A Parameterised Algorithm for Mining Association Rules

Nuansri Denwattana and Janusz R Getta
School of Information Technology and Computer Science
University of Wollongong
Australia
{nd22,jrg@cs.uow.edu.au}

Abstract

A central part of many algorithms for mining association rules in large data sets is a procedure that finds so called frequent itemsets. This paper proposes a new approach to finding frequent itemsets. The approach reduces a number of passes through an input data set and generalises a number of strategies proposed so far. The idea is to analyse a variable number n of itemset lattice levels in p scans through an input data set. It is shown that for certain values of parameters (n,p) this method provides more flexible utilisation of fast access transient memory and faster elimination of itemsets with low support factor. The paper presents the results of experiments conducted to find how performance of association rule mining algorithm depends on the values of parameters (n,p).

Keywords: Data Mining, Association Rules, Frequent Itemsets, Algorithms

1 Introduction

The algorithms for mining association rules in large data sets attracted a lot of attention in the recent years. The original problem [1] was to find the correlations among the sales of different products from the analysis of large set of supermarket data. Association rule is an implication that determines co-occurrence of the objects in a large set of so called transactions, e.g. customer baskets, collections of measurements, etc. At present, the research works on association rules are motivated by an extensive range of application areas such as banking, manufacturing, health care, and telecommunications. Association rule discovery techniques are used to detect suspicious credit card transactions, money-laundering activities [9] in banking and financial businesses. The same techniques are applied in manufacturing, controlling, and scheduling of technical production processes [5]. The other application areas include health care [7] and management of telecommunication networks [6].

The discovery of association rules is typically done in two steps [1]. Analysis of experimental data performed in the first step provides a minimal set of objects *(itemsets)* such that frequency of their co-occurrence is above a given threshold *(minimum support)*. These itemsets are called as *frequent itemsets*. The second step uses the frequent itemsets to construct the association rules. It has been shown that computational complexity of the problem is buried in the searching for a minimal set of frequent itemsets in the first step. Generation of association rules from frequent itemsets has a linear complexity and it has no impact on the overall performance.

A number of algorithms finding frequent itemsets in large data sets have been already proposed. Majority of them counts one category of itemsets, e.g. all k element itemsets in one pass through an input data set. For instance, Apriori algorithm [2] counts n element itemsets in the n-th pass through a data set. All frequent itemsets identified in the n-th pass are used to generate the hypothetically frequent itemsets *(candidate itemsets)* for verification in the next pass. Frequent itemsets obtained from the n-th pass and being the subsets of frequent itemsets identified in the next pass are pruned. The process continues until no new frequent itemsets are found. Sampling for frequent itemsets algorithm [10] extracts a random sample from a data set and finds all frequent itemsets there. Next, it tries to verify the results on a complete data set. A *top-down* approach [11] applies the maximum clique generation algorithm to find a *ceiling* of the minimal set of frequent itemsets. Next, the subsets of all frequent itemsets included in a ceiling are counted in each pass through a data set. DIC algorithm [3] stops counting itemsets as soon as there is no chance for an itemset to be frequent. Each elim-

1530-0919/01 $10.00 © 2001 IEEE

inated itemset is immediately replaced with another itemset. A new technique recently proposed in [4] uses FP-tree to store compressed crucial information about frequent itemsets. This technique needs a huge volume of transient memory if a number of frequent itemsets is too large. Partition algorithm [8] transforms an input data set from a horizontal layout to a vertical layout and uses a list intersection technique to identify the frequent itemsets.

