Software & Systems Received 16.03.17
DOI: 10.15827/0236-235X.118.196-206 2017, vol. 30, no. 2, pp. 196–206

EFFECTIVE ALGORITHM FOR CONSTRUCTING ASSOCIATIVE RULES

V.A. Billig [1], *Ph.D. (Engineering), Senior Researcher, Associate Professor, Vladimir-Billig@yandex.ru*

[1] *Tver State Technical University, Nikitin Quay 22, Tver, 170026, Russian Federation*

Constructing associative rules is one of the most important algorithms for extracting knowledge from databases. All modern algorithms are somehow connected with Apriori algorithm proposed in R. Agrawal's and his co-authors' works published more than 20 years ago and now considered classical. The known effective implementations of the algorithm are connected with database compression and presentation of data structure as a tree, which allows effective evaluation of support and other characteristics of associative rules.

The proposed ConApriori algorithm does not use the above idea. We regard database transactions as enumeration given by a scale. This allows instant calculation of the basic algorithm operation determining whether some set is a subset of another set or not. The calculations are reduced to several logical computer commands. Enumeration also allows us to treat the transaction in the internal presentation as a number, preserving the meaning of the transaction elements in their external presentation at the same time.

Another idea used in the algorithm allows us to construct most of the confident rules on the basis of those previously built.

This article provides evidence for correctness of the algorithm as well as evaluate its complexity. We analyze its effectiveness compared with other known algorithm implementations. Possibility of parallelization is also considered.

Keywords: *data mining, apriori, associative rules, support, confidence, lift, enumeration, scale, database, transaction, knowledge discovery, parallel computation, correctness, complexity, anti-monotony.*

With the exponential growth of the stored data, data mining algorithms start playing a great role. Apriori algorithm proposed by R. Agrawal and his co-authors [1–3] is particularly important. The algorithm is based on the antimonotony property of the function calculating the item set frequency. As a result, the set of k-length frequent item sets can be built using (k–1)-length frequent item sets. This allows avoiding an exhaustive search and constructing frequent item sets and associative rules within reasonable time. The idea is used in all algorithms of constructing associative rules.

An important contribution to constructing effective algorithms was made by F. Bodon [4]. He stated that the algorithm's effectiveness is defined by three factors: the means of generating candidates to frequent sets, the structure of data presentation and implementation details. Bodon proposed a tree structure for transaction presentation, showed the structure's effectiveness as compared with other means of data compression.

One of the modern and probably most effective algorithms is the implementation by Ch. Borgelt [5–7] who has been developing it for more than 20 years. In this algorithm, the database is compressed to a transaction tree. The current version of the algorithm is free at his site [8].

Frequent set and associative rule algorithms have a wide application range, thus both the algorithms and the tasks taking into consideration the specific character of rule application are diverse. We will mention only the works closely correlated with the algorithm considered in the present article.

The first work on building multilevel associative rules was that of J. Han and Y. Fu [9]. In [10] the authors analyze effective building of multilevel rules. In the summarizing article by Ch. Borgelt [11] we find a description of several algorithms. The fundamental book by J. Han, M. Kamber and J. Pei [12] devoted to the conceptions and methods of knowledge extraction from database considers various algorithms and approaches to the construction of associative rules in the broad aspect of overall approach to data mining.

The present article is organized as follows. Section 2 introduces the problem. Section 3 describes the basic conceptions of *ConApriori* algorithm and proves its correctness. Section 4 gives evaluation of its complexity. Section 5 is devoted to possible optimizations of the algorithm. Section 6 considers some constraints. Section 7 describes some details of implementation. Section 8 is devoted to approbation of the algorithm and its effectiveness compared with Ch. Borglet's algorithm realization. Section 9 draws a conclusion.

