algorithm for the incremental updation of association rules. In Proc. 3rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. pp263-266.

[15] Veloso, A.A., Meira, W., de Carvalho B. Possas, M.B., Parthasarathy, S. and Zaki, M.J. (2002). Mining Frequent Itemsets in Evolving Databases. Proc. Second SIAM International Conference on Data Mining (SDM'2002).