2009 Ninth International Conference on Intelligent Systems Design and Applications

A Fast Algorithm for Mining Rare Itemsets

Luigi Troiano
University of Sannio
Department of Engineering
82100 Benevento, Italy
troiano@unisannio.it

Giacomo Scibelli
Poste Italiane S.p.A - TI- SSI
Centro Ricerca e Sviluppo
80133 Napoli, Italy
scibell9@posteitaliane.it

Cosimo Birtolo
Poste Italiane S.p.A - TI- SSI
Centro Ricerca e Sviluppo
80133 Napoli, Italy
birtoloc@posteitaliane.it

***Abstract*—Mining patterns in large databases is a challenging task facing NP-hard problems. Research focused attention on the most occurrent patterns, although less frequent patterns still offer interesting insights. In this paper we propose a new algorithm for discovering infrequent patterns and compare it to other solutions.**

I. Introduction

Mining patterns in business transactions, time-series, genetic records, and many other kinds of data is a basic step in formulating hypotheses and discovering associations among values, i.e. items. For instance, the market basket analysis in the retail business is aimed at discovering which products tend to be bought together in order to capture the purchase behavior of customers, to understand their needs, to develop cross-promotional programs, to acquire new buyers or, more in general, to improve business performances. In order to do this, generally analysts look at the most recurrent patterns, although also looking at the least frequent co-occurrences can provide interesting insights, regarding smaller but not less interesting groups.

Recently, an increasing importance is given to the discovery of those patterns which are infrequent, such as in identifying fraudulent credit card transactions, learning word pronunciations, predicting pre-term births, predicting telecommunication equipment failures, linking cancer to medical tests, and detecting oil spills from satellite images [1]. In market basket analysis, some sets of items, such as peanut butter and jelly, occur frequently and can be considered common cases. Other associations may be extremely rare. For example, food processor and cooking pan will be an extremely rare association in a supermarket, not because the items are unlikely to be purchased together, but because no item is frequently purchased in a supermarket [2]. Non-frequent itemsets can unveil interesting business opportunities and niche markets. Indeed, rare data associations appearing infrequently in a database, for example in only 1% of transactions, become interesting to analyze in a database of $100K$ transactions as this means 1000 cases.

The problem of identifying rare or infrequent patterns is known to be complex as much as the problem of discovering frequent patterns. In literature both problems have been faced by several authors and some algorithms have been proposed. In this paper, we discuss issues related to mining rare itemsets and present a new algorithm, named Rarity, for discovering them in large databases. The remaining of this paper is organized as follows: In Section 2 we provide some preliminary definitions, in Section 3 we discuss the problem of discovering frequent and infrequent itemsets, in Section 4 we introduce the Rarity algorithm, describing main features and strategy; in Section 5 we illustrate experimental results in comparing the algorithm to other solutions; in Section 6 we point out some open issues and future directions.

II. Preliminaries

A transaction is a record of one or more items collected from a finite item domain, and a dataset is a collection of transactions. An itemset is a non-empty subset of items. According to R. Agrawal, T. Imielinski, and A. Swami [3] support is defined as the number of itemset occurences in the dataset. Itemsets whose support is higher than a given threshold are defined as *frequent*. On the contrary, we define as *rare* (or *infrequent*) itemsets those that are not frequent, thus with support below the given threshold.

The main property of support is *antimonotonicity* entailing that all subsets of a frequent itemset are also frequent. On the opposite, this means that all supersets of an infrequent itemset are infrequent themselves. For instance, if ABC is a frequent itemset, subsets $\{AB, BC, AC, A, B, C\}$ are also frequent itemsets. At the same time, if an itemset RS is infrequent, its superset RST is infrequent as well. As proven by Yang [4] the problem of counting the number of distinct maximal frequent itemsets in a dataset, given an arbitrary support threshold, is NP-complete and the problem of mining maximal frequent itemsets is NP-hard. Complexity comes out from the need of traversing the itemset power set lattice. Antimonotonicity can be used to find the downward closure of the power set lattice of frequent itemsets, thus to prune the search space. As the set of infrequent itemsets is complementary to the set of frequent itemsets, counting and mining rare itemsets are still NP-complete and NP-hard respectively.

