Discovering Numeric Association Rules via Evolutionary Algorithm

Jacinto Mata[1], José-Luis Alvarez[1], and José-Cristobal Riquelme[2]

[1] Dpto. Ingeniería Electrónica, Sistemas Informáticos y Automática
Universidad de Huelva, Spain
{mata,alvarez}@uhu.es
[2] Dpto. Lenguajes y Sistemas Informáticos
Universidad de Sevilla, Spain
riquelme@lsi.us.es

Abstract. Association rules are one of the most used tools to discover relationships among attributes in a database. Nowadays, there are many efficient techniques to obtain these rules, although most of them require that the values of the attributes be discrete. To solve this problem, these techniques discretize the numeric attributes, but this implies a loss of information. In a general way, these techniques work in two phases: in the first one they try to find the sets of attributes that are, with a determined frequency, within the database (*frequent itemsets*), and in the second one, they extract the association rules departing from these sets. In this paper we present a technique to find the *frequent itemsets* in numeric databases without needing to discretize the attributes. We use an evolutionary algorithm to find the intervals of each attribute that conforms a *frequent itemset*. The evaluation function itself will be the one that decide the amplitude of these intervals. Finally, we evaluate the tool with synthetic and real databases to check the efficiency of our algorithm.

1 Introduction

Association rules were introduced in [1] as a method to find relationships among the attributes of a database. By means of these techniques a very interesting qualitative information with which we can take later decisions can be obtained. In general terms, an association rule is a relationship between attributes in the way $C_1 \Rightarrow C_2$, where C_1 and C_2 are pair conjunctions (attribute-value) in the way $A = v$ if it is a discrete attribute or $A \epsilon [v_1, v_2]$ if the attribute is continuous or numeric. Generally, the antecedent is formed by a conjunction of pairs, while the consequent usually is a unique attribute-value pair.

In most of databases can appear a rather high number of rules of this kind, so it is essential to define some measures that allow us to filter only the most significant ones. The most used measures to define the *interest* of the rules were described in [1]:

- **support.** It is a statistical measure that indicates the ratio of the population that satisfies both the antecedent and the consequent of the rule. A rule $R : C_1 \Rightarrow C_2$ has a support s, if a $s\%$ of the records of the database contain C_1 and C_2.
- **confidence**. This measure indicates the relative frequency of the rule, that is, the frequency with which the consequent is fulfilled when it is also fulfilled the antecedent. A rule $R : C_1 \Rightarrow C_2$ has a confidence c, if the $c\%$ of the records of the database that contain C_1 also contain C_2.

The goal of the techniques that search for association rules is to extract only those that exceed some minimum values of *support* and *confidence* that are defined by the user. The greater part of the algorithms that extract association rules work in two phases: in the first one they try to find the sets of attributes that exceed the minimum value of support and, in the second phase, departing from the sets discovered formerly, they extract the association rules that exceed the minimum value of confidence. Some of these algorithms can be seen on [2, 9, 13, 14, 7, 8, 12].

The first works on association rules were focused on marketing. In them the databases are transactions that represent the purchases made by the customers. Hence, each transaction is formed by a set of elements of variable size. These kind of rules use to be called *classic association rules*. and the nomenclature proposed for them is still being used for the different variants of association rules. The databases with which we will work, unlike these, will be relational tables, that is, will consist of a set of records or tuples formed by a fixed number of continuous attributes, as can be seen in figure 1.

assists	height	time	age	points
0.0888	201	36.02	28	0.5885
0.1399	198	39.32	30	0.8291
0.0747	198	38.8	26	0.4974
...	...	...	...	...
0.1276	196	38.4	28	0.5703

Fig. 1. Basketball database

In this paper we will use the definitions proposed in [1], adapting them to the databases with which we will work.

Definition 1. ***Itemset****. It is a set of attributes belonging to the database. Each itemset is formed by a variable number of attributes. An itemset formed by k attributes will be called k-itemset. In our case, an itemset is formed by pair (attribute-range of values)*

Definition 2. ***Frequent itemset****. It is that itemset that exceed the minimum value of support.*

Therefore, the problem of mining association rules consists, basically, in finding all the frequent itemsets and obtaining the rules departing from these sets. All the studies and researches are focused on the first phase, which is the most expensive, since the second one can be considered a simple and direct process. Most of the tools cited before work starting with the frequent itemsets of size 1 and joining them to conform frequent itemsets of a greater size in each step.