Problem formulation

Consider database *db* with a set of objects:

$$db=\{v_1, v_2, \ldots, v_N\}. \quad (1)$$

Each object v_k has a certain set of properties P_k:

$$P_k=\{q_{k1}, q_{k2}, \ldots, q_{km}\}. \quad (2)$$

The sets P_k are subsets of the set of *Properties* common to all objects:

$$P_k \subseteq Properties =\{q_1, q_2, \ldots, q_M\}. \quad (3)$$

We assume that all properties of the objects are binary, i.e., object v_k can either have or not have the property of q_j. Further, we will show how this constraint can be reduced.

Next, we will consider *dbp* database containing the object properties since we are not interested in the objects themselves:

$$dbp=\{P_1, P_2, \ldots, P_N\}. \quad (4)$$

Let X be an item set of properties:

$$X =\{q_{i1}, q_{i2}, \ldots, q_{ir}\}. \quad (5)$$

Frequency of the item set or its support in database *dbp* will be defined as a proportion of the records of the database containing the set *X*:

$$Support(X) = \frac{|P_k : X \subseteq P_k|}{N}. \quad (6)$$

An associative rule has the form:

$$X \rightarrow Y. \quad (7)$$

Here *X* and *Y* are not empty and disjoint sets of properties:

$$X \cap Y = \varnothing. \quad (8)$$

An important characteristic of the rule is its frequency (support):

$$Support(Rule) = Support(X \rightarrow Y) = Support(X \cup Y). \quad (9)$$

A rule should be considered as an implication, the "if–then" rule. If *X* is in a certain set, then so is *Y*. In other words, appearance of *X* implies occurrence of *Y*. For the rule to be of practical value it should have a high degree of confidence.

Confidence is the second important characteristic of the rule. We will define it as proportion of the records of the database that contain *X* and *Y* among the records containing *X*:

$$Confidence(Rule) = Confidence\ (X \rightarrow Y) = \frac{|P_k : X \cup Y \subseteq P_k|}{|P_k : X \subseteq P_k|}. \quad (10)$$

It is easy to see that confidence can be expressed through the *Support* function:

$$Confidence(Rule) = Confidence\ (X \rightarrow Y) = \frac{Support(Rule)}{Support(X)}. \quad (11)$$

We define the length of the rule *L*(*Rule*) as the sum of the pair (*m*, *n*), where *m* and *n* are lengths of sets *X* and *Y*, respectively.

The goal of the algorithm. We wish to find all *Associative Rules* with support and confidence more than specified minimum values:

$$Rules = \{Rule | Support(Rule) > support\_min \& Confidence(Rule) > Confidence\_min\}. \quad (12)$$

The basic idea of Apriori algorithm. We call the set *X* frequent if *Support*(*X*) > *support_min*.

The following statement is true:

If X is a frequent set, all its subsets are also frequent sets.

A contrapositive statement is more important:

If X is not a frequent set, all its supersets are not frequent sets either.

This property called Antimonotony allows excluding a large number of sets from our consideration. Construction of all frequent rules of length m can be performed by extending the set of frequent rules of length *m*–1. Then in the built set of frequent rules we can choose the rules with the required confidence.

The lift parameter. In addition to frequency and confidence, we consider other characteristics of the rules.

An important additional feature of the rules is the *lift* parameter specifying the correlation between two events, which are antecedent of the rule and its consequent.

Let us discuss the role of the parameter. We will consider *Support* parameter specifying the frequency of set *X* in the database as the probability of event *X*.

Then for the rule $X \rightarrow Y$ and the symmetric rule $Y \rightarrow X$ the *lift* parameter will be defined as follows:

$$lift = \frac{P(X,Y)}{P(X)*P(Y)} = \frac{P(X)*P(Y/X)}{P(X)*P(Y)} = \frac{P(Y)*P(X/Y)}{P(X)*P(Y)} = \frac{P(Y/X)}{P(Y)} = \frac{P(X/Y)}{P(X)}. \quad (13)$$

