

Self-similar Mining of Time Association Rules

Daniel Barbará

George Mason University
Department of Information and Software Engineering
Fairfax, VA 22303
dbarbara@gmu.edu

and

Ping Chen

CMS Dept. University of Houston-Downtown
1 Main St.
Houston, TX 77002
chenp@uhd.edu

and

Zohreh Nazeri

The MITRE Corporation
1820 Dolley Madison Boulevard
McLean, VA 22102-3481
nazeri@mitre.org

Abstract. Although the task of mining association rules has received considerable attention in the literature, algorithms to find time association rules are often inadequate, by either missing rules when the time interval is arbitrarily partitioned in equal intervals or by clustering the data before the search for high-support itemsets is undertaken. We present an efficient solution to this problem that uses the fractal dimension as an indicator of when the interval needs to be partitioned. The partitions are done with respect to every itemset in consideration, and therefore the algorithm is in a better position to find frequent itemsets that would have been missed otherwise. We present experimental evidence of the efficiency of our algorithm both in terms of rules that would have been missed by other techniques and also in terms of its scalability with respect to the number of transactions and the number of items in the data set.

KEYWORD: temporal association rules, fractal dimension, intrusion detection

1 Introduction

Association rules have received a lot of attention in the data mining community since their introduction in [1]. Association rules are rules of the form $X \longrightarrow Y$ where X and Y are sets of attribute-values, with $X \cap Y = \emptyset$ and $\|Y\| = 1$. The set X is called the antecedent of the rule while the item Y is called consequent. For example, in a market-basket data of supermarket transactions, one may find that customers who buy *milk* also buy *honey* in the same transaction, generating the rule $milk \longrightarrow honey$. There are two parameters associated with a rule: *support* and *confidence*. The rule $X \longrightarrow Y$ has *support* s in the transaction set T if $s\%$ of transactions in T contain $X \cup Y$. The rule $X \longrightarrow Y$ has *confidence* c if c of transactions in T that contain X also contain Y . The most difficult and dominating part of an association rules discovery algorithm is to find the itemsets $X \cup Y$, that have strong support. (Once an itemset is deemed to have strong support, it is an easy task to decide which item in the itemset can be the consequent by using the confidence threshold.)

Algorithms for discovering these “classical” association rules are abundant [2, 3]. They work well for data sets drawn from a domain of values with no relative meaning (categorical values). However, when applied to interval data, they yield results that are not very intuitive. Algorithms to deal with interval data have appeared in the literature. The first algorithm ever proposed (by Srikant and Agrawal [11]), defined the notion of *quantitative association rules* as rules where the predicates may be equality predicates ($Attribute = v$) or range predicates ($v_1 \leq Attribute \leq v_2$). The problem that arises then is how to limit the number of ranges (or intervals) that must be considered: if intervals are too large, they may hide rules inside portion of the interval, and if intervals are too small they may not get enough support to be considered.

Srikant and Agrawal deal with the problem by doing an Equi-depth initial partitioning of the interval data. In other words, for a depth d , the first d values (in order) of the attribute are placed in one interval, the next d in a second interval, and so on. No other considerations, such as the density of the interval or the distance between the values that fall in the same interval are taken.

Miller and Yang in [9] point out the pitfalls of Equi-depth partitioning. In short, intervals that include close data values (e.g., time values in an interval [9 : 00, 9 : 30]) are more meaningful than those intervals involving distant values (e.g., an interval [10 : 00, 21 : 00]). Notice that both types of intervals can coexist in an Equi-depth partitioning schema, since the population of values can be more dense in the closely tight intervals than in the loosely tight intervals. As the authors in [9] point out, it is less likely that a rule involving a loosely tight interval (such as [10 : 00, 21 : 00]) will be of interest than rules that involve the closely tight intervals. Srikant and Agrawal [11] had pointed out this problem, mentioning that their Equi-depth partitioning would not work well with skewed data, since it would separate close values that exhibit the same behavior.

Miller and Yang proceed to establish a guiding principle that we include here for completeness:

Goal 1. [9] In selecting intervals or groups of data to consider, we want a measure of interval quality that reflects the distance between data points.