978-0-7695-3872-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ISDA.2009.55

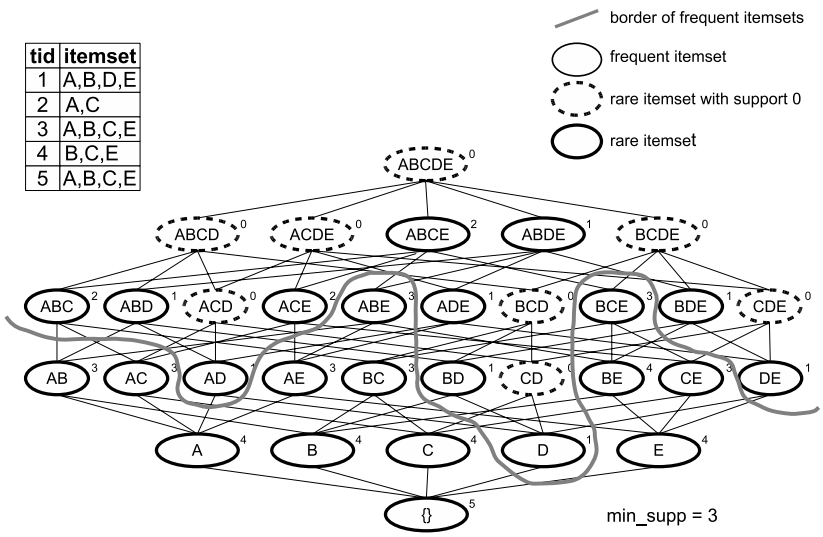

Figure 1. The power set lattice of dataset $\mathcal{D}$, shown on the table at the left

III. Mining Frequent and Rare Itemsets

A primer algorithm to discover frequent itemsets in a database is the *Apriori* algorithm proposed by Agrawal and Srikant [5]. This algorithm is level-wise, as it considers itemsets with different cardinality at each step. At step k frequent itemsets having k items are available in F_k. Each step is made of two phases. First, $(k+1)$-itemsets are generated from elements in F_k. Then their support is computed by scanning the dataset, discarding those itemsets whose support is below the threshold. The result is the list of frequent itemsets F_{k+1}. The algorithm stops when F_{k+1} is empty or $k = l_m$ that is the maximum itemset length. Resulting frequent itemsets are obtained by merging lists F_k with $k = 1..l_m$.

Although the Apriori algorithm is able to prune large parts of the search space, it can be still computationally expensive on large databases, and several improvements to the way frequent itemsets are processed and stored (e.g. see reference [6]). Most of them address the problem of making Apriori parallel. This is the case of Parallel CD algorithm [7] that performs the search of frequent itemsets exploring a spectrum of trade-offs between computation, communication, memory usage, synchronization and availability of some specific information.

Since Apriori requires a database to be scanned several times and the number of scans cannot be determined in advance, Savasere, Omiecinski, and Navathe [8] propose a new algorithm called *Partition* in order to overcome these limitations. The algorithm is executed in two phases and requires to scan the database only twice. In phase I, the database is split into a number of non-overlapping partitions which are considered one at a time and all frequent itemsets for that partition are generated. At the end of phase I, frequent itemsets are merged in order to generate a set of all potential frequent itemsets. In phase II, the actual support for these itemsets is computed and frequent itemsets are identified.

Apriori-like methods have another drawback: they may need to generate a huge number of candidate sets. Therefore Han et al. [9] came up with a solution based on compact tree structure, named FP-Tree, on which to apply a partition-based divide and conquer mining strategy. This approach has proved to perform faster than other techniques.

More recently some authors investigated the problem of mining rare itemsets [10], [11]. As an example, let us consider the case depicted in Fig.1. In particular we consider a dataset $\mathcal{D}$ populated by 5 records. Each record represents a transaction involving a certain number of items identified by capital letters (i.e. $A - E$). Itemsets can be mapped over the power set lattice depicted in the figure. The support is given by the number on the right side of each itemset. In order to mine rare itemsets, algorithms can implement different strategies aimed at moving across the lattice looking for those itemsets with a support below a threshold (i.e. $min\_support$). Antimonotonicity suggests a means for mining rare itemsets.

Koh and Rountree [10] proposed Apriori-Inverse as a variant for discovering sporadic rules by discarding all itemsets above a maximum support threshold. This algorithm is much faster than Apriori in finding perfectly rare itemsets, that are a subclass of rare itemsets containing itemsets whose all subsets are rare. In general a rare itemset can still have frequent subsets. In order to search the whole class of rare itemsets, Szathmary, Napoli and Valtchev [11] proposed an algorithm called ARIMA (A Rare Itemset Miner Algorithm) that is not restricted to perfectly rare itemsets.