For independent events, probability *P*(*X*, *Y*) of co-occurrence of events *X* and *Y* is the product of probabilities of occurrence of the events *P*(*X*) * *P*(*Y*), so that the *lift* parameter in this case is equal to 1. For dependent events the conditional probability *P*(*Y*/*X*) (a probability of occurrence of event *Y* on condition that event *X* takes place) can be higher than probability *P*(*Y*). By analogy, the conditional probability *P*(*X*/*Y*) can be higher than probability *P*(*X*). For fully dependent events, when a single event clearly implies the occurrence of another event, conditional probabilities *P*(*Y*/*X*) and *P*(*X*/*Y*) are equal to 1 and probabilities *P*(*X*) and *P*(*Y*) coincide. In this situation the *lift* parameter has a maximum value: 1/*P*(*Y*) = 1/*P*(*X*). For fully dependent events when a single event clearly implies impossibility of the occurrence of another event, conditional probabilities *P*(*Y*/*X*) and *P*(*X*/*Y*) are equal to 0 and the *lift* parameter has a minimum value equal to 0. It implies that values of the *lift* parameter are in the range [0, max(1/*P*(*Y*), 1/*P*(*X*)].

Values of *lift* close to 1 show no correlation between the premise of the rule and its conclusion, so that such rules should be considered useless, even if they are frequent and confident. Values of *lift* parameter greater than 1, approaching a maximum value, indicate a high positive correlation confirming confidence of the rule. Values of *lift* close to zero indicate a negative correlation between the premise and the conclusion. Such rules may also be useful, as well as rules with positive correlation, allowing us to make important conclusions about the existence of links between the sets of the an alyzed parameters.

ConApriori algorithm. The basic ideas

There are two key ideas underlying the *ConApriori* algorithm. The first is related to the method of data presentation. The second is related to the method of constructing confident rules.

Data presentation. How to present database records containing object properties? It is possible to pre-

sent records as a list of properties in csv format (comma-separated values). In this case, the length of each record varies depending on the number of properties owned by the object. Another method (or approach) takes into account the binary nature of the properties. Each record is a string of fixed length M, where M is the total number of possible properties. The string consists of zeros and ones, where zero means absence of the corresponding property of the object and one identifies its availability. A string of zeros and ones can be viewed as a number in binary notation, so in this case, each record is presented by one integer from the range $[0, 2^M - 1]$.

The first method preserves the meaning of each record. However, its drawback is the size of a database depending not only on the number of records in the database, but also on the length of each record. The second method provides a compact presentation of the data but does not save their meaning, which is extremely important in applications, when users of a software system are experts in a relevant application area.

We have tried to avoid the difficulties. Properties of objects are defined as *enumeration*, a type well-known in programming languages. A feature of this type is that the external presentation of data is a line of text describing properties of objects. The elements of enumeration are mapped onto integers which present internal data. Moreover, to present the dataset we choose a special kind of *enumeration* type called *scale*. Scale is characterized by the fact that the k-th element of *enumeration* is displayed as the number 2^k, and the entire set can be viewed as an integer in binary notation. In the binary presentation of a number, the unit in k-th position means that the corresponding object has the k-th property in the *enumeration*.

Let us consider a simple example where three properties characterize a person: *Properties* = {*good*, *smart*, *rich*}.

In *enumeration* with the property "good" given by *scale* maps to 1, "smart" maps to 2, rich maps to 4. The set of properties {good, rich} maps to 5 (in binary notation 101), the set {smart, rich} maps to 6, and the rare set {good, smart, rich} maps to 7.

Enumeration given by *scale* has an important feature. On its elements, which are numbers, we can define logical bitwise operations. This allows us to efficiently perform the basic operation, that is, to determine whether one set is a subset of another set. Let X be a set of properties and P_k be a database record. Objects X and P_k have *enumeration* type given by *scale*. Then $X \subseteq P_k$, when the following Boolean expression is true:

$$(X \,\&\, P_k) = X. \qquad (14)$$

In this expression & is a bitwise conjunction running over binary presentations X and P_k, and = is a comparison operation. We will use the sign | for a bitwise disjunction operation. Operations & and | can be regarded as operations of joining and intersecting of the corresponding sets of properties.