To achieve this goal, Miller and Yang, defined a more general form of association rules with predicates expressing that an attribute (or set of attributes) ought to fall within a given subset of values. Since considering all possible subsets is intractable, they use clustering [6] to find subsets that “make sense” by containing a set of attributes that are sufficiently close. A cluster is defined in [9] as a set of values whose “diameter” (average pairwise distance between values) and whose “frequency” (number of values in the cluster) are both bounded by pre-specified thresholds. They propose an algorithm that uses BIRCH [15] to find clusters in the interval attribute(s) and use the clusters so found as “items” which with the items of categorical nature are fed into the a-priori algorithm [2] to find high-support itemsets and rules. So, an itemset found in this way could take the form $([9 : 00, 9 : 30], \textit{bread})$ meaning that people buy their bread frequently between 9:00 and 9:30 am. The interval $[9 : 00, 9 : 30]$ would have been found as constituting a cluster by feeding all the time values in the data set to BIRCH.

We believe that Miller’s technique still falls short of a desirable goal. The problem is that by clustering the interval attribute alone, without taking into consideration the other (categorical) values in the data set, one can fail to discover interesting rules. For instance, it is possible that while the itemset $([9 : 00, 9 : 30], \textit{bread})$ is a frequent one, the itemset $([9 : 00, 9 : 15], \textit{bread}, \textit{honey})$ is also frequent. Using clustering in the time attribute before considering the other items in the data set may lead to decide that the interval $[9 : 00, 9 : 30]$ is to be used as an item and therefore the second itemset above will never be found as frequent. Another problem is that invoking a clustering algorithm to cluster the interval values along the itemset that we are considering (say, for instance, *bread* and *honey*), would be prohibitively expensive.

In [14] Apriori algorithm was extended to find association rules which satisfy minimum support and confidence within a user-specified interval. In [13], the authors proposed to use a density-based clustering algorithm to find interval clusters first, then generate temporal association rules with Apriori algorithm. This method requires at least two scans of a data set, which may still be too expensive for large data sets. More seriously, a pre-clustering step assumes all association rules share the same temporal clusters, which may cause the problems similar to Miller’s method as discussed above. Fortunately, we have devised a method that can do the partitioning of the interval data in an *on-line* fashion, i.e, while we are trying to compute the support of the non-interval items of the itemset (*bread* and *honey*). This method uses the notion of *fractal dimension* to produce a natural partitioning of the set of interval data. Although we conduct our experiments only on temporal interval data, our method can be applied to any data sets with one or multiple interval attributes.

The rest of the paper is divided as follows. Section 2 briefly reviews the background on fractal properties of data sets. Section 3 offers a motivating example and our algorithm. In section 4 we show the experimental results of applying our

technique to a real data set. Finally, Section 5 gives the conclusions and future work.

2 Fractal dimension

Nature is filled with examples of phenomena that exhibit seemingly chaotic behavior, such as air turbulence, forest fires and the like. However, under this behavior it is almost always possible to find *self-similarity*, i.e. an invariance with respect to the scale used. The structures that exhibit self-similarity over every scale are known as *fractals* [8]. On the other hand, many data sets, that are not fractal, exhibit self-similarity over a range of scales. (In fact, self-similarity is **the property** that we are interested in exploiting for our algorithm.)

Fractals have been used in numerous disciplines (for a good coverage of the topic of fractals and their applications see [10]). In the database area, fractals have been successfully used to analyze selectivity estimation [4], dimensionality reduction [12], etc..

Self-similarity can be measured using the *fractal dimension*. Loosely speaking, the fractal dimension measures the number of dimensions “filled” by the object represented by the data set. In truth, there exists an infinite family of fractal dimensions. By embedding the data set in an n -dimensional grid which cells have sides of size r , we can count the frequency with which data points fall into the i -th cell, p_i , and compute D_q , the generalized fractal dimension [5], as shown in Equation 1.

$$D_q = \begin{cases} \frac{\partial \log \sum_i p_i \log p_i}{\partial \log r} & \text{for } q = 1 \\ \frac{1}{q-1} \frac{\partial \log \sum_i p_i^q}{\partial \log r} & \text{otherwise} \end{cases} \quad (1)$$

Among the dimensions described by Equation 1, the *Hausdorff fractal dimension* ($q = 0$), the *Information Dimension* ($\lim_{q \rightarrow 1} D_q$), and the *Correlation dimension* ($q = 2$) are widely used. The Information and Correlation dimensions are particularly useful for data mining, since the numerator of D_1 is Shannon’s entropy, and D_2 measures the probability that two points chosen at random will be within a certain distance of each other. Changes in the Information dimension mean changes in the entropy and therefore point to changes in trends. Equally, changes in the Correlation dimension mean changes in the distribution of points in the data set.