ARIMA introduces a taxonomy for the itemsets. First,

the class of rare itemsets is split between rare itemsets with support 0 and rare itemsets with non-zero support. A *minimal rare itemset* is an itemset whose sub-itemsets are all frequent; instead a *maximal frequent itemset* is a frequent itemset whose super-itemsets are all rare. A *minimal* or *key generator* is an itemset whose sub-itemsets differ by support; a *minimal zero generator*, is an itemset with support 0 and non-zero support sub-itemsets. In order to find rare itemsets, it is sufficient to identify the minimal rare itemsets and related super-itemsets as they will be surely rare. After, because rare itemsets with support 0 are not of interest, the algorithm can stop when a minimal zero generator is found. The support is simply computed by scanning the dataset and counting the itemset occurrences. ARIMA aims at identifying the borderline which separates the frequent itemsets from the rare. The borderline is drawn by identifying the maximal frequent itemsets and the minimal rare itemsets. The strategy ARIMA follows is to start from the bottom of lattice and to move upwards in order to reach the limit of rare itemsets. All itemsets above that limit belong to the class of rare itemsets. In other terms, ARIMA implements a levelwise bottom-up approach, computing the itemset support by scanning the dataset at each level.

Therefore ARIMA presents the following limitations: (i) if there exist few rare itemsets they will probably be on the top of the lattice, thus a bottom-up approach can be not so efficient, (ii) some rare itemsets have a support equal to 0 and thereby these itemsets are not in the database, (iii) the database is scanned once per level during the execution of the algorithm in order to evaluate the support.

IV. Rarity Algorithm

Due to considerations above, the Rarity algorithm implements a different strategy. It starts by identifying the longest rare itemsets on the top, and moves downwards the power set lattice cutting itemsets resulting as frequent, and developing only those that are confirmed to be rare. Indeed, as described in Section 2, a frequent itemset entails sub-itemsets that are necessarily frequent. Differently, rare itemsets can provide sub-itemsets that are possibly but not necessarily rare.

In order to implement this strategy, the algorithm requires two data structures, namely (i) the candidate list C and (ii) the veto list V.[1] The candidate list as aimed at collecting itemsets that are possibly rare, whilst the veto list contains known frequent itemsets. Both lists are organized by levels, so that $C(l)$ and $V(l)$ refer only to itemsets long l. In addition, a list R containing all resulting rare itemsets is also considered. Pseudo-code is outlined by Algorithm 1.

The algorithm initializes the candidate list going through the database, inserting and counting each record long l in $C(l)$, as each record identifies a possible rare itemset.

[1]For the sake of simplicity, we refer to them as lists, although they are actually sets as they do not admit duplicates.

Algorithm 1 Rarity pseudo-code

```
1: l_m = max len(t) ∀t ∈ D
2: for all record t ∈ D do
3:    add t to C
4: end for
5: for l = l_m..1 do
6:    if C(l) ≠ ∅ then
7:       for all is ∈ C(l) do
8:          if supp(is) > min_supp then
9:             remove is from C(l)
10:            add is to V(l)
11:         else
12:            add is to R(l)
13:            if len(is) > 1 then
14:               for all sub ∈ subsets(is) do
15:                  if sub ∉ V then
16:                     add sub to C
17:                     v_sub = v_sub + v_is
18:                  end if
19:               end for
20:            end if
21:         end if
22:      end for
23:      for all is ∈ V(l) do
24:         if len(is) > 1 then
25:            for k = l − 1..1 do
26:               for all c ∈ C(k) do
27:                  cis = c ⋂ is
28:                  remove cis from C, if cis ∈ C
29:                  add cis to V
30:               end for
31:            end for
32:         end if
33:      end for
34:   end if
35: end for
```

The veto list V is initially empty. The algorithm starts by considering the longest itemsets in $C(l_m)$, where $l_m = argmax C(l) | C(l) \neq \emptyset$. For each $l = l_m..1$, the algorithm considers the candidate itemset $c_i \in C(l)$. If $supp(c_i)$ is known to be greater than threshold t, it is considered as frequent then moved into veto list $V(l)$. Differently, c_i is rare, therefore inserted into the rare list R. In addition, sub-itemsets long $l-1$ are possibly rare, then inserted into $C(l-1)$. After, the veto list $V(l)$ is scanned and each known frequent itemset $f_j \in V(l)$ is compared to shorter candidates $g_k \in C(h)$ with $h < l$, and intersection $e_{jk} = f_j \cap g_k$ is determined in order to find a common sub-itemset. As e_{jk} is known to be frequent as derived by f_j, thus it is moved to (or inserted into it has never been considered as far) the veto list $V(l_{jk})$ where $l_{jk} = length(e_{jk})$. The last level that

is considered by the algorithm is $l = 1$ made of singletons, even though the algorithm stops early when $C(l)$ is empty.