The chosen presentation allows the complex operation of determining the occurrence of a set X in a set Y to be performed in constant time, independent of the size of sets, almost instantly, as the implementation requires only two machine operations of the computer. Hence, the complexity of computing *Support* function is significantly simplified. Complexity of computing the function is $O(N)$, it depends linearly on the size of the database with a small constant factor.

It is necessary to mention a possible limitation inherent to this method of data presentation. While implementing the algorithm, we have a limited number of possible properties due to the finiteness of the interval presenting the integer type of the programming language.

In the following examples of recording algorithms and their fragments we use an implementation of our algorithm in the C# programming language. Note that in C# the maximum integer is 64-bit ulong type, and therefore the number of possible properties may not exceed 64.

Further we will show in more detail how to overcome this limitation and discuss the question of the reasonable number of properties.

Constructing confident associative rules. The basic idea of *Apriori* algorithm is associated with the iterative scheme, which allows constructing frequent rules. Frequent sets of length k are constructed on the basis of frequent sets of length k–1. Construction of rules does not involve infrequent sets . This is the essence of *Apriori* algorithm.

In addition to the properties of antimonotony characterizing frequent sets and rules, *ConApriori* algorithm uses a similar property characterizing confident rules. This property allows us to construct confident rules at step k on the basis of confident rules constructed at step k – 1. The algorithm is based on the following statement:

Lemma 1.

If $X \to Y_k$ is a confident rule, any rule $X \to Y_j$, where Y_j is a proper subset of Y_k is a confident rule.

Let us prove that the statement is true.

Since Y_j is a proper subset of Y_k, the latter can be presented as: $Y_k = Y_j \mid add$. This record takes into account the variables specified by *enumeration*, so that the union of sets can be expressed through disjunction.

The confidence of rule $X \to Y_k$ is computed as follows:

$$Confidence\,(X \to Y_k) = \frac{Support\left(X \left| Y_j \right| add\right)}{Support\left(X\right)}. \qquad (15)$$

The confidence of rule $X \to Y_j$ is computed as follows:

$$Confidence\,(X \to Y_j) = \frac{Support(X \,|Y_j)}{Support\left(X\right)}. \qquad (16)$$

The denominators in formulas (15) and (16) coincide, and the numerator in formula (15) is less than or

equal to the numerator in formula (16) by the property of set frequency. Hence, our statement is true.

Consequence. The negation of the statement is more important than the statement itself. And here antimonotony takes place. If rule $X \rightarrow Y_j$ is not confident, rule $X \rightarrow Y_k$ is not confident either.

It follows that confident rules with a given premise X can be constructed on the basis of the previously built confident rules with the same premise.

Note that the statement only applies to rules with a given premise. It is possible that rules $X \rightarrow Y$ and $Z \rightarrow Y$ are not confident, and rule $X \mid Z \rightarrow Y$ is a confident rule. Therefore, it is not sufficient to build confident rules at step k using only confident rules of step $k - 1$. However, it is sufficient to add confident rules, whose premise is of length k, and the conclusion is a single frequent set. We give more details about it below in considering the iterative scheme of the algorithm.

Iterative scheme of ConApriori algorithm. Consider the iterative scheme of *ConApriori* algorithm. First, at step 1 we construct a set of frequent sets of length 1 and a set of frequent confident rules of length 2 with a single premise and a single conclusion. Next, iteratively, at step k the built set of confident rules of the previous step is extended with the addition of a single frequent item in the premise or in the conclusion. To complete the construction of the set of all frequent confident rules of length $k + 1$, it is sufficient to add confident rules, their premise being taken from the set of frequent examples of length k, and their conclusion being a frequent single set. The process continues as long as it is possible to construct new rules.