The traditional way to compute fractal dimensions is by means of the box-counting plot. For a set of N points, each of D dimensions, one divides the space in grid cells of size r (hypercubes of dimension D). If $N(r)$ is the number of cells occupied by points in the data set, the plot of $N(r)$ versus r in log-log scales is called the *box-counting plot*. The negative value of the slope of that plot corresponds to the Hausdorff fractal dimension D_0 . Similar procedures are followed to compute other dimensions, as described in [7].

To clarify the concept of box-counting, let us consider the famous example of George Cantor’s dust, constructed in the following manner. Starting with the

closed unit interval $[0,1]$ (a straight-line segment of length 1), we erase the open middle third interval $(\frac{1}{3}, \frac{2}{3})$ and repeat the process on the remaining two segments, recursively. Figure 1 illustrates the procedure. The “dust” has a length measure of zero and yet contains an uncountable number of points. The Hausdorff dimension can be computed the following way: it is easy to see that for the set obtained after n iterations, we are left with $N = 2^n$ pieces, each of length $r = (\frac{1}{3})^n$. So, using a unidimensional box size with $r = (\frac{1}{3})^n$, we find 2^n of the boxes populated with points. If, instead, we use a box size twice as big, i.e., $r = 2(\frac{1}{3})^n$, we get 2^{n-1} populated boxes and so on. The log-log plot of box population vs. r renders a line with slope $D_0 = -\log 2 / \log 3 = -0.63\dots$. The value 0.63 is precisely the fractal dimension of the Cantor’s dust data set.



Figure 1: The construction of the Cantor dust. The final set has fractal (Hausdorff) dimension 0.63.

In what follows of this section we present a motivating example that illustrates how the fractal dimension can be a powerful way for separating data points. Figure 2 shows the effect of superimposing two different Cantor dust sets. After erasing the open middle interval which results of dividing the original line in three intervals, the leftmost interval gets divided in 9 intervals, and only the alternative ones survive (5 in total). The rightmost interval gets divided in three, as before, erasing the open middle interval. The result is that if one considers grid cells of size $\frac{1}{3 \times 9^n}$ at the n -th iteration, the number of occupied cells turns out to be $5^n + 6^n$. The slope of the log-log plot for this set is $D'_0 = \lim_{n \rightarrow \infty} (\log(5^n + 6^n)) / \log(3 \times 9^n)$. It is easy to show that $D'_0 > D_0$, where $D_0 = \log 2 / \log 3$ is the fractal dimension of the rightmost part of the data set (the Cantor dust of Figure 1). Therefore, one could say that the inclusion of the leftmost part of the data set produces a change in the fractal dimension and this subset is therefore “anomalous” with respect to the rightmost subset (or vice-versa). From the clustering point of view, for a human being it is easy to recognize the two Cantor sets as two different clusters. And, in fact, an algorithm that exploits the fractal dimension (as the one presented in this paper) will indeed separate these two sets as different clusters. Any point in the right Cantor set would change the fractal dimension of the left Cantor set if included in the left cluster (and vice-versa). This fact is exploited by our algorithm (as we shall explain later) to place the points accordingly.

Figure 2: A “hybrid” Cantor dust set. The final set has fractal (Hausdorff) dimension larger than that of the rightmost set (which is the Cantor dust set of Figure 1).

3 Our approach

3.1 Motivation

To motivate our technique, consider the data set shown in Figure 3, where timestamped customer transactions (baskets) are shown.