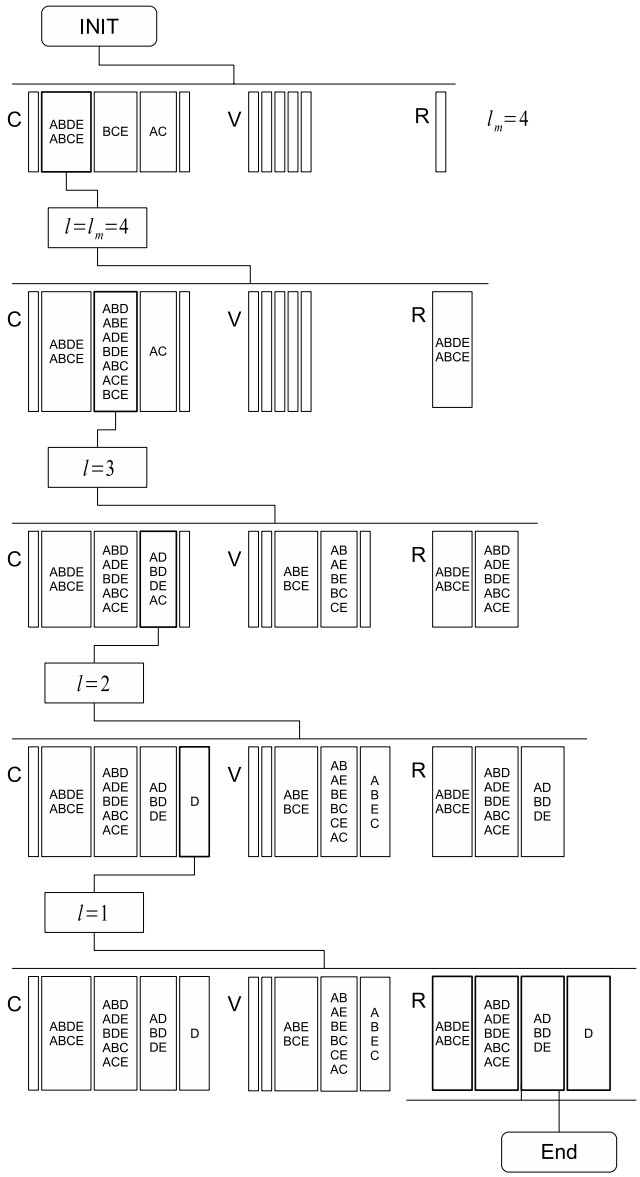

Figure 2. Execution of the Rarity algorithm

Rarity performs an efficient computation of support. Each itemset contributes by its occurrences to the support of each sub-itemset, as shown by Fig.1. In particular we can write

$$\mathbf{v}_{is} = \sum_{i \in P(is)} \mathbf{v}_i \tag{1}$$

where i is a generic itemset belonging to the set of is's super-itemsets $P(is)$. Vector $\mathbf{v}$ takes into the account the contribution to the itemset's support provided by super-itemsets at different level. This contribution arrives to the itemsets by different paths on the lattice. The number of paths, which depends on the level difference between the itemset and the super-itemset, is $(l - h)!$. In addition the element of $\mathbf{v}$ related to the level of is holds its own occurrences.

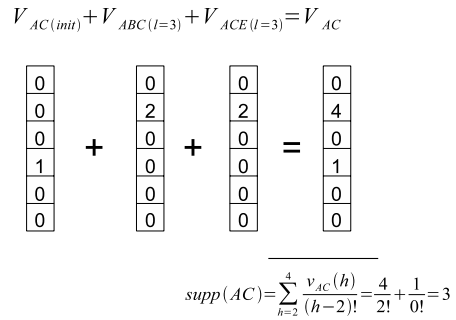

Figure 3. Computation of vector $\mathbf{v}$ and support

Therefore, the support of is can be computed as

$$supp(is) = \sum_{h=l_{is}}^{l_m} \frac{v_{is}(h)}{(h - l_{is})!} \tag{2}$$

where l_{is} is the is's length, whilst l_m is still the maximum itemset length as contribution is no further provided. This entails Rarity considers an additional data structure holding the support vector $\mathbf{v}$ for each itemset. When a candidate is evaluated, all contributions to its support are available in $\mathbf{v}$, and support can be computed according to Eq.2.