An approach presented in this paper considers a hypothetical *perfect algorithm* capable of guessing and verifying all frequent itemsets in one scan through an input data set. An input to the perfect algorithm is a set of frequent and non-frequent itemsets called as a *perfect guess*. A perfect guess includes both frequent and non-frequent itemsets because for each frequent itemset found we have to show that none of its supersets is frequent. For example, if a set of all items is $\{A, B, C\}$ and $\{A\}$, $\{B\}$, $\{C\}$, $\{A, B\}$ are frequent itemsets then to verify that $\{\{A, B\}, \{C\}\}$ is the minimal set of frequent itemsets we have to check in a data set that $\{A, B\}$, $\{C\}$ are frequent and that $\{A, C\}$, $\{B, C\}$ are not frequent. As the result a perfect guess consists of the candidate itemsets from many levels of itemset lattice. The quality of association rule mining algorithms is determined by two factors. The first one is a number of passes through an input data set. The other one is a number of comparisons of candidate itemsets with input transactions in order to find which candidate itemsets should be counted. The perfect algorithm minimises both parameters. It needs to read an input data set only once and it needs to perform the smallest number of comparisons to verify a perfect guess. For instance, elimination of any candidate itemset in order to reduce a number of comparisons results with a different solution.

We are aware that implementation of the *perfect algorithm* is unrealistic because probability of making a correct guess in a large data set is very low. Our idea is to treat a concept of perfect algorithm in a way similar to how a concept of "absolute zero temperature" is treated in physics. It is going to be the ultimate goal, i.e. a point which cannot be achieved and in the same moment a point that can be used to measure the quality of the realistic algorithms.

One of the objectives is to construct an algorithm that makes a *good guess*, i.e. a guess that is not perfect and in the same moment it does not contain too many errors. To make a good guess we need to get some information about the properties of an input data set. It leads to a strategy where a data set is read once, the statistics are collected and used to guess all $2, 3, \ldots, n-th$ element candidate itemsets. Next, a guessed set of candidate itemsets is minimised and extended by a minimal set of non-frequent itemsets that have to be tested to prove its correctness. At the end an input data set is scanned for the second time to verify a guess. Due to a fact that initial guess is not perfect some of the items that suppose to be frequent appear not to be frequent and vice versa. A set of mistakes detected during verification is used to generate a new set of candidate itemsets that should be verified again. The third scan through a data sets eliminates all mistakes and provides the final solution for a range of $2, 3, \ldots, n-th$ element itemsets. Then, the same procedure is repeated for the next range of itemsets. To implement such an algorithm we need a procedure capable of guessing frequent itemsets from the statistics collected in the first scan of input data set. To our best knowledge none of the algorithms proposed so far has such properties.

A problem with the approach sketched above is that we make more errors in guessing of itemsets that contain more items. This is because the errors done at the lower levels of itemset lattice multiple themselves very fast at the higher levels. A number of error has an important impact on performance because each of them requires the additional comparisons of candidate itemsets with transactions from an input data set. These observations lead to a parameterised version of the algorithm. In order to decrease a number of errors at the higher levels, we parameterise a range of itemsets for which a guess is done. On the other hand, smaller guessing range increases a number of passes through an input data set. The parameterised (n, p) algorithm finds all frequent itemsets from a range of n levels in itemset lattice in p passes $(n >= p)$ through an input data set. A classical Apriori algorithm is a special case of (n, p) algorithm where $n = p = 1$, i.e. the candidate itemsets from one level of itemset lattice are verified in one pass. An interesting question is what combinations of n and p values provide the best performance. Intuitions are such that as a ratio n/p increases we have to perform more unnecessary comparisons of candidate itemsets with transactions from an input data set. On the other hand, if ratio n/p decreases then we perform less unnecessary comparisons and in the same moment we read an input data set more frequently.

The rest of this paper is organised as follows. A detail of (n, p) algorithm, including guessing, verifying procedures, and an example, is given in Section 2. Experiments of (n, p) algorithm is demonstrated in Section 3. A summary and a discussion of future research are provided in Section 4.

2 Finding frequent itemsets

This section presents a parameterised (n, p) algorithm for mining frequent itemsets. It also contains the description of guessing and verification of candidate itemsets.

2.1 Problem description

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called *items*. Let D be a set of transactions, where each transaction $t \in D$ consists of transaction identifier *tid* and set of items $I_t \subseteq I$. We assume that the items are kept ordered within each transaction. We call an itemset that contains k items as k-itemset.