Time	Items				Time	Items			
9:00	bread	honey	milk	cookies	12:45	honey	diapers		
9:01	bread	honey	milk	cookies	3:25	beer			
9:02	bread	honey	cookies	milk	14:10	poultry			
9:05	bread	honey	poultry	milk	18:00	beer	diapers		
9:07	bread	honey	cereal	cookies	18:10	beer	diapers		
9:10	bread	honey	beer	diapers	18:22	beer	diapers		
9:12	bread	honey	poultry		18:24	bread	honey	milk	cookies
9:15	bread	honey			18:25	bread	honey	milk	cookies
9:19	bread	beer	diapers		18:26	bread	honey	cookies	milk
9:20	bread	poultry			18:29	bread	honey	poultry	milk
9:22	cereal	milk			18:31	bread	honey	cereal	cookies
9:25	bread				18:34	bread	honey		
9:27	bread	honey			18:35	beer	diapers		
9:30	bread	cereal			18:36	bread	honey	poultry	
11:00	meat	beer			18:39	bread	honey		
12:00	poultry	beer			18:40	beer	diapers	honey	bread
					20:40	honey	bread		

Figure 3: A timestamped set of baskets.

An Equi-depth algorithm such as the one presented in [11] may run the risk of breaking the obvious temporal interval $[9 : 00, 9 : 30]$ and miss the fact that the item *bread* had high support for this entire period of time. Miller and Yang’s algorithm will find that the attribute Time contains at least two clusters: points in the interval $[9 : 00, 9 : 30]$ and points in the interval $[18 : 00, 18 : 40]$. (The rest of the time points may be grouped in other clusters or considered outliers.) Using the first cluster and with support threshold equal to $0.16 (\frac{4}{24})$, Miller and Yang’s algorithm would find, among others, high support itemsets $([9 : 00, 9 : 30], \textit{bread}, \textit{honey})$, $([9 : 00, 9 : 30], \textit{milk}, \textit{cookies})$ and $([18 : 00, 18 : 40], \textit{beer}, \textit{diapers}, \textit{bread}, \textit{honey})$. The algorithm would have missed the itemset $([9 : 00, 9 : 15], \textit{bread}, \textit{honey})$ that has support 0.33, and the itemset $[9 : 00, 9 :$

05], *milk, cookies*) with support 0.16. The reason is that this algorithm would have fed the item $[9 : 00, 9 : 30]$ to the a-priori algorithm, without further consideration for subintervals within it, like the $[9 : 00, 9 : 15]$ subinterval, which in this case is a better “fit” for the items (*bread, honey*). (Or the $[9 : 00, 9 : 05]$ interval for the items (*milk, cookies*).)

As pointed out before, it would be extremely expensive to invoke clustering of each itemset under consideration due to too many scans of a dataset. However, using the fractal dimension we can design a simple, incremental way of deciding when to break the interval data for any itemset under consideration. The central idea is to modify a-priori in such a way that while the algorithm is measuring the support for an itemset whose items are categorical (e.g., (*bread, honey*)), we incrementally compute the fractal dimensions of all (possibly multidimensional, since the number of interval attributes in the clusters can be more than 1) clusters that include each of the interval attributes where the itemset shows up. To illustrate, Figure 4 shows the configuration of this data set using the data of Figure 3, while considering the support of the itemset (*bread, honey*). As each of the rows in the data set of Figure 3 is considered, the rows of the cluster (so far, we only have two clusters $[9 : 00, 9 : 15]$ and $[18 : 24, 18 : 39]$ which has only one interval attribute and we are going to process point 18:40) shown in Figure 4 can be built and the fractal dimension of the cluster of rows seen so far can be computed. Figure 5 shows the two box-counting plots of the data set of Figure 4. The first plot includes all the points up to time 9:15 (the first cluster), while the second includes the first cluster plus point 18:40 currently under consideration. The first straight region shows a slope of -0.2 for a fractal dimension of 0.2. The addition of the 18:40 point decreases the fractal dimension to 0.16. Similarly we compute the fractal dimension of second cluster $[18 : 24, 18 : 39]$, and fractal dimension of $[18 : 24, 18 : 39]$ plus point 18:40, and the difference is smaller than 0.01. So point 18:40 is added to the second cluster and the new second cluster will be $[18 : 24, 18 : 40]$. When we input next transaction which have itemset (*bread, honey*) with time 20:40, which will cause big changes of fractal dimension to both existing clusters, which indicates that we should start a new cluster with the new point 20:40. A big change of fractal dimension (suppose we set the change threshold as 0.01) is a good indication for a breaking point in the interval data, and a good place to compute the support of the itemset composed by the interval considered so far ($[18 : 24, 18 : 40]$) and the other two items (*bread, honey*). If the support is equal or exceeds the threshold, then this itemset would be reported by the algorithm. In summary, if the minimum change of fractal dimension for all existing clusters is less than a preset threshold, we should put the new point into the cluster whose fractal dimension is changed the least, otherwise, we should start a new cluster. For the itemset (*milk, cookies*) there is a change after the point 9 : 07. (The itemset receives its strong support before that time.) And for the itemset *beer, diapers*, there is a clear change after the point 18 : 00. Indeed, after that point a new interval opens in which the itemset receives strong support. Thus, a potential high support itemset would be ($[18 : 00, 18 : 40]$, *beer, diapers*). (This, in fact would reveal an interesting nugget:

that this pair of items are bought frequently after work hours, for example.) Again, we could not have found these itemsets ($([9 : 00, 9 : 05], \textit{milk}, \textit{cookies})$, $([9 : 00, 9 : 15], \textit{bread}, \textit{honey})$ and $([18 : 00, 18 : 40], \textit{beer}, \textit{diapers})$) by pre-clustering the time attribute.

By our method we do not require any order in interval attributes, and it can be readily applied to a data set with multiple interval attributes.

Time	9:00	9:01	9:02	9:05	9:07	9:10	9:12	9:15	18:24
	18:25	18:26	18:29	18:31	18:34	18:36	18:39	18:40	20:40

Figure 4: The data set resulting selecting the values of the interval attribute (Time) for which the itemset $(\textit{bread}, \textit{honey})$ is present in the transaction.

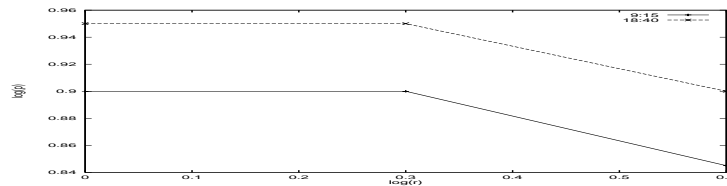


Figure 5: Loglog plot of the data set shown in Figure 4 up to the points 9:15 and 18:40 respectively. The slope of the 9:15 curve is -0.2 , indicating a fractal dimension 0.2 , while the slope of the 18:40 curve is -0.16 , for a fractal dimension of 0.16

3.2 Algorithm

The pseudo-code for the our algorithm is shown in Figure 6. Line 1 selects all the 1-itemsets, as in a-priori, but considering only the categorical (non-interval) items. (Every L_k will contain only categorical itemsets.) Then the algorithm iterates until no more candidates exist (Line 2), generating the next set of candidates from L_{k-1} (as in a-priori), and for every candidate I in that set initializing the multidimensional set M_I as empty and making the lower bound of the interval for I , low_I equal to $-\infty$. (This will be changed as soon as the first occurrence of I is spotted in the data set, in Lines 14-15.) For each transaction (basket) t in D , if the transaction contains I , a new row is added to the set M_I containing the interval attribute values for t in Line 13. (The notation $interval(t)$ indicates extracting the interval data from a tuple t .) Regardless of whether the item is in t or not, the overall count for the item interval $M_I.totalcount$ is increased in Line 10. This count will serve as the denominator when the support of I within the interval needs to be computed. Both, the overall count ($I.count$) and interval count ($M_I.count$) for the itemset are increased (line 12). The fractal dimension of the set M_I is then computed (Line 16), and if a significant change is found, a new interval is declared (line 17). The notation $last(M_I, interval(t))$ indicates

the interval data point before the current $interval(t)$ in M_I . Line 19 tests if the itemset composed by this interval and I has enough support. Notice that both the $M_I.count$ and $M_I.totalcount$ need to be reduced by one, since the current transaction is not part of the interval being tested. If the support is greater than or equal to $minsup$, the itemset is placed in the output. (Line 19.) Line 20 re-initializes the counts. Line 22 is the classic test for the overall support in a-priori, with N being the total number of transactions in the data set.

