Discovering Association Rules in Large, Dense Databases

Tudor Teusan[1,3],Gilles Nachouki[2,4],Henri Briand[1,3], and Jacques Philippe[1,2,3]

[1]Ecole Polytechnique de l'Univ. de Nantes, rue Christian Pauc, Nantes Cedex 3, France
tjeusan@ireste.fr
[2]Irin, Faculté des Sciences et de Techniques, dép. Informatique Cedex
nachouki@irin.univ-nantes.fr
[3]PerformanSe, Espace Performance, Atlanpole, BP 703, 44481, Carquefou
[4]IUT Nantes, dép. Informatique, 3, rue Maréchal Joffre, 44041 Nantes Cedex

Abstract. In this paper we propose an approach for mining association rules in large, dense databases. For finding such rules, frequent itemsets must first be discovered. As finding all the frequent itemsets is very time-consuming for dense databases, we propose an algorithm that is able to quickly discover an image of the complete set containing all the frequent itemsets. We define what an image is, and we present a genetic algorithm for discovering such an image. To monitor the discovery process we introduce the notion of *dynamics* of the algorithm. To measure the performances of our frequent itemsets discovery algorithm, we introduce the notion of *efficiency* of the discovery process.

1 Introduction

The problem of discovering association rules within databases was introduced in [1]. In this section we give a succinct presentation of the problem.

Given a table of boolean values (the database), the fields of the table are named *items*, and the records, *transactions*. An *itemset* X is a set of items. X is included in a transaction T, if T contains all the items in X. The support of an itemset X, *sup(X)*, is the number of transactions that contain X. An itemsets is *frequent* if its support is greater than a user specified value, *minsup*. An *association rule* is a X->Y rule with X, Y itemsets, $X \cap Y = \phi$. The support of a rule is defined as $sup(X \cup Y)$ and the confidence, *conf*, as $sup(X \cup Y)/sup(X)$ (the conditional probability of Y given X).

Given two user-specified *minsup* and *minconf* values, the problem of association rules discovery consists in finding all the rules that satisfy $sup(X->Y) \geq minsup$ and $conf(X->Y) \geq minconf$. This is classically done in two steps. Firstly all the frequent itemsets Z are discovered - the support value being stored with each itemset. Secondly each Z is repeatedly divided in two disjoint itemsets X, Y ($X \cup Y=Z$). If conf(X->Y)≥minconf, the rule is a valid one. With this approach the difficulty resides in the first step. An efficient algorithm for solving the second step is presented in [2].

Starting from the classical, two-steps approach, other approaches have been developed: simultaneously considering multiple minimum supports [7], pre-storing information about itemsets, for OLAP-like, ulterior processing [3], simultaneously using multiple constraints in the mining process [4], or searching only optimal rules according to some interestingness metric [5].

D.A. Zighed, J. Komorowski, and J. Zytkow (Eds.): PKDD 2000, LNAI 1910, pp. 638-645, 2000.

The characteristics of the database to be mined have a great impact on the performances of a frequent itemsets discovery algorithm. The prototypical data source for association rules mining is the "basket data". "Basket databases" contain only a few items per transaction – they are "rare" databases. However, basket-data is not the only data source for mining association rules. Databases that come from other domains, and even basket-data, can be dense, in the sense that they have [4]:

- many items in each transaction (as compared to the total number of items),
- many frequently occurring items,
- strong correlations between several items

This paper is focalized on mining association rules in dense databases that meet the first criterion. In this context, we present an approach for efficiently discovering frequent itemsets (section 2), provide a genetic algorithm that follows this approach (section 3), and give test results showing the interest of our approach (section 4).

2 Approach for Finding an Image of the Complete Set of Frequent Itemsets

2.1 Defining an Image

As stated in section 1, the difficult problem of finding the association rules within a database consists in generating all the frequent itemsets. The most common solution [1,2] is to start with the frequent itemsets containing only one item and to add them, step by step, more items, until no growth is possible. Each step generates a set of candidate itemsets that are counted in order to select the true frequent ones. On dense databases the main drawback of this approach is the number of candidates that suffers a combinatorial explosion [4], rendering the algorithms very slow and inefficient.