Consider the process of extending a set of confident rules at each step. Suppose at step $k - 1$, there is a built frequent confident rule: $X \rightarrow Y$. Let *cand* be a frequent single set satisfying the rule. This means that *cand* is not contained in the union of X and Y of length $k - 1$ and all subsets of length $k - 1$ obtained by replacing one of the elements of the joint set X and Y by *cand* element are frequent. Checking the candidate with the given way of presenting the sets is quite easy. If checking of *cand* is successful, it is possible to generate 6 new rules:

$$X|cand \rightarrow Y; \quad Y \rightarrow X \mid cand; \quad X \rightarrow Y \mid \text{cand};$$
$$Y|cand \rightarrow X \quad X|Y \rightarrow cand; \quad cand \rightarrow X|Y. \quad (17)$$

All 6 rules have the same support – *Support*($X \mid Y \mid cand$). If the support value is less than the minimum value, no new rules are built. For frequent rules, only after checking their confidence they are included in the list of confident rules at the next step. To calculate the confidence of all six rules, it is sufficient to calculate two values of the *Support* function – *Support*($X \mid cand$) and *Support*($Y \mid cand$).

In accordance with lemma 1, when extending confident rule $X \rightarrow Y$ we should only build confident rules of the form $X \rightarrow Y \mid cand$. However, for effectiveness other confident rules of length k are also constructed, since for five added rules it is required to additionally calculate only one value of *Support* function. A simple check is sufficient to avoid duplication in constructing these rules. This technique does not guarantee the construction of all confident rules of length k. However, as practice shows, few rules will have to use premises not included in the set of premises of length $k - 1$ rules.

Lemma 2*. If at step k – 1 a set of all frequent confident rules of length k is built, the considered iterative process at step k will construct a set of all frequent confident rules of length k + 1.*

Indeed, the iterative process at step k builds all the rules of length $k + 1$ of the form $X \rightarrow Y \mid cand$ for each rule of length k of the previous step $X \rightarrow Y$. In accordance with lemma 1, this means that all possible confident rules will be built with the same premises as the rules of the previous step and all possible conclusions. If we add to this set all rules with all possible single conclusions and frequent premises of length k not coinciding with the premise of the previous step confident rules, the full set of confident rules of length $k + 1$ will be built.

Let us now consider the software interpretation of *ConApriori* algorithm. It is based on the *Create_Rules* method, which allows us to build confident rules. A possible implementation is as follows:

```
public void Create_Rules()
    {
      iteration = 1;
         Create_Rules_1();
      bool exist_new_rules = true;
      while(exist_new_rules)
      {
       iteration++;
          exist_new_rules = Create_Rules_K();
      }
         }
```

Lemma 3*. CreateRules method of ConApriori algorithm builds all frequent confident rules.*

To prove the lemma it is necessary to prove the correctness of the following Hoare triads:

$$\{Pred1\}\ Create\_Rules\_1\ \{Post1\} \quad (18)$$
$$\{Pred2\}\ Create\_Rules\_K\ \{Post2\}. \quad (19)$$

Predicate *Pred*1 of the precondition states that before calling *Create_Rules*_1() a database db has been built in the previously described data format. The predicate of the postcondition *Post*1 states that there is a constructed set of the rules – the set of all frequent confident rules of length 1 with a single premise and a single conclusion. The truth of the triad (18) ensures that if the precondition is true, the postcondition of the method will also be true after the completion of *Create_Rules*_1() method.

Precondition *Pred*2 confirms the truth of the conjunction: *exist_new_rules* & *Rules*($k - 1$) – the conjunction of Boolean variable – and the statement about the construction of all confident rules of length k at step $k - 1$. Postcondition *Post*2 can be written as follows:

$$(Rules(k)\ \&\ exist\_new\_rules)\ XOR\ !exist\_new\_rules. \quad (20)$$

The truth of the triad (19) ensures that if the precondition is true, the postcondition will be true after *Create_Rules_K*() method is completed. Informatively, it

means that every time *Create_Rules_K*() method is called in the *while* loop, it builds the set of all confident rules, increasing the length of the rules by one. If the constructed set is not empty, Boolean variable *exist_new_rules* equals *true*. If building a new set of rules is impossible, the Boolean variable gets the value *false*. In this situation the cycle ends, thus completing the process of creating rules.