- (1) $L_1 = \{ \text{large 1-itemsets, from categorical values} \};$
- (2) for ($k = 2; L_{k-1} \neq \emptyset; k++$) do
 - (3) $C_k = \text{generate}(L_{k-1})$
 - (4) for every I in C_k
 - (5) make $M_I = \emptyset$ for every I in C_k
 - (6) make $low_I = -\infty$
 - (7) while there are transactions in D do
 - (8) consider next transaction t in D
 - (9) for every candidate in I in C_k
 - (10) $M_{Ic}.totalcount++$
 - (11) if I is in t
 - (12) make $M_I.minfdchange = +\infty$
 - (13) for every interval cluster c in I in C_k
 - (14) $I.count++$, $M_{Ic}.count++$
 - (15) add row with $interval(t)$ to M_{Ic}
 - (16) if $low_I = -\infty$
 - (17) $low_I = interval(t)$
 - (18) compute $F(M_{Ic}) = fd(M_{Ic})$
 - (19) if the fractal dimension change is less than $M_I.minfdchange$
 - (20) assign this smaller change to $M_I.minfdchange$
 - (21) assign the number of the cluster with the smallest change to j
 - (22) if $M_I.minfdchange$ is larger than τ
 - (23) if $\frac{M_I.count - 1}{M_I.totalcount - 1} \geq minsup$
 - (24) output itemset $[low_I, last(M_I, interval(t))], I$
 - (25) $M_{I_j} = interval(t)$, $low_I = interval(t)$,
 $M_I.count = M_I.totalcount = 1$
 - (26) end
 - (27) $L_k = \{ I \text{ in } C_k \mid \frac{I.count}{N} \geq minsup \}$
 - (28) output items in L_k
 - (29) end

Figure 6: Our algorithm.

4 Experiments

The experiments reported in this section were conducted over a Sun Workstation Ultra2 with 500 MB. of RAM, running Solaris 2.5. The experiment was done over a real data set of sniffed connections to a network, collected for intrusion detection purposes. The data is collected over MITRE's network for the purpose of intrusion detection. The attributes in the collected data are: Time (one field containing both date and time), source IP address, destination IP address, duration of the connection, type of the connection (TELNET, Login,), and name of the sensor that reported the connection. All IP addresses and sensor names in the data are changed to symbolic names such as sensor1 or 111.11.1.1 IP address. A total of 4000 data records were used in this experiment. These records cover about 40 hours of network events (telnet, login, etc.). The Time attribute is the quantitative or interval attribute.

We tried the Equi-depth approach [11] and our algorithm on this data. For our algorithm, the parameters needed from the user are minimum support, minimum window size, and a threshold for the Fractal dimension. The parameters needed for the Equi-depth approach are minimum support, minimum confidence, and maximum support. We used the same minimum support for both algorithms. This parameter then has equal impact on both approaches and this is desirable for the comparison of the rules generated by each. For the Equi-depth approach, we used a minimum confidence of 50% and after trying different values of maximum support we picked the one that gave the best results, that is 12%. Using the maximum support in the equations suggested in Srikant and Agrawal's paper [11], the number of intervals is 8. Dividing the 24 hours by 8, our intervals are 0:00-2:59, 3:00-5:59, 6:00-8:59, and so on.

For our algorithm, we tried different thresholds values. The threshold specifies the amount of change that should be seen in the fractal dimension before the algorithm draws a line and starts a new window. The bigger this number is, the less number of rules are generated. The minimum window size forces the window to have at least the specified minimum number of records regardless of the change in fractal dimension. A bigger window size reduces the execution time but might miss some rules.

The results generated by each approach in our experiment are shown in Figures 7 and 8 and compared below. We show the output rules in groups so they can be matched and compared easily. For example, rules 1 through 5 in the Equi-depth output are comparable to rule 1 in the Fractals output. To read the rules, Equi-depth rule 1 shows that 91% of network connections on July 25 between 9am to 12pm are TELNET connections. Fractals rule 1 shows 89% of connection from 12am (midnight) to 9:06:43 am on July 25 are TELNET connections; it also shows that this is 11% of all connections in the data set. Note that Fractal rules are all associated with the interval attribute (Time) while the Equi-depth rules show such associations only if they make the minimum support and minimum confidence. As a result, the Equi-depth rules 6 and 7 show associations between source IP= 111.11.1.1 and destination IP = 222.22.2.2 but their associations with the interval attribute, as shown by Fractals rule 2, are missed. An itemset's

association with the interval attribute, Time in this case, could be an important one. For example, the fact that 26% of connections after business hours from 16:04 to midnight (as shown by third line under Fractals rule 2) have occurred between source IP address 111.11.1.1 and destination IP address 222.22.2.2, is an interesting rule which is worth further investigation (since normally one expects less number of connections after business hours).

