Mining Frequent Itemsets from Large Data Sets using Genetic Algorithms

R. Vijaya Prakash
Dept. of Informatics
Kakatiya University,
Warangal, India

Dr. Govardhan
Dept. of Computer Science & Engineering
JNTU, Jagityal, India

Dr. S.S.V.N. Sarma
Dept. of Computer Science & Engineering
Vaagdevi Engineering College, Warangal, India

ABSTRACT
Association Rules are the most important tool to discover the relationships among the attributes in a database. The existing Association Rule mining algorithms are applied on binary attributes or discrete attributes, in case of discrete attributes there is a loss of information and these algorithms take too much computer time to compute all the frequent itemsets. By using Genetic Algorithm (GA) we can improve the generation of Frequent Itemset for numeric attributes. The major advantage of using GA in the discovery of frequent itemsets is that they perform global search and its time complexity is less compared to other algorithms as the genetic algorithm is based on the greedy approach. The main aim of this paper is to find all the frequent itemsets from given data sets using genetic algorithm.

General Terms
Genetic Algorithm (GA), Association Rule, Frequent itemset, Support, Confidence, Data Mining.

Keywords
Genetic Algorithm (GA), Association Rule Mining(ARM), Frequent itemset, Data Mining(DM).

1. INTRODUCTION
Large amounts of data have been collected routinely in the course of day-to-day management in business, administration, banking, the delivery of social and health services, environmental protection, security and in politics. Such data is primarily used for accounting and for management of the customer base. Typically, management data sets are very large and constantly growing and contain a large number of complex features. While these data sets reflect properties of the managed subjects and relations, and are thus potentially of some use to their owner, they often have relatively low information density. One requires robust, simple and computationally efficient tools to extract information from such data sets. The development and understanding of such tools is the core business of data mining. These tools are based on ideas from computer science, mathematics and statistics. Mining useful information and helpful knowledge from these large databases has thus evolved into an important research area [1][2].

Data mining has attracted a great deal of attention in the information industry and in society as a whole in recent years, due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge. The information and knowledge gained can be used for applications ranging from market analysis, fraud detection, and customer retention, to production control and science exploration.

Frequent pattern mining is an important area of Data mining research. The frequent patterns are patterns (such as itemsets, subsequences, or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread that appear frequently together in a transaction data set is a *frequent itemset*. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a *frequent sequential pattern*. A *substructure* can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences. If a substructure occurs frequently, it is called a *frequent structured pattern*. Finding such frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks as well.

The process of discovering interesting and unexpected rules from large data sets is known as association rule mining. This refers to a very general model that allows relationships to be found between items of a database. An association rule is an *implication* or *if-then-rule* which is supported by data. The association rules problem was first formulated in [3][4] and was called the *market-basket* problem. The initial problem was the following: given a set of items and a large collection of sales records, which consist in a transaction date and the items bought in the transaction, the task is to find relationships between the items contained in the different transactions. A typical association rule resulting from such a study could be "90 percent of all customers who buy bread and butter also buy milk" – which reveals a very important information. Therefore this analysis can provide new insights into customer behaviour and can lead to higher profits through better customer relations, customer retention and better product placements.

Mining of association rules is a field of data mining that has received a lot of attention in recent years. The main association rule mining algorithm, Apriori, not only influenced the association rule mining community, but it affected other data mining fields as well. Apriori and all its variants like Partition, Pincer-Search, Incremental, Border algorithm etc. take too much computer time to compute all the frequent itemsets. The papers [10][11][12][13] contributed a lot in the field of Association Rule Mining (ARM). In this paper, an attempt has been made to compute frequent itemsets by applying genetic algorithm so that the computational complexity can be improved.

2. ASSOCIATION RULE MINING (ARM)
Association Rule Mining aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items in the transaction databases or other data repositories

[8][14]15][16]. The major aim of ARM is to find the set of all subsets of items or attributes that frequently occur in many database records or transactions, and additionally, to extract rules on how a subset of items influences the presence of another subset. ARM algorithms discover high-level prediction rules in the form: IF the conditions of the values of the predicting attributes are true, THEN predict values for some goal attributes.

In general, the association rule is an expression of the form X=>Y, where X is antecedent and Y is consequent. Association rule shows how many times Y has occurred if X has already occurred depending on the support and confidence value.

Support: It is the probability of item or itemsets in the given transactional data base: support(X) = n(X) / n where n is the total number of transactions in the database and n(X) is the number of transactions that contains the item set X.
Therefore, support (X→Y) = support(XUY).

Frequent itemset: Let A be a set of items, T be the transaction database and *minsup* be the user specified minimum support. An itemset X in A (i.e., X is a subset of A) is said to be a *frequent itemset* in T with respect to *minsup* if $\text{support}(X)_T > minsup$

The problem of mining association rules can be decomposed into two sub-problems:

- Find all itemsset whose support is greater than the user-specified minimum support, *minsup*. Such itemsets are called *frequent itemsets*.
- Use the frequent itemsets to generate the desired rules. The general idea is that if, say ABCD and AB are frequent itemsets, then we can determine if the rule AB=>CD holds by checking the following inequality

  support({A,B,C,D}) / support({A,B}) > *minconf*, where the rule holds with confidence *minconf.*

To demonstrate the use of the support-confidence framework, we illustrate the process of mining association rules by the following example.