The predicate *Rules*(*k*) is described as follows: *the set of all frequent confident rules of length k + 1 is constructed at step k.*

This predicate is an invariant of the *while* loop. The truth of the predicate before the execution of the *while* loop guarantees the postcondition *Post*1 of *Create_Rules*_1() method. *Create_Rules_K*() method preserves the truth of the predicate. At the end of the cycle it is guaranteed that the set of all frequent and confident rules is constructed, and it is impossible to extend the set. It is easy to show that the *while* loop will end since the length of rules is limited and cannot exceed the length of the longest record in the database.

For lemma 3 to be true in a particular implementation of the algorithm, it is necessary to build correct *Create_Rules*_1() and *Create_Rules_K*() methods in relation to the pre and postconditions. We will not give a specific implementation as it requires giving details of data presentation, including the necessary internal data. We only note that both methods are not complex, and programmers can easily create a specific implementation ensuring the truth of statements (18) and (19).

*Create_Rules*_1 method first calls another method to build the set of single frequent sets. The set is actively used in further constructions. In the double cycle a set of all frequent confident rules of length 1 is easily constructed by this set, which guarantees the truth of the postcondition of the method. Certainly, prior to calling *Create_Rules*_1 it is necessary to call a method presenting the database in the desired format, thus ensuring fulfillment of the method's precondition.

Implementation of *Create_Rules_K* method is somewhat more complicated than that of the *Create_Rules*_1 method. In its first call in the body of the *while* loop the precondition of the method is performed. It is guaranteed by the postcondition of *Create_Rules*_1 method preceding the *while* loop in *Create_Rules* method. So that at calling *Create_Rules_K* method there exists a *current* set, i.e., the set of all frequent confident rules constructed at the previous step *k* – 1. In the cycle there is constructed an extension to each to the rules according to the set building rules, as described above. All confident rules with a single conclusion and a premise of length *k* not belonging to the premises of the previous step confident rules are added to the newly obtained confident rules. *Create_Rules_K* method preserves the invariant of the *while* loop and builds a new *current* set consisting of all frequent and confident rules of length *k* + 1. In case it is impossible to build the rules of this length, the process ends in creating a complete set of all frequent confident rules. It is guaranteed that the process will be completed.

Algorithm complexity

Let us evaluate the complexity of the basic methods used in *ConApriori* algorithm.

Complexity of Support method. The *Support* method calculating the frequency of the set, largely determines the efficiency of the whole algorithm, since all characteristics of the rules are defined through the *Support* function. For this reason, special attention should be given to both optimization of calculations of the method and the number of method calls. Presentation of sets and database records by *enumeration* given by *scale* allows us to reduce a complex operation of checking the occurrence of one set in another to a finite number of machine operations of the computer. Complexity of the operation is $O(1)$, whereas complexity of the *Support* method is $O(N)$, where N is the number of records in the database.

In implementing the algorithm we tried to minimize the number of possible calls of the function. Further, we consider two possible optimizations to reduce the time of computing the function significantly.

Complexity of Create_OBC_1 method. The method computes a set of single frequent items – *often_set_one*. The method is called only once in *Create_Rule*_1 method constructing frequent confident rules of length 1.

The method is simple. For each property of *Properties* enumeration we compute its support (*Support*) and if it exceeds the minimum value, the property is included in the *often_set_one* set. It is clear that complexity of the method is $O(M * N)$, where M is the number of properties in the *Properties* enumeration.

Complexity of Create_OBC_K method. The method computes a set of the frequent sets of length *k* – *often_set_current*. The method is called only once in the *Create_Rule_K* method constructing frequent confident rules of length $k+1$.

The method builds pairs (*set*, *cand*), where *set* is an element of the *often_set_current* set built at the previous step, *cand* is an element of *often_set_one*. If the pair passes the *Apriori* test and can generate a frequent set, the *Support* function is calculated for it. Frequent pairs are included in set *often_set_current* of the next step.

Complexity of this method is $O(q * M * N)$, where q is the number of pairs having passed the *Apriori* test.