Association rule is an expression $X \Rightarrow Y$ where X, Y are itemsets and $X, Y \subseteq I$ and $X \cap Y = \emptyset$. The *support* for an itemset is defined as a fraction of all transactions that includes $X \cup Y$. The *confidence* of a rule $X \Rightarrow Y$ is defined as $(X \cup Y)/X$. We accept a rule $X \Rightarrow Y$ as true if its confidence exceeds a given threshold value.

A *candidate itemset* is an itemset selected for verification of its support in a data set. An itemset is *a positive candidate itemset* when it is assumed (guessed) to be frequent. Otherwise, it is called as *a negative candidate itemset.* Both positive and negative candidate itemsets are verified in single pass through a data set. A candidate itemset becomes *a frequent itemset* when verification shows its support level above a given threshold value. *A remaining candidate itemsets* is candidates verified in another scan.

In the rest of the paper candidate k-itemsets are denoted by C_k, positive (negative) candidate k-itemsets are denoted C_k^+ (C_k^-), remaining candidates are denoted by C_k^r, and frequent k-itemsets are denoted by L_k.

2.2 The algorithm

The algorithm starts from an initial pass through an input data set in order to find all frequent 1-itemsets (L_1) and to collect the statistics of the total number of 1-, 2-, ..., n-element transactions that contain the elements from L_1. The statistics are stored in table T, e.g. see Table 1. Then, the initial value of current level k is set to 2 and initial result is set to L_1. If L_{k-1} is not empty, a procedure **guess_candidates** is called to guess the candidate itemsets from the next n levels. The procedure returns a set **C** of positive and negative candidate itemsets. The elements of **C** are verified in an input data set by a procedure **verify_candidates**. The procedure finds all errors done by **guess_candidates** in one pass through an input data set. Then, it corrects the errors and finds the solution for levels from k to $k+n-1$ in the second pass through the data set. A minimal set of frequent frequent itemsets found is added to the result set. The value of k is then increased by n. These steps are repeated until $L_{(k-1)}$ is empty. A pseudo-code of the algorithm is given below

```
n := number of lattice levels traversed at a time;
sup := minimum support;
Results := $\emptyset$;
generate $L_1$;
generate statistics table $T$;
Result := Result $\cup L_1$;
k := 2;
while $L_{k-1} \neq \emptyset$ do
    guess_candidates($T$, $L_{k-1}$, $t_f$, $t_t$, n, k, C);
    verify_candidates(C, sup, n, k);
    Result := Result $\cup$ $\{L_k, L_{k+1}, \ldots, L_{k+n-1}\}$;
    k := k + n;
end;
```

2.3 Guessing candidate itemsets

The procedure **guess_candidates** finds all candidate itemsets from a range of levels from k to $k+n-1$ that accordingly to our guessing method would verify as frequent itemsets. The procedure takes on input statistics table T, frequency thresholds (t_f), m-element transaction threshold (t_t), set L_{k-1} of frequent itemsets, level k it starts from, and number n of levels to be considered.

Guessing starts at level k. The procedure uses **apriori_gen** function proposed in [2] to generate a set C_k of candidate k-itemsets from L_{k-1}. Then, it uses the statistics from table T to decide which candidate itemsets in C_k are positive (C_k^+) and which one are negative (C_k^-). A frequency threshold is applied to all transactions that consists of k or more elements. The output of this step is a set of single items whose frequencies satisfy the frequency threshold. If any itemsets in C_k are subset of the output set, then we put them into a set of positive candidate k-itemsets. We repeat this step until it reaches transaction length m. Finally, if there are any k-itemsets in C_k which are not in a set of positive candidate k-itemsets then they are appended to a set of negative candidate k-itemsets.

In the next step **apriori_gen** is applied to C_k^+ to form set of $C_{(k+1)}$. This time we consider from (k+1)-element to m-element transactions. The sets of $C_{(k+1)}^+$ and $C_{(k+1)}^-$, are then generated. Next, $C_{(k+1)}^+$ is used to form $C_{(k+2)}$. This procedure is repeated until we reach level $(k+n-1)$. Finally, all subsets of itemsets in $C^+_{(k+n-1)}$ at lower levels are pruned.