Rather than exhaustively enumerating all the frequent itemsets, we want to provide an image, a "fingerprint" of data. If finding *all* the frequent itemsets is too time-consuming, we are willing to trade the completeness of the result for an approximate set of frequent itemsets, obtained in a significantly shorter time. This approximate set must be a representative image of the set containing *all* the frequent itemsets [10]:

- It should contain a *number* of frequent itemsets that should be close to the *number* of *all* frequent itemsets.
- If the complete set contains different lengths frequent itemset, the approximate set should also contain, roughly in the same proportions, all lengths frequent itemsets.

2.2 Efficiently Discovering an Image

For presenting our approach we begin by enunciating a well-known result: any set of frequent itemsets is downwards closed; any subset of a frequent itemset is also a frequent one. This leads us to defining the notions of *frontier* and *pseudo-frontier*.

Definition 1: The *frontier* is the set of *maximal frequent itemsets*. A *maximal frequent itemset* is a frequent itemset, which is not contained in any other frequent

itemset [11]. Such an itemset gets infrequent by adding it any item not already contained in the itemset.

Definition 2: A *pseudo-frontier* is a set of *potentially maximal frequent itemsets*. A *potentially maximal itemset* is an itemset that, at a certain moment of the frequent itemsets discovery algorithm, is frequent and not contained in an already discovered frequent itemset. A pseudo-frontier is an approximation of the real frontier.

The frontier contains all the information about the frequent itemsets. If it is known determining the whole set of frequent itemsets is trivial: generating all the subsets of the frontier itemsets solves the problem [11].

For efficiently discovering an image, we search an algorithm that converges towards the frontier. At every moment of its execution it must have different approximations of the frontier: pseudo-frontiers. It must obtain these pseudo-frontiers without systematically counting the underlying subitemsets. In an unpredictable (but finite) time the algorithm would discover the true frontier. As our purpose is to discover an image and not the complete set, we must be able to interrupt the algorithm at a certain moment. Once interrupted it must return the itemsets on the current pseudo-frontier, as well as the underlying frequent subitemsets. All these itemsets must form an image in the sense of the two criteria presented in 2.1.

The interruption moment should be decided by regarding the *dynamics* of such an algorithm. *Dynamics* is defined as number of newly discovered frequent itemsets in the time unit. The algorithm should permanently indicate its dynamics. The user, as well as the algorithm itself, should be able to decide to stop and return the discovered image, if the evolution of this parameter shows that the process becomes inefficient.

To measure the effectiveness of the algorithm, we define the *efficiency*, η, of a frequent itemsets discovery algorithm. Given such an algorithm A, a database D, and a threshold minsup, $\eta=|\mathbf{L}|/|\mathbf{T}|$, where L is the set of frequent itemsets discovered by A, T is the set of all the itemsets counted by A, and |X| denotes the cardinal of set X.

The optimal value for η is greater than 1, and is given by a purely hypothetical algorithm that only counts the frequent itemsets on the frontier. If the exact support of every frequent itemset is required, the optimal η is 1, and is given by an equally hypothetical algorithm that counts only the frequent itemsets. A real algorithm, as APRIORI [2], working in real conditions, usually has $\eta<1$.

3 A Genetic Algorithm for Discovering an Image

An efficient pseudo-frontier discovery algorithm has to ignore a significant number of frequent, but few-items, itemsets. It must also not count too many infrequent itemsets, in order to quickly obtain a pseudo-frontier – a synthetic "representation" of the frequent itemsets. A genetic algorithm, in which itemsets represent chromosomes, can generate close-to-the-real-frontier itemsets, without considering all their sub-itemsets, hence finding an approximate image of the complete set of frequent itemsets. If the database is dense such an algorithm gives interesting results, both as execution time, and as efficiency. Figure 1 shows the template of the genetic algorithm, which we will use as a base for further detailing.

```
List find_freq_itemsets (Database db) {
  Population pop, raw_pop;
  Hash_table freq_is;
  Hash_tree  infreq_is;
  pop = initial_population();
  while (! stop_condition) {
    raw_pop = generate_next_raw_pop(pop);
    compute_fitness(raw_pop, db, freq_is, infreq_is);
    store_freq_&_infreq_is(raw_pop, freq_is, infreq_is);
    freq_is = expand_infreq_is(freq_is, infreq_is);
    pop = select_new_population(raw_pop);
    }
return freq_is;
}
```

Fig. 1. The genetic algorithm – template

Some of the topics traditionally related to the design of the genetic algorithms were treated in a classical manner [6], or have straightforward solutions. The *representation* problem has a simple solution: each itemset is represented as an array of bits, each bit standing for a gene. There is a one-to-one mapping between chromosome and itemsets. The *initial population*, generated by `initial_population()`, is filled with chromosomes containing only "0"s. The raw population (fixed size) is generated in `generate_next_raw_pop()`using 1-cutting-point crossover and 1-gene mutation as *genetic operators*. The selection of the next population out of the raw one is done in `select_new_population()` by the classical "roulette" method; a chromosome's chances of survival being proportional to its fitness value.