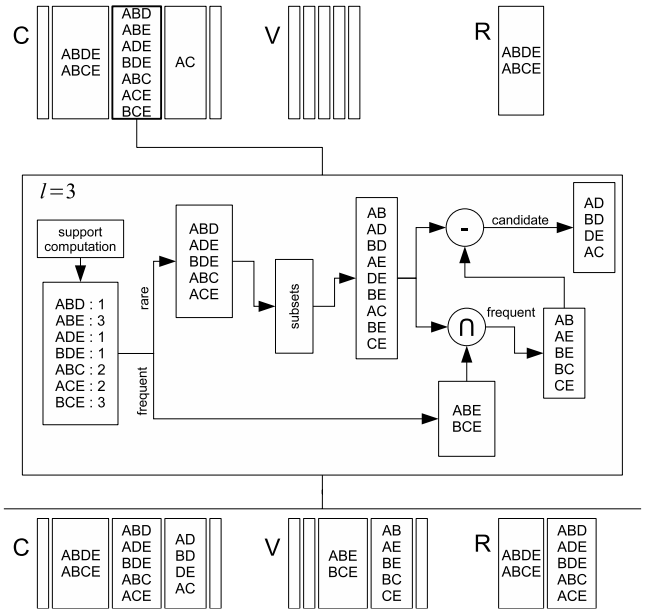

Figure 4. Algorithm processing at $l = 3$

So Rarity is able to pass through the database only once at initialization time, differently by Arima. This feature comes with other optimizations in order to improve overall

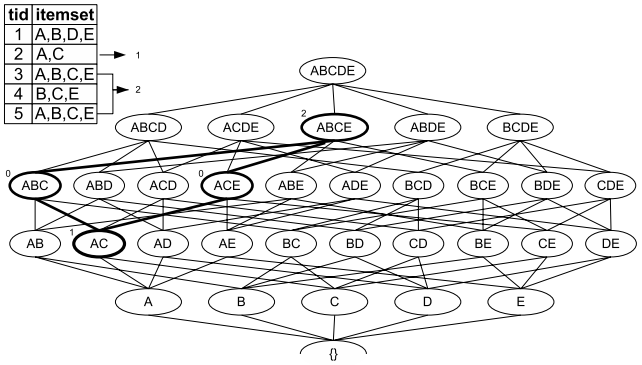

Figure 5. Application of example

performances. For instance, at level l the algorithm skips the whole step, thus with no scan of the veto list $V(l)$, if the candidate list $C(l)$ is empty. Indeed, if there is no candidate $c_i \in C(l)$, the itemsets into $V(l)$ have been necessarily obtained as intersection with longer candidate itemsets, already considered. In other words, a new frequent itemset to be at level l, it would be necessarily first a candidate, making the candidate list $C(l)$ not empty. This suggests a further optimization: the only itemsets in $V(l)$ required to be considered for intersection are those newly determined as frequent at step l.

The execution of Rarity on dataset $\mathcal{D}$ (Fig.1) with $min\_supp = 3$ is illustrated in Fig.2. The algorithm first performs one database scan to initialize its structures (C, V, R). So in the C list are added the itemsets stored in the dataset. After, Rarity starts by exploring each level l. Since there no exist a 5-itemset in the database, $C(5)$ is empty so rarity starts analyzing the 4^{th} level. It calculates the support (Eq.2) of $ABDE$ and $ABCE$ and since it is less than min_supp the itemsets are copied to R and the subsets are generated and inserted into C. The same happens at the 3^{rd} level. The elements in $C(3)$ are scanned and two of them (ABE and BCE) are moved into V. This entails Rarity scans the itemsets in $V(3)$ in order to inhibit them. So, after the scanning of V, AB, BE, AE, BC, CE (which are the subsets of ABE and BCE) are moved to V.

In Fig.3 is also shown how Rarity calculates $\mathbf{v}_{sub}$. The example illustrates the value of $\mathbf{v}_{AC}$. At the initialization time $\mathbf{v}_{AC}$ is $\mathbf{v}_{AC,init}$. When Rarity scans $C(3)$ and analyzes ABC, since this itemset is rare, Rarity must induce ABC's $\mathbf{v}$ to AC as when Rarity analyzes ACE. So according with Eq.1 we obtain the new value of $\mathbf{v}_{AC}$. When Rarity scans the 2^{nd} level, the elements in $C(2)$ are all rare so they are copied to R and then scanning $V(2)$ rarity inhibits singletons A, B, E, C. Finally, Rarity scans $C(1)$. D is the only itemset and since its support is less than min_supp it is copied into R. In particular, Fig.4 shows the execution of algorithm when $l = 3$. When support of all itemsets in $C(3)$ has been evaluated, frequent itemsets are moved into veto list $V(3)$, whilst the others are moved into R. After, the algorithm generates the candidate with $l = 2$ which are subsets of rare itemsets whose length is equal to 3. The result of intersection between $C(2)$ and $V(3)$ is the list of frequent itemsets which are moved from $C(2)$ to $V(2)$ (in the case study depicted in Fig.4, these itemsets are AB, AE, BE, BC and CE). In Fig.5 we show the result of the example described in Fig.2 and in Fig.3. In this case study the number above the nodes represents the occurrences of each subset in the dataset $\mathcal{D}$.