For example, assume that procedure **guess_candidates** is called with the following parameters: item frequency threshold equals to 80%, m-element transaction threshold equals to five (5-element transaction), number of levels to traverse equals to three, starting level equals to two, and table statistics table T as follows.

Item	freq. according to tr. length			
	3 els.	4 els.	5 els.	Total freq
A	3	2	2	7
B	4	2	3	9
C	3	2	3	9
D	1	1	1	3
E	3	1	3	7
F	1	0	3	4
no. of m-els trs.	5	2	3	10

Table 1: Sample statistics in table T

Suppose we are at level k, and the *apriori_gen* function, generated $C_k = \{AB, AC, AD, AE, AF, BC, BD, BE, BF, CD, CE, CF, DE, DF, EF\}$ As there is no k-element transactions in table 1, we consider (k+1)-element transaction. 80% of the total number of (k+1)-transactions, i.e. five, is four. The output set of single items whose frequencies satisfy the frequency threshold is $\{B\}$. Consequently, there is no itemsets in C_k which is subset of this set. Next, applying the frequency threshold to transaction length 4, and this time the output set is $\{ABC\}$. As there are some itemsets in C_k are subset of $\{ABC\}$, they are put into a set of positive candidates. We repeat this step in the transaction length 5. Finally, set of C^+_k and C^-_k are as follows:
$C^+_k = \{AB, AC, AE, AF, BC, BE, BF, CE, CF, EF\}$
$C^-_k = \{AD, BD, CD, DE, DF\}$

We use two thresholds to guess the candidate itemsets: item's frequency threshold and m-element transaction threshold. The accuracy of candidate guessing is determined by both of them. If a value of item's frequency threshold is high then accuracy of candidate guessing will be high as well. However, we will get less frequent itemsets from the first scan because we have less positive candidates. Consequently, the extra database passes may be needed to determine large number of remaining candidate itemsets. On the other hand, if the value of item frequency threshold is low, we have too many errors. In addition, the higher value m-element transaction threshold is, more errors of candidate itemsets are generated.

2.4 Verification of candidate itemsets

Verification of candidate itemsets includes verification of candidates provided by **guess_candidates** procedure and elimination of errors done at the guessing stage. The procedure **verify_candidates** takes on input a set **C** with positive and negative candidate itemsets, minimum support (sup), starting level (k), and number of lattice levels traversed (n).

In the first stage, the procedure scans an input data set and finds all positive candidate itemsets which appear to be negative and vice versa. Due to errors in guessing it has to construct a new set of candidate itemsets and verify them once more. If certain C^+_j appears to be not frequent then all its subsets from levels k to $(k+j-1)$ are generated. Then, they are trimmed by supersets which appear to be frequent. Similarly, if certain C^-_j appears to be not frequent then all its supersets from levels $j+1$ to $k+n-1$ are generated and trimmed by the verified frequent itemsets. In the next stage, the confirmation procedure scans an input set for the second time and verifies the final solution. Although the (n,p) algorithm moves n levels at a time, the total number of candidate itemsets is more or less the same as other algorithms moving level by level. It is because itemsets in lower levels will not be subsets of any candidates in higher levels, both in positive and negative candidate itemsets.