But in the real world there are numerous databases where the stored information is numeric. In these databases, attributes have thousand of possibilities of taking one value, by this reason the process described above is unthinkable from a computational point of view. Association rules obtained on numeric databases will be called *quantitative association rules*. The problem of mining quantitative association rules was first introduced in [15]. These rules are a variant of classic association rules where the value that the attribute takes in the rule is an interval instead of a discrete value. An example of this kind of rules is: *if height* ϵ $[196, 201]$ *and time* ϵ $[35.3, 37.8]$ *then assist* ϵ $[0.025, 0.076]$.

The basic idea of the algorithm presented in their work consists in dividing the range of each numeric attribute into intervals, treating them, from that moment onwards, as discrete attributes. That strategy is the same that have been followed by the diverse authors that have worked with numeric databases. Each of them uses different methods: clustering techniques, partition of the domain into intervals of the same size, techniques to merge adjacent intervals until reaching a maximum support, discretization by means of fuzzy sets, etc., but all of them have in common the fact that they need information a priori from the user. Some of these techniques can be consulted in [11, 16, 3].

The main problem of all of them lies in the fact that the data must be prepared before applying the tool. This preparation, either by means of the user or by means of an automatic process, conveys a loss of information because the rules will be only generated departing from the partitions previously created.

Our goal is to find association rules in numeric databases without the necessity of preparing previously the data. In order to get this objective we present a tool based in an evolutionary algorithm [4] that discovers the frequent itemsets in numeric databases. We have designed the evolutionary algorithm to find the intervals in each of the attributes that conforms a frequent itemset, in such a way that the fitness function itself is the one that decides the amplitude of the intervals.

2 A Motivation Example

In figure 2 we can see the result obtained by our algorithm for the basketball database. We have only represented two of the frequent itemsets found. The most important of our results with regard to formerly tools is the possibility of obtaining ranges with overlapping in different itemsets. For example, in the first itemset, the best interval for *height* attribute is [179,198], while in the second one, the best interval for this attribute is [175,196]. In the previously referenced techniques, the attributes are discretized before searching the itemsets. So, if

the discretization process finds the interval [179,198] for *height* attribute, the interval [175,196] can not appear in any itemset. This fact generates a loss of information. For example, if the minimum support is 30% and the discretization process has created the interval [179,198] for the *height* attribute, the second itemset would never be discovered because, probably, it would not exceed the minimum support or it would be smaller than 36.31%. Nevertheless, if their limits are slightly dynamically modified (we make it by means of mutations), the second itemset can also be discovered.

assist	[0.0721,0.2529]
height	[179,198]
age	[22,32]
Support	= 39.45 %

assist	[0.0978,0.277]
height	[175,196]
time	[19.02,38.97]
age	[22,31]
points	[0.3071,0.59]
Support	= 36.31 %

Fig. 2. 2 itemsets discovered in basketball database

3 Preliminaries

The tool presented in this paper is based on the evolutionary algorithm theory (EA). In order to find the optimal itemsets, that is, those with the best support without being their intervals excessively wide, we depart from a population where the individuals are potential itemsets. These individuals will be evolving by means of crossover and mutations, so that, at the end of the process, the individual with the best fitness will correspond with the "best" frequent itemset.

One of the problems we find when we work with EA theory is the convergence of all the individuals towards the same solution. In our case, this means that all the individuals evolve towards the same frequent itemset, that is, the individuals that conform the last generation provide, in practice, the same information. There are many techniques to solve this problem. Among them evolutionary algorithm with niches and iterative rule learning [5], which is the one used in our tool.

In this paper we develop only the first phase of a process of mining association rules, that is, the one that undertakes to find the frequent itemsets, because we use for the second phase some of the algorithm presented in the studies cited before.

4 Practical Implementation

As it was above, the core of this tool is an EA where the individuals are the possible itemsets we want to discover. In the following sections we will see the general

structure of the algorithm, the same that the fitness function, representation of the individuals and the meaning of the genetic operators.