V. Experimental Results

In our experiments we compared Rarity to ARIMA [11]. Both Rarity and ARIMA have been made available in Java as components of the same framework.

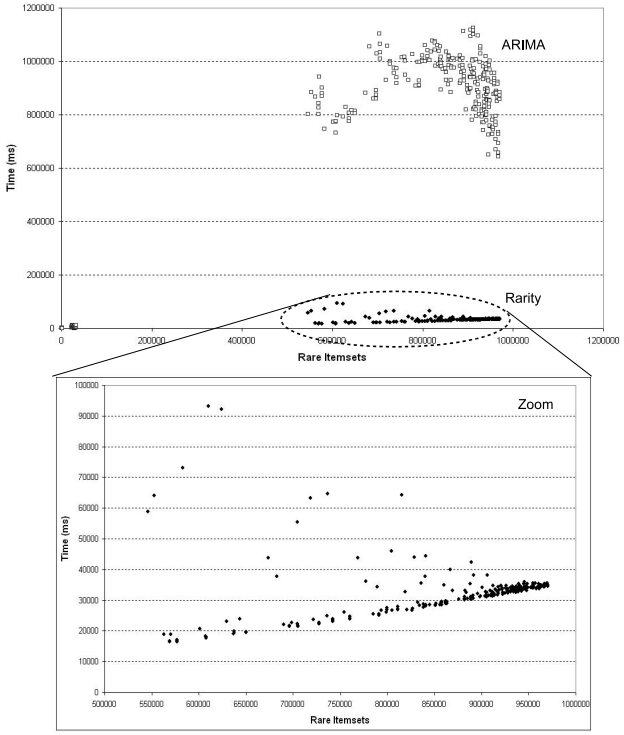

Figure 6. Execution Time: ARIMA and Rarity

Experiments were carried out on Intel Xeon 2.66 GHz machine with 4 GB of RAM running Windows Server 2003 Enterprise Edition Service Pack 2. For experimentation we randomly generated 200 datasets differing by the number of rows and the maximum number of items in a transaction. In particular we generated dataset with a number of rows ranging from 1000 to 10000 and with maximum transaction length of 5, 10, 15, and 20 items, each configuration considered 5 times. For the support threshold we assumed different values of min_supp: specifically 0.1%, 0.5%, 1.0%, 5.0%, 10.0% of the dataset size and $40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 60, 70, 80$ occurrences. Fig.6 compares the execution time of ARIMA and