Fractals rules 14 through 21 are associations found by the Fractals approach and not found by the Equi-depth approach. Some of these rules could be important. For example Fractals rule 16 shows connections with 0 seconds connection duration. A connection with very short duration is an important characteristic in intrusion detection. The first line under rule 16 shows 19% of the connections between midnight and 9 in the morning (on July 25) have occurred with a duration of 0. Again this is a rule that is worth further investigation. Note that this association, $[7/25 - 00 : 00 : 00 - 7/25 - 09 : 06 : 43] - > duration = 0$, has an overall support of 2% and would never make it through the filter of 10% minimum support.

Rule Number	Rule	confidence
1.	$Time = 7/25 - [09 - 12] - > event = TELNET$	91%
2.	$Time = 7/25 - [12 - 15] - > event = TELNET$	90%
3.	$Time = 7/25 - [15 - 18] - > event = TELNET$	93%
4.	$Time = 7/26 - [09 - 12] - > event = TELNET$	90%
5.	$Time = 7/26 - [12 - 15] - > event = TELNET$	90%
6.	$dstIP = 222.22.2.2 - > srcIP = 111.11.1.1$	100%
7.	$srcIP = 111.11.1.1 - > dstIP = 222.22.2.2$	100%

Figure 7: Results of the Equi-depth approach with minimum support = 10%, minimum confidence = 50%, maximum Support = 12%, and number of intervals = 8 (every 3 hours). Only first 7 rules are shown due to page limit.

5 Conclusions

We have presented here a new algorithm to efficiently find association rules for data sets that contain one dimension of interval values. Previous algorithms dealt with this problem by dividing the interval data in Equi-depth intervals, risking missing some important rules, or by clustering the interval data before using the classical a-priori approach, and using the intervals found by the clustering algorithm as items. In doing so, these intervals remain defined for the rest of the algorithm, independently on whether for certain itemsets it would have been more adequate to consider a subinterval. We have successfully implemented the code shown in Figure 6 and conducted experiments in synthetic and real data sets with it. Our results show that this technique can efficiently find frequent itemsets that were missed by previous approaches. Also, our results demonstrate that we are able to find important associations that could be missed by previous