4.1 GAR Algorithm

The GAR (Genetic Association Rules) algorithm is based in the theory of evolutionary algorithms and it is an extension of the GENAR algorithm presented in [10], that search directly for the association rules, so it is necessary to prepare the data to indicate to the tool which attributes form part of the antecedent and which one is the consequent. Nevertheless, this process is not necessary in GAR, because the algorithm finds the frequent itemsets and the rules are built departing from them.

```
algorithm GAR
1.  nItemset = 0
2.  while (nItemset < N) do
3.    nGen = 0
4.    generate first population P(nGen)
5.    while (nGen < NGENERATIONS) do
6.      process P(nGen)
7.      P(nGen+1) = select individuals of P(nGen)
8.      complete P(nGen+1) by crossover
9.      make mutations in P(nGen+1)
10.     nGen++
11.   end_while
12.   I[nItemset] = choose the best of P(nGen)
13.   penalize records covered by I[nItemset]
14.   nItemset++
15. end_while
end
```

Fig. 3. GAR algorithm

In figure 3 the structure of the algorithm is shown. The process is repeated until we obtain the desired number of frequent itemsets N. The first step consists in generating the initial population. The evolutionary algorithm takes charge of calculating the fitness of each individual and carries out the processes of selection, crossover and mutation to complete the following generation. At the end of the process, in step 12, the individual with the best fitness is chosen and it will correspond with one of the frequent itemsets that the algorithm returns. The operation made in step 13 is very important. In it, records covered by the obtained itemset in the previous step are penalized. Since this factor affects negatively to the fitness function we achieve that in the following evolutionary process the search space tends to not be repeated.

4.2 Structure of Individuals

Due to the nature itself of the problem to solve, that is, the fact that the value of the attributes are taken from continuous domain, we use real codification to represent the individuals. An individual in GAR is a k-itemset where each gene represents the maximum and minimum values of the intervals of each attribute that belongs to such k-itemset.

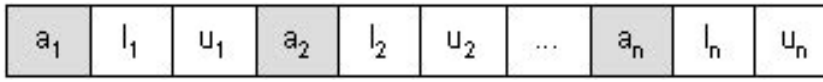

Fig. 4. Representation of an individual (n-itemset)

In general, the frequent itemsets are formed by a variable number of attributes, that is, for a database with n attributes there can be frequent itemsets from size 2 to size n, as can be seen in figure 4, where l_i and u_i are the limits of the intervals corresponding to the attribute a_i.

4.3 Initial Population

The generation of the initial population consists in the random creation of the intervals of each attribute that conforms the itemset. The number of attributes of each itemset is also chosen in a random way between 2 and the maximum number of attributes of the database. We condition the itemesets to cover at least a record of the database and that their intervals have a reduced size.

4.4 Genetic Operators

The genetic operators used in GAR are the usual ones, that is, selection, crossover and mutation. For the selection, we use an elitist strategy to replicate the individual with the best fitness. By means of the crossover operator we complete the rest of the population, choosing randomly, the individuals that will be combined to form new ones. From each crossover between two individuals two new ones are generated, and the best adapted will pass to the next generation. Given two individuals of the population $I = ([l_1, u_1], [l_3, u_3])$ and $I' = ([l'_1, u'_1], [l'_2, u'_2], [l'_3, u'_3])$, that are going to be crossed, the crossover operator generates the following two offspring:

$$O_1 = ([[l_1, u_1] \vee [l'_1, u'_1]], [[l_3, u_3] \vee [l'_3, u'_3]])$$
$$O_2 = ([[l'_1, u'_1] \vee [l_1, u_1]], [l'_2, u'_2], [[l'_3, u'_3] \vee [l_3, u_3]])$$

In figure 5 a possible result of the crossover operator for two itemsets of different size can be seen.

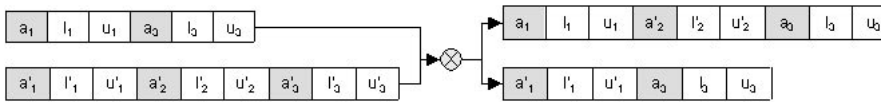

Fig. 5. Example of a crossover operation

The mutation operator consists in altering one or more genes of the individual, that is, in modifying the values of some of the intervals of a itemset. For each limit of the selected interval we have two possibilities, to increase or to decrease its value. In this way we achieved four possible mutations: to shift the whole interval to the left or to the right and to increase or to decrease its size.

Finally, a process of adjusting the chosen individual is carried out. This consists in decreasing the size of its intervals until the number of covered records be smaller than the records covered by the original itemset. Again, the goal of this post processing is to obtain more quality rules.

4.5 Fitness Function

As any evolutionary algorithm, GAR has a function implemented in order to evaluate the fitness of the individuals and to decide which are the best candidates in the following generations.