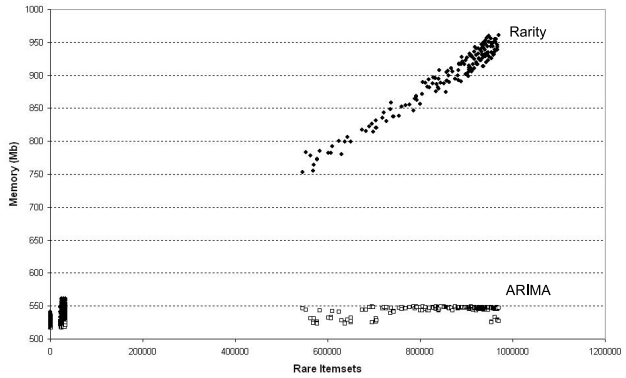

Figure 7. Memory Occupation: ARIMA and Rarity

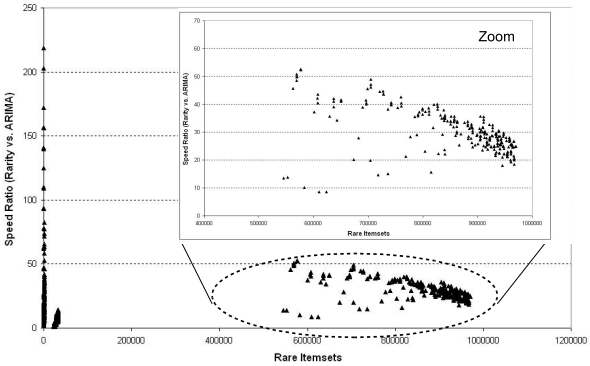

Figure 9. Speed ratio: Rarity vs. ARIMA

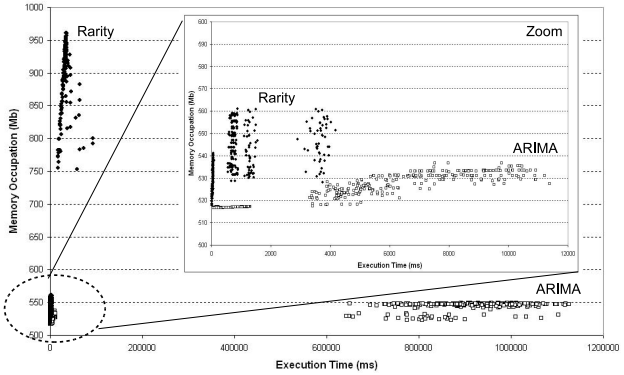

Figure 8. Memory vs. Time: Rarity and ARIMA

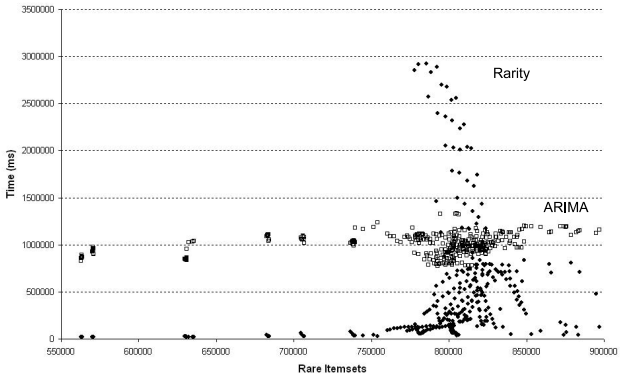

Figure 10. Extremely rare itemsets. Execution Time: ARIMA and Rarity

Rarity. The comparison in memory usage is outlined in Fig.7. All figures highlight Rarity to be faster at expenses of memory usage, whilst ARIMA is slower with a smaller memory footprint. This result becomes more evident by plotting time versus memory as depicted in Fig.8. In Fig.9 we provide the speed ratio of Rarity vs. ARIMA at varying the number of rare itemsets. We notice that Rarity speedup decreases when the number of rare itemsets increases, although it is constantly greater than 1.

The same analysis has been performed on absolute support thresholds. Fig.10 and Fig.11 respectively compare execution time and memory usage. In this case, itemsets become extremely rare, and Rarity performances become spreader in time, using more memory in general as depicted in Fig.12. This outcome is confirmed by Fig.13 related to Rarity speed vs. ARIMA one.

VI. Conclusions and Future Work

We presented a new approach for mining rare itemsets in large databases. The described algorithm, differently from existing implementations, uses a top-down strategy in traversing the power set lattice as rare itemsets generally occupy the top. In order to evaluate the solution proposed, we compared performances with ARIMA. Experimental results highlight Rarity to be faster than ARIMA in the most of cases, but it requires more memory. When the support threshold becomes extremely low compared to the dataset size, the comparison between the two algorithms become unpredictable. This is caused by the larger number of intersections between frequent itemsets found at each level and candidates stored at the following levels. This leads to an optimization aimed at reducing the number of subsequent itemsets to evaluate. This optimization is based on the condition that an itemset is already known to be frequent as obtained as subset of a larger frequent itemset. Finally, we are planning to apply this algorithm to some business cases. In particular, we aim to discover non-frequent patterns in customer purchasing behavior in order to identify new and unexploited business opportunities. As databases involve millions of transactions (e.g. 20Mln daily transactions, 2Mln money orders per day, 250.000 parcel deliveries per day, etc.) made of a large number of items (e.g. 10.000 different items sold by PosteShop, etc.), the mining of rare itemsets is demanded to face feasibility of solution, thus leading to a parallel implementation of algorithms.