How large is this set? Consider a database of size N consisting of identical records, each of them containing all the elements of the *Properties* set of size M. Then any set will be frequent, any rule $X \rightarrow Y$ will be frequent and confident. The example is only of theoretical interest, since all the rules in this case are uninformative and therefore useless for practice. According to our estimates, when the record has the length an order less than M, the value of q is two to three orders less than N.

Complexity of Create_Rule_1 method. The method is called only once and builds the *current* set – a set of frequent confident rules of length 1. The method builds a rule with a single premise and a single conclusion. If the rule meets the required frequency and confidence, it is included in the *current* set.

Complexity of the method is $O(M2 * N)$, where M is the number of elements of the *often_set_one* set.

Complexity of Create_Rule_K method. The method is called repeatedly in the *while* loop of the main *Create_Rules* method building all the confident rules. At step k, it builds the *current* set of frequent confident rules of length $k + 1$ using the *current* set constructed at the previous step.

Complexity of the method is $O(NCR * M * N)$, where NCR is the number of elements of the *current* set.

Complexity of Create_Rules method. The method builds all frequent and confident rules and determines the final complexity of *ConApriori* algorithm. To evaluate its complexity is not difficult, as here in the *while* loop there is called *Create_Rule_K* method, whose complexity has already been evaluated. Let *iteration* be the number of iterations of the *while* loop. The final evaluation of the complexity of *Create_Rules* method and therefore of the entire *ConApriori* algorithm is:

$$O(iteration * NCR * M * N). \quad (21)$$

Among the four factors, the greatest is parameter N usually exceeding the product of all other factors, since the databases searched for Associative rules can be large.

Algorithm optimization

Optimization of the algorithm can be performed at different levels. To build an effective implementation, we can use low-level languages, such as Assembler or C, where we can write a code taking full advantage of the computer, which the code is built for. This is the way the effective Borglet's implementation [8] is constructed. Implementation in high-level languages, such as, for example, our implementation in the C# language, does not allow generating a code as efficient as a C language code, however it has such advantages as reusability, ease of understanding and modification of the code. Let us consider two possible optimizations applicable to our implementation of the algorithm.

Compressing the database. Usually database records are repeated. Therefore, it is possible to compress the database saving only the unique record and the number of its repetitions. If the average number of repetitions of records is equal to q, parameter N can be reduced to q times, thereby reducing the computational time by the same factor. This operation should be performed only when the average number of repetitions is greater than two.

Algorithm parallelization. Now that all computers are multi-core, we should parallelize the computation to increase the algorithm efficiency. In our case, it is advisable to parallelize the calculation of *Support* method. A database of N records can be divided into k groups and work with them in parallel. The number of groups can be adjusted to the number of cores on the computer the calculation is performed on. Parallelization of the *Support* method is transparent enough and does not cause any difficulties in the use of various tools of parallelism typical of modern programming languages.

This optimization has helped to speed up the computation about 4 times on a four-core computer. Using technologies, such as CUDA, and the GPU with its large number of cores we can achieve a much greater effect of parallelization.

Restrictions on the algorithm

Restriction on the data type. The algorithm assumes all data (object properties) to be binary – an object can either have a property or not. How can we overcome this limitation?

First, consider a property whose values belong to some interval. Such a property can be replaced with k binary properties. For example, a parameter characterizing a patient's temperature can be replaced with binary parameters: *normal*, *high*, *low*, *extra high*, *extra low*. The unnecessary details complicate detection of connections existing between items.

Consider a property of the discrete type that can take a finite number of values. Let these values be $v1$, $v2$, ..., vk. Then the property can be replaced with k binary properties. Whenever the source property takes the value vi, the i-th binary property takes the value of 1, and other binary properties are set to 0.

Restriction on the number of properties. If we use the *enum* type defined in a programming language, the number of properties is limited for enumeration specified by scale. This limitation is connected with the limited interval of integer type the elements of the enumeration are mapped onto, for example, in C# the number of elements of the enumeration cannot be more than 64.