Example 1. Assume that we have a transaction database in a supermarket, as shown in Table 1. There are six transactions in the database with their transaction identifiers (TIDs) ranging from 100 to 600. The universal itemset I ={A, B, C, D, E}, where A, B, C, D and E can be any items in the supermarket. For instance, A = ''bread", B = ''milk", C = ''sugar", D = ''coffee", and E = ''biscuit".

Table 1. An example transaction database

TID	Items bought
100	ABCD
200	BCE
300	ABCE
400	BE
500	ACD
600	BCE

There are totally 2^5(=32) itemsets. {A}, {B}, {C}, {D}, and {E} are all 1-itemsets, {AC} is a 2-itemset, and so on. Supp(BC) = 4/6 = 0.67 because there are four transactions that contain both A and B. Let *minsupp* = 50% and *minconf* = 80%. Then, A, B, C, E, AC, BC, BE and BCE are all frequent itemsets. The confidence of association rule A=>C is conf(A=>C) = supp(AC) / supp(A) = 3/3 = 1.0. Hence rule A => C is valid. Similarly we have conf (C =>A) = 3/5 = 0.6. Hence, rule C => A is not valid.

3. GENETIC ALGORITHM

Genetic Algorithms (GAs) are adaptive heuristic search algorithm premised on the evolutionary ideas of natural selection and genetic. The basic concept of GAs is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such they represent an intelligent exploitation of a random search within a defined search space to solve a problem.

GAs are one of the best ways to solve a problem for which little is known. They are a very general algorithm and so will work well in any search space. The Genetic Algorithm [5] was developed by John Holland in 1970. GA is stochastic search algorithm modeled on the process of natural selection, which underlines biological evolution [6].

GA has been successfully applied in many research, optimization and machine learning problems. GA works in an iteretative manner by generating new populations of strings from old ones. Every string is the encoded binary, real etc. version of a candidate solution. An evaluation function associates a fitness measure to every string indicating its fitness for the problem [7].

Standard GA apply genetic operators such *selection, crossover a*nd *mutation* on an initially random population in order to compute a whole generation of new strings. GA runs to generate solutions for successive generations. The probability of an individual reproducing is proportional to the goodness of the solution it represents. Hence the quality of the solutions in successive generations improves. The process is terminated when an acceptable or optimum solution is found. GA is appropriate for problems which require optimization, with respect to some computable criterion. The functions of genetic operators are as follows:

1) *Selection:* Selection deals with the probabilistic survival of the fittest, in that, more fit chromosomes are chosen to survive. Where fitness is a comparable measure of how well a chromosome solves the problem at hand.

2) *Crossover:* This operation is performed by selecting a random gene along the length of the chromosomes and swapping all the genes after that point.

3) *Mutation:* Alters the new solutions so as to add stochasticity in the search for better solutions. This is the chance that a bit within a chromosome will be flipped (0 becomes 1, 1 becomes 0).

Genetic algorithms are a method of "breeding" computer programs and solutions to optimization or search problems by means of simulated evolution. Processes loosely based on natural selection, crossover, and mutation are repeatedly applied to a population of binary strings which represent potential solutions. Over time, the number of above-average individuals increases and highly-fit building blocks are combined from several fit individuals to find good solutions to the problem at hand.

Not only does GAs provide alternative methods to solving problem, it consistently outperforms other traditional methods in most of the problems link. Many of the real world problems involved finding optimal parameters, which might prove difficult for traditional methods but ideal for GAs.