In our scenery, we look for the frequent itemsets with a larger support, that is, those that cover more records in the database. But, if we use this criterion as the only one to decide the limits of the intervals the algorithm will try to span the complete domain of each attribute. For this reason, it is necessary to include in the fitness function some measure to limit the size of the intervals.

The fitness function f for each individual is:

$$f(i) = covered - (marked * \omega) - (amplitude * \psi) + (nAtr * \mu) \quad (1)$$

The meaning of the parameters of the fitness function is the following:

- **covered.** It indicates the number of records that belong to the itemset that represent to the individual. It is a measure similar to support.
- **marked.** It indicates that a record has been covered previously by a itemset. We achieve with this that the algorithm tend to discover different itemsets in later searches. To penalize the records, we use a value that we call *penalization factor* (ω) to give more or least weight to the marked record, that is, we will permit more or least overlapping between the itemsets found depending on this value. This factor will be defined by the user.
- **amplitude.** This parameter is very important in the fitness function. Its mission is to penalize the amplitude of the intervals that conform the itemset. In this way, between two individuals (itemsets) that cover the same number of records and have the same number of attributes, the best information is given by the one whose intervals are smaller, as we can see in figure 6. By means of the factor ψ it is achieved that the algorithm be more or least

permissive with regard to the growth of the intervals. Within this concept, we penalize both the mean and the maximum amplitude of the intervals.

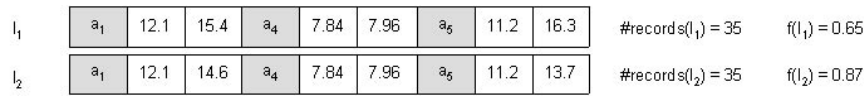

Fig. 6. Amplitude effect

– **number of attributes (*nAtr*)**. This parameter rewards the frequent itemsets with a larger number of attributes. We will be able of increasing or decreasing its effect by means of the factor μ.

All the parameters of the fitness function are normalized into the unit interval. In this way all of them have the same weight when obtaining the fitness of each individual.

5 Experimental Results

To test if the developed algorithm finds in a correct way the frequent itemsets, we have generated several synthetic databases. We have used different functions to distribute the values in the records of the database, in such a way that they group on predetermined sets. The goal will be to find, in an accurate way, the intervals of each one of the sets artificially created. Besides, we have tested our tool with numeric databases from the Bilkent University Function Approximation Repository [6].

To carry out the tests, the algorithm was executed with a population of 100 individuals and 200 generations. We have chosen the following parameters in the GAR algorithm: 15% of selected individuals for the selection operator, 50% of crossover probability and 80% of mutation probability.

5.1 Synthetic Databases

A first database formed by four numeric attributes and 1000 records was generated. The values were distributed, by means of a uniform distribution, into 5 sets formed by predetermined intervals. Besides, 500 new records were added with the idea of introducing noise in the data, distributing their values, by means of a uniform distribution, between the minimum and maximum values of the domain of the intervals. In table 1 the 5 sets synthetically created are shown and in table 2 we show the frequent itemsets found by GAR.

The exact support for each of the synthetically defined sets is 13.34%, since each of them cover 200 records. As can be seen in table 2, the support of each of the sets found is quite close to such value, with a suitable size for each interval. The results show that the algorithm behaves in a correct way when the database

Table 1. Sets synthetically created by means of an uniform distribution

sets
$A_1 \in [1, 15]$, $A_2 \in [7, 35]$, $A_3 \in [60, 75]$, $A_4 \in [0, 25]$
$A_1 \in [5, 30]$, $A_2 \in [25, 40]$, $A_3 \in [10, 30]$, $A_4 \in [25, 50]$
$A_1 \in [45, 60]$, $A_2 \in [55, 85]$, $A_3 \in [20, 25]$, $A_4 \in [50, 75]$
$A_1 \in [75, 77]$, $A_2 \in [0, 40]$, $A_3 \in [58, 60]$, $A_4 \in [75, 100]$
$A_1 \in [10, 30]$, $A_2 \in [0, 30]$, $A_3 \in [65, 70]$, $A_4 \in [100, 125]$

Table 2. Frequent itemsets found by GAR

frequent itemsets	sup(%)	#records
$[1, 15]$, $[6, 35]$, $[60, 76]$, $[0, 26]$	13.40	201
$[5, 30]$, $[24, 40]$, $[10, 30]$, $[26, 51]$	13.07	196
$[44, 61]$, $[55, 84]$, $[20, 35]$, $[50, 75]$	13.34	200
$[74, 77]$, $[0, 40]$, $[58, 60]$, $[75, 101]$	13.34	200
$[9, 29]$, $[0, 30]$, $[62, 71]$, $[102, 125]$	12.80	192

contains a set of records that can not be grouped in any frequent itemsets. The values used in the fitness function were: ω=0.7, ψ=0.6 and μ=0.7.