Rule Number	Rule(window support, overall support)
1.	event=TELNET [7/25-00:00-7/25-09:06](89%,11%),[7/25-09:06-7/25-11:13](90%,11%) [7/25-11:13-7/25-16:04](91%,22%),[7/25-16:04-7/26-00:01](93%,11%) [7/26-00:01-7/26-12:17](91%,22%),[7/26-12:17-7/26-16:22](90%,11%)
2.	srcIP=111.11.1.1, dstIP=222.22.2.2 [7/25-09:06-7/25-11:13](16%,2%), [7/25-11:13-7/25-16:04](21%,5%) [7/25-16:04-7/26-00:01](26%,3%),[7/26-00:01-7/26-12:17](15%,3%)
Rule 3 through 13 are omitted due to page limit.	
14.	dstIP=222.22.2.2 [7/25-09:06-7/25-11:13](16%,2%),[7/25-11:13-7/25-16:04](21%,5%) [7/25-16:04-7/26-00:01](26%,3%),[7/26-00:01-7/26-12:17](15%,3%)
15.	srcIP=111.11.1.1 [7/25-09:06-7/25-11:13](16%,2%),[7/25-11:13-7/25-16:04](21%,5%) [7/25-16:04-7/26-00:01](26%,3%),[7/26-00:01-7/26-12:17](15%,3%)
16.	duration=0 [7/25-00:00-7/25-09:06](19%,2%),[7/25-09:06-7/25-11:13](14%,1%) [7/25-11:13-7/25-16:04](12%,3%),[7/25-16:04-7/26-00:01](12%,1%) [7/26-00:01-7/26-12:17](13%,3%),[7/26-12:17-7/26-16:22](11%,1%)
17.	sensor1 [7/25-00:00-7/25-09:06](62%,7%),[7/25-09:06-7/25-11:13](48%,6%) [7/25-11:13-7/25-16:04](41%,10%),[7/25-16:04-7/26-00:01](44%,5%) [7/26-00:01-7/26-12:17](45%,11%),[7/26-12:17-7/26-16:22](66%,8%)
18.	sensor3 [7/25-00:00-7/25-09:06](36%,4%),[7/25-09:06-7/25-11:13](42%,5%) [7/25-11:13-7/25-16:04](54%,13%),[7/25-16:04-7/26-00:01](54%,6%) [7/26-00:01-7/26-12:17](52%,13%),[7/26-12:17-7/26-16:22](29%,3%)
19.	duration=0, sensor1 [7/25-00:00-7/25-09:06](11%,1%),[7/25-09:06-7/25-11:13](9%,1%) [7/25-11:13-7/25-16:04](5%,1%),[7/26-00:01-7/26-12:17](6%,1%) [7/26-12:17-7/26-16:22](8%,1%)
20.	duration=0, sensor3 [7/25-11:13-7/25-16:04](6%,1%),[7/26-00:01-7/26-12:17](7%,1%)
21.	duration=0, event=TELNET [7/25-00:00-7/25-09:06](11%,1%),[7/25-09:06-7/25-11:13](8%,1%) [7/25-11:13-7/25-16:04](6%,1%),[7/25-16:04-7/26-00:01](8%,1%) [7/26-00:01-7/26-12:17](7%,1%)

Figure 8: Results of our algorithm with minimum support = 10.00%, minimum window size = 500, FD threshold = 0.02.

algorithms for interval data. We are currently studying issues such as how to optimize our algorithm, perhaps taking advantage of the previous fractal dimension computation when we need to re-compute it adding a point for an itemset.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets in large databases. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data, Washington, DC*, 1993.
2. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri. Fast Discovery of Association Rules. In *Advances in Knowledge Discovery and Data Mining*, U. Fayyad, G. Shapiro, P. Smyth, R. Uthurusamy, eds. AAAI press, 1996.
3. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, 2000.
4. A. Belussi and C. Faloutsos. Estimating the Selectivity of Spatial Queries Using the ‘Correlation’ Fractal Dimension. In *Proceedings of the International Conference on Very Large Data Bases*, pp. 299-310, September 1995.
5. P. Grassberger and I. Procaccia. Characterization of Strange Attractors. *Physical Review Letters*, 50(5):346–349, 1983.
6. A. Jain, R.C. Dubes. Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs, New Jersey 1988.
7. L.S. Liebovitch and T. Toth. A Fast Algorithm to Determine Fractal Dimensions by Box Counting. *Physics Letters*, 141A(8), 1989.
8. B.B. Mandelbrot. *The Fractal Geometry of Nature*. W.H. Freeman, New York, 1983.
9. R.J. Miller and Y. Yang. Association Rules over Interval Data. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data, Phoenix, Arizona*, 1997.
10. M. Schroeder. *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise*. W.H. Freeman, New York, 1991.
11. R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data, Montreal, Canada*, 1996.
12. C. Traina Jr., A. Traina, L. Wu and C. Faloutsos, Fast feature selection using the fractal dimension. In *Proceedings of the XV Brazilian Symposium on Databases (SBBD), Paraiba, Brazil*, October 2000.
13. W. Wang, J. Yang, R. Muntz, TAR: Temporal Association Rules on Evolving Numerical Attributes In *Proceedings of 17th International Conference on Data Engineering April 02-06, 2001 Heidelberg, Germany* 2001.
14. X. Chen, I. Petrounias Discovering Temporal Association Rules: Algorithms, Language and System In *Proceedings of 16th International Conference on Data Engineering San Diego, California*, February 28-March 03, 2000.
15. R. Zhang, R. Ramakrishnan and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data, Montreal, Canada*, 1996. Pages 103-114.