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

4 PRACTICAL IMPLEMENTATION

4.1 GAR Algorithm

The GAR (Genetic Association Rules) algorithm is based in the theory of evolutionary Algorithms, 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.

```
algorithm GAR
Begin
  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.1. GAR Algorithm

In Figure 1 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.

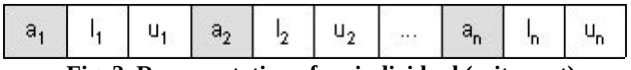

Fig. 2. Representation of an individual (n-itemset)

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 2, 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, and 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:

$O1 = ([[l_1, u_1] \vee [l'_1, u'_1]], [[l_3, u_3] \vee [l'_3, u'_3]])$
$O2 = ([[l'_1, u'_1] \vee [l_1, u_1]], [[l'_2, u'_2], [[l_3, u_3] \vee [l_3, u_3]])$

In Figure 3 a possible result of the crossover operator for two itemsets of different size can be seen.

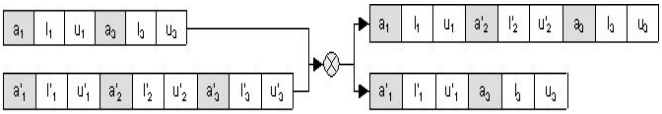

Fig.3. Example of a crossover operation

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 the best candidates are in the following generations.

In our scenery, we look for the frequent item sets 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:

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

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 an 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 4. 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.

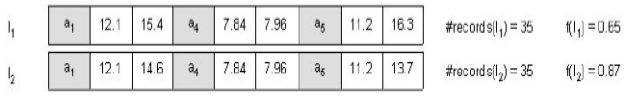

Fig.4. Amplitude effect

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.

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 2 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 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.

Table 2. Sets synthetically created by means of a uniform distribution

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

Table 3. 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

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 4 the three sets synthetically created are shown and in table 5 we show the frequent itemsets found by GAR.

Table 4. Sets synthetically created with overlapping

sets
$A1 \varepsilon$ [18, 33], $A2 \varepsilon$ [40, 57], $A3 \varepsilon$ [35, 47]
$A1 \varepsilon$ [1, 15], $A2 \varepsilon$ [7, 30], $A3 \varepsilon$ [0, 20]
$A1 \varepsilon$ [10, 25], $A2 \varepsilon$ [20, 40], $A3 \varepsilon$ [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 6 the five sets synthetically created are shown and in table 7 we show the frequent itemsets found by GAR.

Table 5. 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 6. Sets variable size

sets
$A1 \varepsilon$ [1, 15], $A2 \varepsilon$ [7, 35], $A4 \varepsilon$ [0, 25]
$A2 \varepsilon$ [25, 40], $A3 \varepsilon$ [10, 30], $A4 \varepsilon$ [25, 50]
$A2 \varepsilon$ [55, 85], $A4 \varepsilon$ [50, 75]
$A1 \varepsilon$ [75, 77], $A2 \varepsilon$ [0, 40], $A3 \varepsilon$ [58, 60], $A4 \varepsilon$ [75, 100]
$A1 \varepsilon$ [10, 30], $A3 \varepsilon$ [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 an 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 8 the results obtained are shown. The first and second column indicates 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 7. 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
$A2 \varepsilon$ [0, 40], $A3 \varepsilon$ [13, 70]	42.74	641
$A1 \varepsilon$ [0, 31], $A3 \varepsilon$ [9, 73]	33.47	502

Table 8. Results for real-life databases by GAR

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

Table 9. Results for real-life databases

Database	records	#att	#itemsets	support	size	%ampl	#records
baskball (BK)	96	5	7.6	36.69	5.38	20	90
bodyfat (FA)	252	18	6.2	65.26	8.45	38	76
bolts (BL)	40	8	8.6	25.97	9.29	44	67.5
pollution (PO)	60	16	6.8	46.55	9.32	25	90
quake (QU)	2178	4	8.9	38.65	3.33	30	80.5
sleep (SL)	62	8	7.2	35.91	6.21	15	89.03
stock price (SP)	950	10	8.8	45.25	6.80	46	90.26
vineyard (VY)	52	4	5.6	36.08	3.50	17	100

The Table 9 results are obtained by applying the approri algorithms on different data sets to get frequent itemsets. By comparing the Table 8 and 9, we can say that the Genetic Algorithms approach for finding the Frequent Itemset is more efficient.

6 CONCLUSIONS

We have presented in this paper a tool to discover association rules in 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 behavior 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. 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] Chen M.S., Han J. and Yu P.S (1996) Data Mining : An Overview from a Database Prospective, IEEE Trans.Knowledge and Data Eng., 866-883

[3] Agarwal R, Imielinski. T., Swami, (1993) Database Mining: a performance prospective, IEEE Transaction on Knowledge and Data Engineering 5(6), 914-925.

[4] Agrawal, R., Srikant, R: Fast Algorithms for Mining Association Rules. Proc. Of the VLDB Conference (1994) 487–489, Santiago (Chile)

[5] Pei M., Goodman E.D.,Punch F. (2000) Feature extraction using Genetic Algorithm, Case Center for Computer Aided Engineering and Manufacturing W. Department of Computer Science

[6] Stuart J. Russel, Peter Novig (2008) Artificial Intellegence: A Modern Approach

[7] Goldberg, D.E: Genetic algorithms in search, optimization and machine learning. Addison-Wesley. (1989)

[8] Han J., Kamber M. Data Mining Concepts & Techniques, Morgan & Kaufmann, 2000.

[9] Pujari A.K., Data Mining Technique, Universities Press, 2001

[10] Anandhavalli M, Suraj Kumar Sudhanshu, Ayush Kumar and Ghose M.K. (2009) Optimized Association Rule Mining using Genetic Algorithm, Advances in Information Mining, ISSN:0975-3265, Volume 1, Issue 2, 2009, pp-01-04.

[11] Markus Hegland. The Apriori Algorithm – a Tutorial, CMA, Australian National University, WSPC/Lecture Notes Series, 22-27, March 30, 2005.

[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 Jos´e, 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