The first experiment was carried out creating sets independent among them, that is, without overlapping. In order to test if the tool works properly when the sets have records in common, a second database was created in the same way that the first one but with overlapping among the sets. In this case 600 records with the values distributed into 3 sets were generated and other 200 records were added to generate noise. In table 3 the three sets synthetically created are shown and in table 4 we show the frequent itemsets found by GAR.

Table 3. Sets synthetically created with overlapping

sets
$A_1 \in [18, 33]$, $A_2 \in [40, 57]$, $A_3 \in [35, 47]$
$A_1 \in [1, 15]$, $A_2 \in [7, 30]$, $A_3 \in [0, 20]$
$A_1 \in [10, 25]$, $A_2 \in [20, 40]$, $A_3 \in [15, 35]$

The penalization factor was decreased to carry out this test in order to permit overlapping among the itemsets. The values used in the fitness function were: ω= 0.4, ψ = 0.6 and μ = 0.7. In both examples we can see that the sizes of the intervals have been reduced to discover the smallest intervals that cover the larger number of records.

The next test was carried out to test the behaviour of the tool when the itemsets are of a variable size. For this test we used the first database but distributing the values only among some of the attributes. In table 5 the five sets synthetically created are shown and in table 6 we show the frequent itemsets found by GAR.

Table 4. Frequent itemsets found by GAR

frequent itemsets	sup(%)	#records
$[16, 32]$, $[41, 57]$, $[35, 46]$	22.12	177
$[1, 16]$, $[7, 30]$, $[1, 22]$	27.38	219
$[11, 25]$, $[19, 41]$, $[13, 35]$	23.88	191
$[1, 24]$, $[7, 37]$, $[0, 34]$	49.50	396

Table 5. Sets variable size

sets
$A_1 \in [1, 15]$, $A_2 \in [7, 35]$, $A_4 \in [0, 25]$
$A_2 \in [25, 40]$, $A_3 \in [10, 30]$, $A_4 \in [25, 50]$
$A_2 \in [55, 85]$, $A_4 \in [50, 75]$
$A_1 \in [75, 77]$, $A_2 \in [0, 40]$, $A_3 \in [58, 60]$, $A_4 \in [75, 100]$
$A_1 \in [10, 30]$, $A_3 \in [65, 70]$

The result of the test shows how the tool found the predefined frequent itemsets. Besides, two new sets appeared as a consequence of the random distribution of the rest of the values. In this test the penalization factor and the number of attributes were loosen to find itemsets of variable size. The values used in the fitness function were: $\omega = 0.5$, $\psi = 0.6$ and $\mu = 0.45$.

5.2 Real-Life Databases

With the idea of evaluating our tool with real databases, we carried out some experiments using the Bilkent University Function Approximation Repository.

Due to the fact that the performance of the tool is based in a EA, we have carried out five times the proofs in the examples and the results fit in with the average values of such proofs. In 7 the results obtained are shown. The first and second column indicate the number of records and the number of numeric attributes of each database respectively. The third column (*#itemsets*) indicates the mean number of frequent itemsets found. The value of the column *support* indicates the mean of support of the found itemsets, while *size* shows the mean number of attributes of the itemsets. The column *%amplitude* indicates the mean size of the intervals that conform the set. This measure is significant to test that the intervals of the sets are not too many ample. The last column (*%records*) shows the percentage of records covered by the found itemsets on the total records.

Due to the fact of not knowing a priori the distribution of the values of the records, we use a minimum support of 20% and thresholds of $\omega = 0.4$, $\psi = 0.7$ and $\mu = 0.5$ to carry out this tests. The tool found frequent itemsets with high values of support but without expanding the intervals in excess (amplitude percentage below 30%).