3.1 Fitness Function

The fitness function is the core of the genetic algorithm. The principles that led us to its formula closely followed the principles and purposes presented in section 2:

- generate as few as possible infrequent itemsets.
- the discovered pseudo-frontier, and the underlying subitemsets must form a set that is rich in every possible length frequent itemsets.
- generate chromosomes that should be close to the real frontier

The first goal can be satisfied if the value of the fitness function for the infrequent chromosomes is much smaller than its value for any frequent one. A minimal chance of survival is given to the infrequent chromosomes.

The last two objectives can be satisfied if we use *a fitness function that encourages the chromosomes to grow in length, as long as, by growing in length, they stay frequent*. Such a function favors the apparition of long itemsets and, implicitly, of all their subitemsets. At the same time, encourages the *evolution* of the frequent itemsets to the point when the growth becomes impossible, that is, to the frontier.

The chromosomes that are infrequent, but near the frontier are equally interesting. If a chromosome grows in length and gets infrequent, but remains near the frontier, we generate its subitemsets that are, most probably, frequent.

Hence, we use a two-branch fitness function: for all the chromosomes that have a support that is less than 80% of minimum support, the fitness value is a fixed, minimal one, so that such chromosomes have a minimal chance of survival. For those chromosomes that have a support greater than 80% of the minimum required support, the fitness function is presented in the following paragraphs.

For synthesizing such a fitness function, we start from the support. The support diminishes as a chromosome grows in length. As the fitness value has to grow, we have to compensate the loss of support. Let $p(X_i)$ be the frequency of the X_i item in the database (ratio between the number of transactions containing X_i and the size of the database). If a chromosome gets from $X_{i1}X_{i2}\ldots X_{ik}$ to $X_{i1}X_{i2}\ldots X_{ik}X_{ik+1}$, an overestimated loss of support is given by $p(X_{ik+1})$. To compensate it, it is enough to multiply the support of the longer chromosome with $1/p(X_{ik+1})$. Such a correction applied for every item within a chromosome – $1/p(X_{i1})*\ldots*1/p(X_{ik+1})$ – insures that the fitness does not diminish with the growth of the chromosome. In reality, the support does not diminish as much as estimated, so the correction determines a fitness function that grows with the length of the chromosome.

Identifying and multiplying the frequencies of the items within a chromosome each time its fitness has to be evaluated is quite time-expensive. Instead of the different $p(X_i)$ we use a mean frequency, α, computed as the geometric mean of the all $1/p(X_i)$. In this way, for $C= X_{i1}\ldots X_{ik}$, $1/p(X_{i1})*\ldots*1/p(X_{ik})$, becomes $\alpha^{length(C)}$.

As already pointed out, we are interested in chromosomes that *evolve* and not in long chromosomes. Therefor, we seriously penalize those chromosomes that are not able to grow any further. For achieving this we use the *age of a chromosome*, computed as the number of generations the chromosome has survived. Taking all the above in consideration, the fitness function for the frequent chromosomes is:

$$\text{fitness}(C) = \frac{\text{support}(C)\cdot\alpha^{length(C)}}{\text{age}(C)}, \quad \alpha = \frac{1}{\sqrt[n]{p(X_1)\,p(X_2)\ldots p(X_n)}} \tag{1}$$

3.2 Managing the Discovered, Frequent and Infrequent, Itemsets

This step is reflected in the **`store_freq_&_infreq_is()`** function in the algorithm template. This is an essential step that permits achieving two major objectives.

The frequent chromosomes that have already been discovered should not be recounted. For this purpose we have used a hash-table in which all the frequent are stored. Newly discovered frequent itemsets, as well as their subitemsets, are stored in the hash-table. The age parameter is also handled using the hash-table.

A newly generated itemset that contains an already discovered infrequent subitemset should not be counted. For achieving this we use a data structure on which we are capable of quickly deciding whether the structure contains a subset of an input, tested-if-infrequent, set. Such a structure is a variant of the hash-tree presented in [2], with itemsets stored not only in the leaves, but at all the levels of the hash-tree.