2.5 Example

This subsection describes a sample execution of (n,p) algorithm for $n = 3$, $p = 2$, and the statistics given in Table 1. Suppose that frequency threshold $t_f = 80\%$, m-element transaction threshold $t_t = 5$, and the sets of positive and negative candidates at level 2 are:
$C^+_2 = \{$AB AC AE AF BC BE BF CE CF EF$\}$
$C^-_2 = \{$AD BD CD DE DF$\}$

Using only set of C^+_2 and apply the thresholds to Table 1, set of C^+_3 and C^-_3 are as follows:
$C^+_3 = \{$ABC ABE ABF ACE ACF AEF BCE BCF BEF CEF$\}$
$C^-_3 = \{\}$

The procedure is repeated at level 4.
$C^+_4 = \{$ABCE ABCF ABEF ACEF BCEF$\}$
$C^-_4 = \{\}$

Set of final positive and negative candidate itemsets after pruning all subsets of positive superset are:
$C^+_4 = \{$ABCE ABCF ABEF ACEF BCEF$\}$
$C^+_3 = C^+_2 = \{\}$

$C_2^- = \{\text{AD BD CD DE DF}\}$
$C_3^- = C_4^- = \{\}$

The database are scanned to verify these candidate itemsets. With 20% of minimum support, partial frequent 2-, 3-, 4-itemsets are as follows:
$L_2 = \{\text{BD CD DE}\}$
$L_3 = \{\}$
$L_4 = \{\text{ABCE ABCF ABEF ACEF BCEF}\}$

Then set of remaining candidate itemsets are generated,
$C_2^R = \{\}$
$C_3^R = \{\text{BCD BDE CDE}\}$
$C_4^R = \{\text{BCDE BCDF}\}$

Verifying sets of remaining candidate by scanning the database, frequent 2-, 3-, and 4-itemsets are generated.
$L_2 = \{\text{AB AC AE AF BC BD BE BF CD CE CF DE EF}\}$
$L_3 = \{\text{ABC ABE ABF ACE ACF AEF BCD BCE BCF BDE BEF CEF}\}$
$L_4 = \{\text{ABCE ABCF ABEF ACEF BCEF}\}$

By using frequent 4-itemsets, candidate itemsets of another three levels are formed. As there is only one 5-itemset, there is no need to form sets of candidate 6-, 7-itemsets.
$C_5 = \{\text{ABCEF}\}$

Scanning the database, frequent 5-itemsets are, finally, determined.
$L_5 = \{\text{ABCEF}\}$

3 Experimental Results

To assess the performance of (n, p) algorithm, we conducted several experiments on different data sets. The algorithm was implemented in C language and we tested it on Unix platform. The experiments used the synthetic data sets generated by IBM's synthetic data generator from Quest project. We considered the following parameters: number of transactions in a database (ntrans), average transaction length (tl), number of patterns (np), and a minimum support (sup).

We have tried a range of number of transactions, average transaction length, and a number of patterns. As we expected, the results show that for $n \neq 1$ and $p \neq 1$ our approach provides better results than Apriori algorithm ($n = 1$ and $p = 1$) both in terms of execution time and the total number of database scans.

Parameters			no. database scans						
tl	np	sup	Apr	$(n, 2)$					
				3	5	7	8	10	12
10	10	20	9	6	4	4	3	-	-
12	10	20	11	7	4	4	4	3	-
14	10	20	13	7	6	4	4	4	3
20	100	10	8	5	4	3	-	-	-

Table 2: A comparison of no. database scans between Apriori and (n, p) algorithm

Table 2 shows a number of database scans of (n, p) algorithm compared with Apriori in different distributions of data sets.