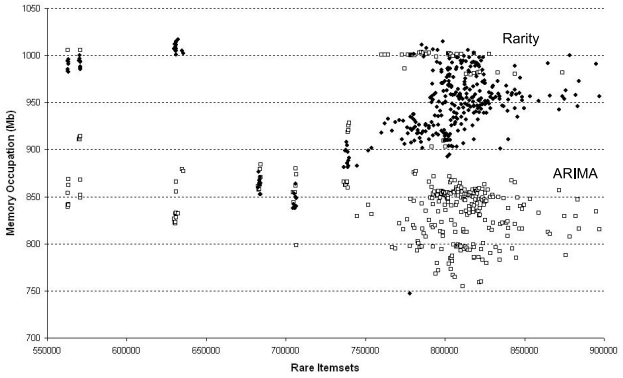

Figure 11. Extremely rare itemsets. Memory Occupation: ARIMA and Rarity

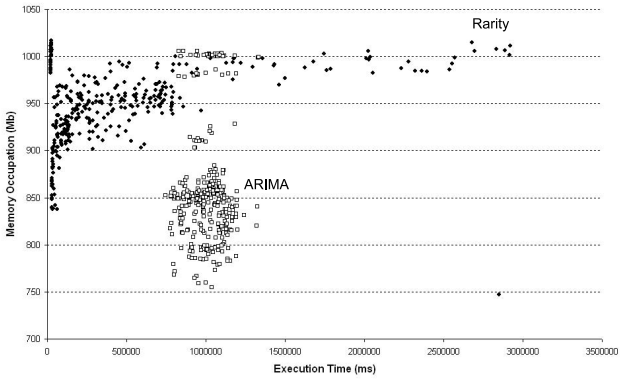

Figure 12. Extremely rare itemsets. Memory vs. Time.

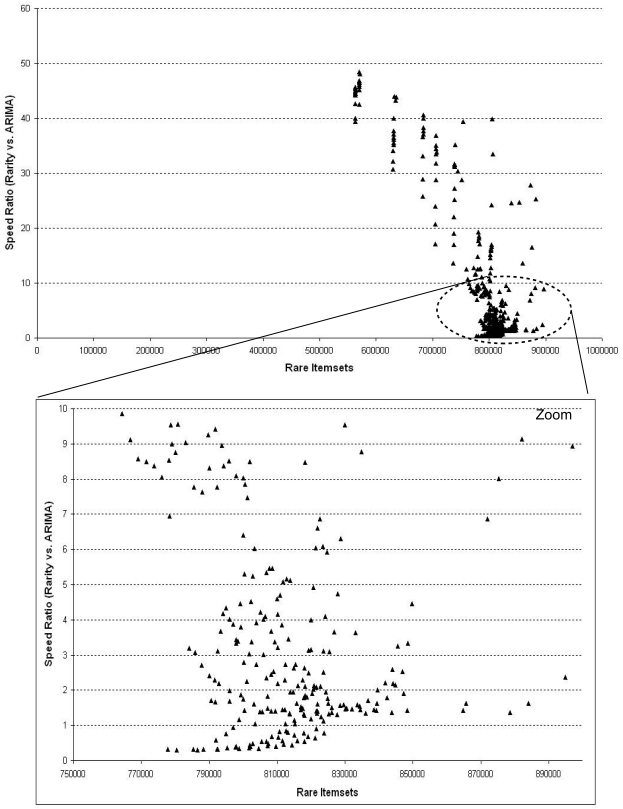

Figure 13. Extremely rare itemsets. Speed ratio: Rarity vs. ARIMA

REFERENCES

[1] G. M. Weiss, "Mining with rarity: a unifying framework," *SIGKDD Explor. Newsl.*, vol. 6, no. 1, pp. 7–19, 2004.

[2] B. Liu, W. Hsu, and Y. Ma, "Mining association rules with multiple minimum supports," in *KDD '99: Proc. of the 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data mining*. New York, NY, USA: ACM, 1999, pp. 337–341.

[3] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *SIGMOD '93: Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of data*. New York, NY, USA: ACM, 1993, pp. 207–216.

[4] G. Yang, "The complexity of mining maximal frequent itemsets and maximal frequent patterns," in *KDD '04: Proc. of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data mining*. New York, NY, USA: ACM, 2004, pp. 344–353.

[5] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB'94, Proc. of 20th Int. Conf. on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. Morgan Kaufmann, 1994, pp. 487–499.

[6] J. Janas, "An enhanced a priori algorithm for mining multidimensional association rules," in *Information Technology Interfaces, 2003. ITI 2003. Proc. of the 25th Int. Conf. on*, June 2003, pp. 193–198.

[7] R. Agrawal and J. C. Shafer, "Parallel mining of association rules," *IEEE Trans. on Knowl. and Data Eng.*, vol. 8, no. 6, pp. 962–969, 1996.

[8] A. Savasere, E. Omiecinski, and S. B. Navathe, "An efficient algorithm for mining association rules in large databases," in *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 432–444.

[9] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Min. Knowl. Discov.*, vol. 8, no. 1, pp. 53–87, 2004.

[10] Y. S. Koh and N. Rountree, "Finding sporadic rules using apriori-inverse," in *PAKDD*, ser. LNCS, T. B. Ho, D. W.-L. Cheung, and H. Liu, Eds., vol. 3518. Springer, 2005, pp. 97–106.

[11] L. Szathmary, A. Napoli, and P. Valtchev, "Towards rare itemset mining," in *ICTAI '07: Proc. of the 19th IEEE Int. Conf. on Tools with Artificial Intelligence*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 305–312.