Once a new population is generated, and the support of its chromosomes has to be

evaluated (as the first step of the fitness evaluation), we search the chromosomes in the hash-table containing the frequent itemsets. For those chromosomes not found to be frequent, we try to find a subitemset in the hash-tree containing the infrequent itemsets. Only those chromosomes neither found to be frequent, nor to be infrequent, are counted. After being counted, they are stored in their corresponding structure.

An apart hash-table is filled with those chromosomes that represent infrequent, but near-the-frontier itemsets (infrequent itemsets with an absolute support greater than 80% of the minimum support). Each new such itemset has its subitemsets generated and counted in the **`expand_infreq_is()`** function.

3.4 Stopping Condition

The stopping condition of the genetic algorithm is related to the *dynamics* of the discovery process as defined in 2.1. At a certain moment of its execution, the genetic algorithm starts to discover few new frequent itemsets, and spend a lot of time on counting infrequent ones. Ideally this situation arrives once all the frequent itemsets are discovered. In practice, as our tests have shown, the algorithm becomes inefficient after discovering 80% to 90% of the total number of frequent itemsets, which is enough for providing an image of the complete set of frequent itemsets. Based on our tests, the algorithm should be stopped (by the user or by itself) once its dynamics gets between 10% and 13% of the initial, “moment 0”, dynamics.

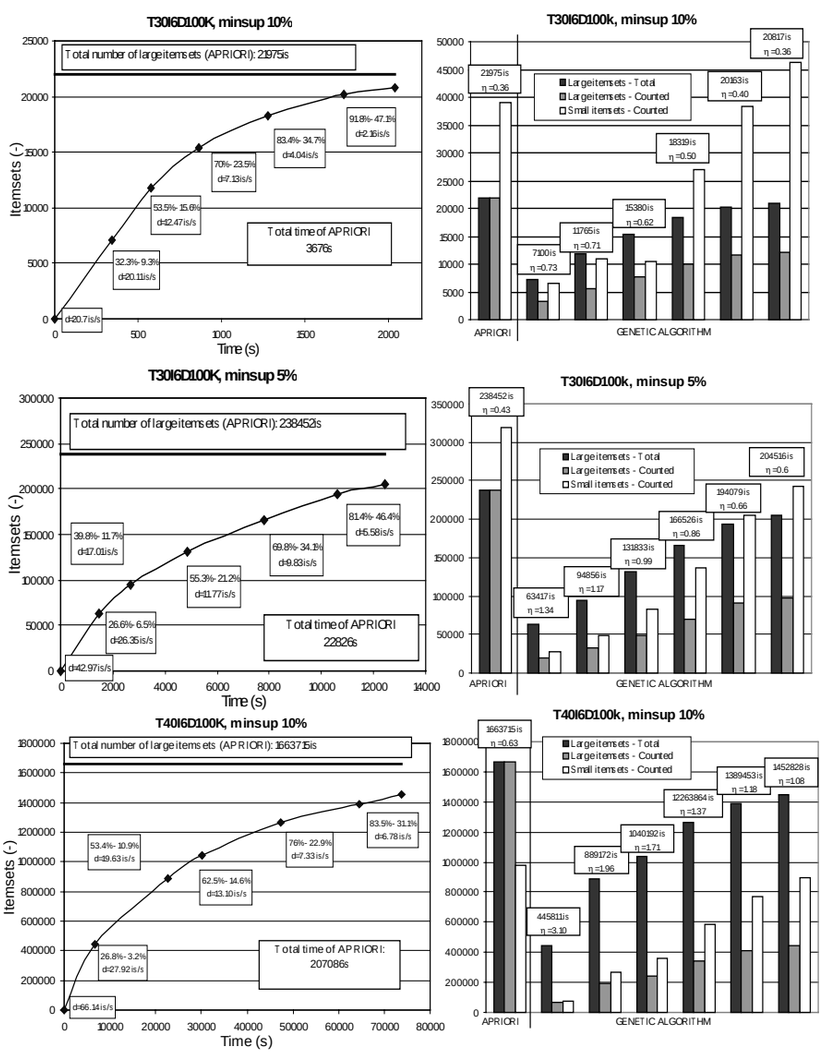
4 Tests

The genetic algorithm can efficiently discover frequent itemsets only if the database is memory replicated. Traditional approaches as APRIORI[2] or Partition[9] try to minimize the I/O overhead by reducing the number of passes over the database. Our main objective is to minimize the number of count operations, even if this is achieved at the cost of an arbitrary number of passes over the database.