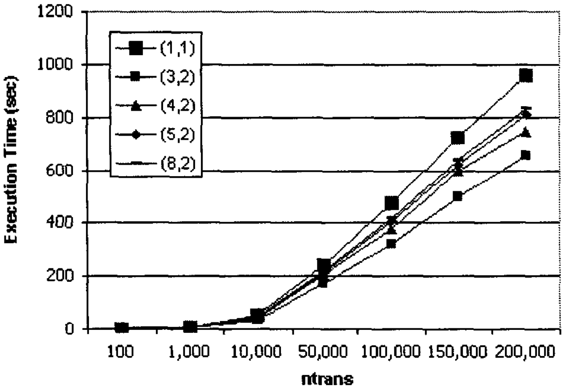

Figure 1. Performance of Apriori and (n,p) with tl=10 np=10 sup=20%

Figure 1 presents the results for different numbers of transactions and fixed number of candidate itemsets are. We compared Apriori with (n, p) algorithm by moving several levels in two passes. With small size of databases, the performance of Apriori and (n, p) algorithm is approximately the same. When an input data set is larger, the performance of (n, p) algorithm is much better than Apriori. It is because a number of database scans of (n, p) algorithm is less than in Apriori. In addition, three to four levels are the optimal movings which are the best performance of this data set. We also conducted the other experiments with different data distributions, as shown in Figures 2 and 3. When the data distribution is more scattered, the execution time of (n, p) algorithm is not much different to Apriori. It is because both algorithms have to determine many candidate itemsets which are not frequent.

Figure 4 shows the performance of (n, p) algorithm with the increasing of the ratio of n/p. We parame-

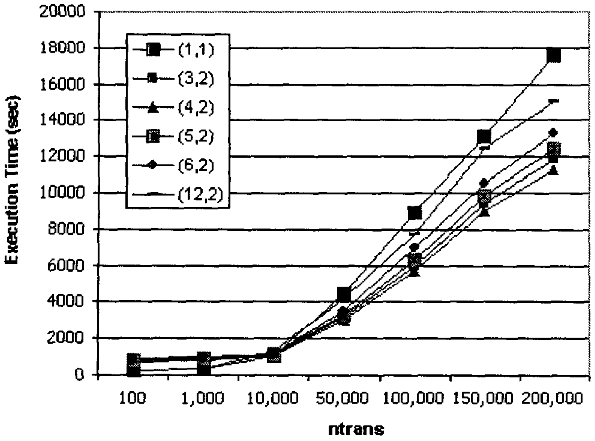

Figure 2. Performance of Apriori and (n,p) algorithm with tl=14 np=10 sup=20%

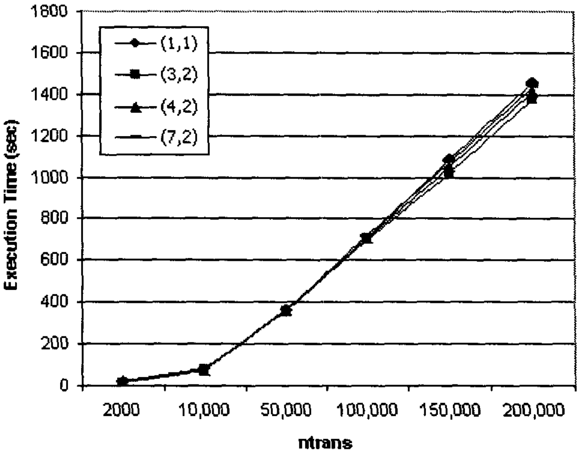

Figure 3. Performance of Apriori and (n,p) algorithm with tl=20 np=100 sup=10%

terised the algorithm by moving one level in one pass of data set and moving more than one levels in three passes. It showed that when a ratio increases, the performance decreases due to the itemset guessing with more elements, which resulted in getting more errors.

Finally, we illustrated performance of (n, p) algorithm by varying number of database passes (p) and fixing number of levels moving a time $(n = 8)$, as shown in Figure 5, to confirm that we should not move too many levels in a few database scans, as well as should not move one level in one database pass.

4 Summary and future works

This work proposes a new approach to finding frequent itemsets in mining association rules. The important contribution of our method is the reduction of number of scans through a data set. The main idea of our new algorithm are to guess candidate itemsets in each level of itemset lattice starting from level k up to $k + n - 1$ and to verify such candidate itemsets. To have a good guess, some statistical data from input data are corrected during the database is scanned. By using such information, the candidate itemsets are generated. Next, these candidate itemsets are verified by scanning the database. If there are some errors from the guessing, another scan through a database will be needed to eliminate such errors and produce the final solution of frequent itemsets. Experiments based on different data sets have been conducted to evaluate performance of the algorithm.