Table 6. Frequent itemsets found by GAR

frequent itemsets	sup(%)	#records
$[1, 15]$, $[8, 34]$, $[0, 24]$	10.94	164
$[25, 38]$, $[12, 30]$, $[24, 46]$	10.20	153
$[55, 77]$, $[50, 73]$	11.60	174
$[75, 78]$, $[1, 37]$, $[58, 61]$, $[75, 100]$	12.40	186
$[10, 30]$, $[64, 70]$	14.07	211
$A_2 \in [0, 40]$, $A_3 \in [13, 70]$	42.74	641
$A_1 \in [0, 31]$, $A_3 \in [9, 73]$	33.47	502

Table 7. Results for real-life databases

Database	records	#att	#itemsets	support	size	%ampl	%records
baskball (BK)	96	5	5.6	36.69	3.38	25	100
bodyfat (FA)	252	18	4.2	65.26	7.45	29	86
bolts (BL)	40	8	5.6	25.97	5.29	34	77.5
pollution (PO)	60	16	4.8	46.55	7.32	15	95
quake (QU)	2178	4	6.9	38.65	2.33	25	87.5
sleep (SL)	62	8	5.2	35.91	4.21	5	79.03
stock price (SP)	950	10	6.8	45.25	5.8	26	99.26
vineyard (VY)	52	4	6.6	36.08	3	17	100

6 Conclusions

We have presented in this paper a tool to discover association rules in numeric databases without the necessity of discretizing a priori, the domain of the attributes. In this way the problem of finding rules only with the intervals created before starting the process is avoided. We have used an evolutionary algorithm to find the most suitable amplitude of the intervals that conform a k-itemset, so that they have a high support value without being the intervals too wide. We have carried out several test to check the tools behaviour in different data distributions, obtaining satisfactory results if the frequent itemsets have no overlapping, if they have overlapping and if they are of a variable size. Nowadays, we are studying new measures to include in the fitness function and to find, with more accuracy, the size of the intervals in a k-itemset.

7 Acknowledgments

This work has been supported by Spanish Research Agency CICYT under grant TIC2001-1143-C03-02

References

[1] Agrawal, R., Imielinski. T., Swami, A.: Mining association rules between sets of items in large databases. Proc. ACM SIGMOD. (1993) 207–216, Washington, D.C.

[2] Agrawal, R., Srikant, R: Fast Algorithms for Mining Association Rules. Proc. of the VLDB Conference (1994) 487–489, Santiago (Chile)
[3] Aumann, Y., Lindell, Y.: A Statistical Theory for Quantitative Association Rules. Proceedings KDD99 (1999) 261–270, San Diego, CA
[4] Goldberg, D.E: Genetic algorithms in search, optimization and machine learning. Addison-Wesley. (1989)
[5] González, A., Herrera, F.: Multi-stage Genetic Fuzzy System Based on the Iterative Rule Learning Approach. Mathware & Soft Computing, 4 (1997)
[6] Guvenir, H. A., Uysal, I.: Bilkent University Function Approximation Repository, http://funapp.cs.bilkent.edu.tr (2000)
[7] Han, J., Pei, J., Yin, Y.: Mining Frequent Patterns without Candidate Generation. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (2000)
[8] Lin, D-I., Kedem, Z.M.: Pincer Search: A New Algorithm for Discovering the Maximum Frequent Set. In Proc. of the 6th Int'l Conference on Extending Database Technology (EDBT) (1998) 105–119 Valencia
[9] Manila, H., Toivonen, H., Verkamo, A.I.: Efficient algorithms for discovering association rules. KDD-94: AAAI Workshop on Knowledge Discovery in Databases (1994) 181–192 Seatle, Washington
[10] Mata, J., Alvarez, J.L., Riquelme, J.C.: Mining Numeric Association Rules with Genetic Algorithms. 5th Internacional Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA (2001) 264–267 Praga
[11] Miller, R. J., Yang, Y.: Association Rules over Interval Data. Proceedings of the International ACM SIGMOD Conference (1997) Tucson, Arizona
[12] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules
[13] Park, J. S., Chen, M. S., Yu. P.S.: An Effective Hash Based Algorithm for Mining Association Rules. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (1995) San José, CA
[14] Savarese, A., Omiecinski, E., Navathe, S.: An efficient algorithm for mining association rules in large databases. Proc. of the VLDB Conference, Zurich, Switzerland (1995)
[15] Srikant, R, Agrawal, R.: Mining Quantitative Association Rules in Large Relational Tables. Proc. of the ACM SIGMOD (1996) 1–12
[16] Wang, K., Tay. S.H., Liu, B.: Interestingness-Based Interval Merger for Numeric Association Rules. Proc. 4th Int. Conf. KDD (1998) 121–128