In our tests we established the behavior of the genetic algorithm for dense databases. We have mainly studied two parameters: dynamics - d, and, efficiency - η. To show the interest of finding an image instead of the complete set of frequent itemsets, we have compared the results with those obtained by an optimized version of APRIORI (optimized candidate generation, memory-replication of the database).

For generating test data we have used the generation algorithm proposed in [2], and publicly available at [12]. We present 3 test cases, for a relatively reduced, a moderate and a high number of frequent itemsets. For the first two tests we used the same database, T30I6D100k [2], with minsup set at 10% and 5%. The third case was tested on a T40I6D100k database with minsup at 10%. In all cases N, the total number of items, was 100. The test platform was an SGI Origin 200, 4 R10000 processors running at 180MHz, 1 Gbyte RAM, under an IRIX 6.5 operating system.

For each test we present two graphics. The first graphic shows de dependency execution time – number of discovered frequent itemsets. The total number of itemsets and APRIORI's total time are also indicated. For the first 5 points we show the percentage of discovered frequent itemsets (as compared with APRIORI) and the percentage of APRIORI's total time, as well as the dynamics around each point. The second graphic presents the number of discovered frequent itemsets – long, dark gray bars, the number of counted frequent itemsets – shorter, lighter gray bars and the number of counted infrequent itemsets – white bars. The number of discovered frequent itemsets and the efficiency are indicated above each group of three bars.



The next to last point figured in each first type graphic roughly corresponds to the stopping point. The corresponding dynamics are 10.43%, 12.98% and 10.58% of the initial dynamics, that is 10%-13% of the initial dynamics. The genetic algorithm has discovered 80% to 90% of the total set of frequent itemsets, in times ranging from 31% to 47% of APRIORI's time. Efficiency graphics show that the genetic process stays significantly more efficient than APRIORI for up to 80%-83% of the complete set. This makes it well suited for finding an image, in better efficiency conditions.

In the first case, the efficiency for discovering more than 90% of the frequent itemsets is the same as APRIORI's. The time is significantly shorter (about 50%) as the not-count-related operations of the genetic algorithm are simpler than APRIORI's. In the two last cases, the shorter times are a direct result of an increased efficiency.

5 Conclusion

In this article we have presented an algorithm for discovering an image, an approximation, of the complete set of frequent itemsets. For finding this image we have tried to discover the frontier without systematically covering the underlying frequent itemsets. The genetic algorithm we have implemented succeeded in finding the most of the frequent itemsets, for different dense databases and different minimum supports. The efficiency of the discovery process was greatly improved.

References

1. Agrawal, R., Imielinski, T., Swami, A.: Mining Association Rules between Items in Large Databases. Proc. of the 1993 ACM-SIGMOD Int'l Conf. on Management of Data, Washington, D.C., 1993
2. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. Proc. of the 20th VLDB Conference, Santiago, Chile, 1994
3. Aggarwal, C., Yu, P.: Online Generation of Association Rules. Proc. of 14th Int'l Conf. on Data Engineering, 1998
4. Bayardo, R.J., Agrawal, R., Gunopulos, D.: Constraint Rules Mining in Large, Dense Databases. Proc. of 15th Conf. on Data Engineering, 1999
5. Bayardo, R.J., Agrawal R.: Mining the Most Interesting Rules. Proc. of the 5th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, 1999
6. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, Adisson–Wesley Pub. Co., 1989
7. Liu, B., Hsu, W., Ma, Y.: Mining Association Rules with Multiple Minimum Supports. KDD-99 San Diego CA USA, 1999
8. Park, J.S., Chen, M-S, Yu, P.S.: An Effective Hash-Based Algorithm for Mining Association Rules. SIGMOD, 1995
9. Savasere, A., Omiecinski, E., Navathe, S.: An Efficient Algorithm for Mining Association Rules in Large Databases. Proc. of the 21st VLDB Conference, Zurich, Switzerland, 1995
10. Tudor, J., Nachouki G., A genetic algorithm for discovering frequent itemsets in large, dense databases, research report of Nantes University (IRIN) n° 00.9, 2000
11. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: Parallel Algorithms for Discovery of Association Rules. Data Mining and Knowledge Discovery 1-4, Kluwer 1997
12. http://www.almaden.ibm.com/cs/quest/syndata.html. Quest Project. IBM Almaden Research Center, San Jose, CA, 95120