As the central point of the algorithm is precise guessing of candidate itemsets the future works include significant improvements in collecting statistics and accuracy of guessing. It is necessary to measure what are the costs of getting more complex statistics in the first pass through an input data set and what benefits may be achieved from such statistics in the remaining part of the algorithm. It is also necessary to improve the internal data structures of the algorithm in order to eliminate an impact of inefficient searching methods on the overall performance.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between set of items in large databases. In *Proceedings of ACM SIGMOD*, pages 207–216, May 1993.

[2] R. Agrawal and R. Srijant. Fast algorithms for mining association rules. In *In Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994.

[3] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of ACM SIGMOD*, pages 255–264, 1997.

[4] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM-SIGMOD International Conference on Management of Data (SIGMOD'00)*, Dallas, TX., May 2000.

[5] W. Kloesgen. *New techniques and Technologies for Statistics*, chapter Tasks, methods, and applications of knowledte extraction., pages 163–182. IOS Press, Amsterdam, 1996.

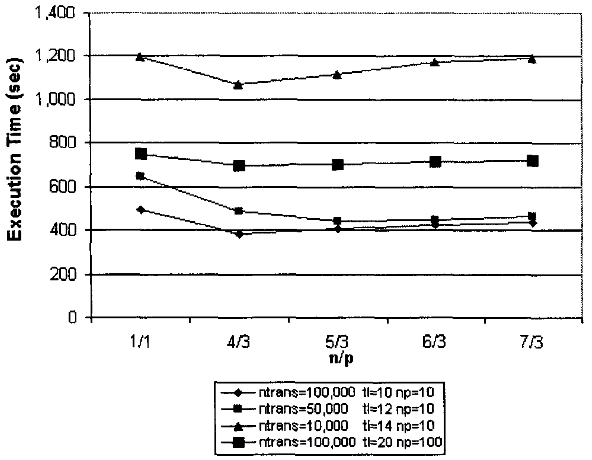

Figure 4. A performance of (n,3) with increasing ratio of (n/p)

[6] H. Mannila, H. Toivonen, and A. Verkamo. Efficient algorithms for discovery in databases. In U. M. F. Eds and Ramasamy, editors, *AAAI Workshop on Knowledge Discovery in Databases*, pages 181-192, Seattle, Washington, 1994.

[7] C. Matheus, G. Piatetsky-Shapiro, and D. McNeill. *Advances in Knowledge Discovery and Data Mining*, chapter Selecting and reporting what is interesting: The KEFIR application to healthcare data, pages 495–516. AAI Press/The MIT Press, Cambridge, 1996.

[8] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of the 21st VLDB Conference*, Zurich, Swizerland, 1995.

[9] T. Senator, H. Goldberg, J. Wooten, M. Cottini, A. Khan, C. Klinger, W. Llamas, M. Marrone, , and R. Wong. The financial crimes enforcement network ai system (fais): Identifying po tential money laundering from reports of large cash transactions. 1995.

[10] H. Toivonen. Sampling large databases for association rules. In *Proceedings fo the 22nd VLDB Conference*, Mumbai (Bombay), 1996.

[11] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *3rd International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 283–286, Newport, California, August 1997.

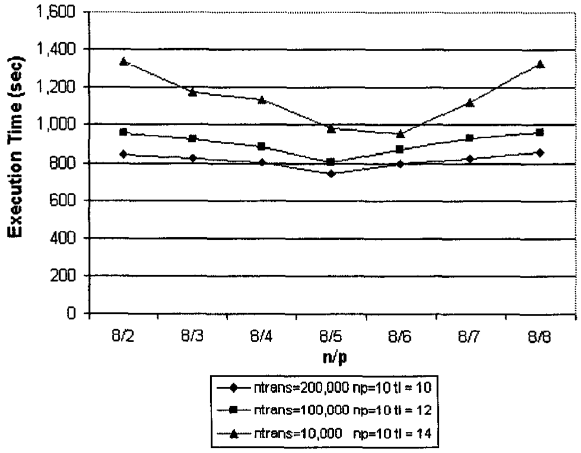

Figure 5. A performance of (8,p) with increasing parameter p