This limitation can be easily overcome. To implement the "long enumeration" we can write our own class *LongEnumScale*. That is exactly what we did in the project implementing the algorithm.

Note that, in our opinion, for meaningful associative rules, the number of properties should not be too large. For such classical applications of associative rules as market basket analysis you should look for multi-level associative rules, where the number of properties is limited at each level.

Details of implementation

The algorithm was implemented in C# in Visual Studio 2015 development environment. The Solution

includes four projects, namely DLL and three interface projects. The class library includes 6 classes.

- LongEnumScale – enumeration defined by a string of arbitrary length. The string is cut into *k* fragments, each being mapped onto the *ulong* integer type. *K* numbers will internally represent each transaction. The technique eliminates restrictions on the length of enumeration.
- Rule – a class defining properties and behavior of the objects presenting rules.
- ConAprioriLongRules – a class constructing frequent sets and confident rules.
- Apriori_DB – a class to work with database. The *Support* method is included in the class.
- Model – a class allowing creation of a model database.
- Converter – a class whose methods allow us to convert the original database into the format required for the algorithm.

Three projects define a user-friendly interface, which allows: creating the model database, converting the database, building confident rules, filtering and saving the rules, as well as building a specific rule.

Approbation and efficiency of the algorithm

Comparison of ConApriori with Microsoft Analysis Services and Arules tools. *ConApriori* algorithm was implemented to solve a real problem of medical diagnosis [13]. For physicians the discovered rules had a scientific interest that allowed them to confirm the hypothesis of a possible early diagnosis of metabolic syndrome. The rules had a high degree of confidence and correlation.

The article [14] shows a comparison of the effectiveness of *ConApriori* Toolkit with two widely used tools for building Associative rules.

- *Microsoft Analysis Services* Tools [15, 16] allows constructing Associative rules for data prepared in Excel and then processed in SQL Server.
- *Arules* Tools are built into RStudio, a popular development environment for the R language [17]. The *Arules* package includes one of the earliest versions of *Apriori* algorithm developed by Christian Borgelt.

In contrast to *ConApriori* algorithm both tools – *Microsoft Analysis Services* and *arules* – can only build rules with a single conclusion.

In most cases the time of constructing rules was minimal for Borgelt's version. However, reading a database implemented by means of the R language, substantially slowed down the whole process of creating rules. A rather complex process of building the rules used in *Microsoft* tools demanded more time in comparison with other algorithms and occasionally led to incorrect results.

It should be noted that the current version of *ConApriori* algorithm has been significantly improved in comparison with the version considered in [14].

Comparison of current ConApriori version with current Borgelt's version. Let us compare the efficiency of the *ConApriori* algorithm and the *Apriori* algorithm developed by Christian Borgelt. Borgelt's algorithm used version 6.24 as of 15.10. 2016 available on his website [8], written in C and having been developed for 20 years. The version is an *exe* file being run from the command line.

All the following results are obtained on a computer with an Intel Core I7, 2.20 GHz processor.

Model database. Using our toolkit we created two model databases of 100,000 and 500,000 records. The number of items in the records is 100. They are divided into four groups of 50, 25, 15 and 10 items. The probability of occurrence of the elements in each group is equal to 0.05, 0.1, 0.15 and 0.7, respectively. The maximum length of transactions is 10.

High probability of occurrence of the forth group transaction items implies appearance of the corresponding associative rules. Table 1 shows the results of calculations for *ConApriori* algorithm for a database of 100,000 transactions.

The last two columns of the table show the number of rules and the time of their construction.

Table 1

Model database 100,000. ConApriori algorithm

Confidence_min	Support_min	N_Rules	T_rules (sec.)
0,3	0,1	90	0,38
0,3	0,05	447	1,7
0,3	0,03	450	1,7
0,3	0,01	1 290	6
0,15	0,03	704	2,3

The time of reading the database of 100,000 transactions is 0.12 seconds.

Table 2 gives the results for Borglet's algorithm. The last column shows the number of nodes in the transaction tree.

Table 2

Model database 100,000. Borglet's Apriori algorithm

Confidence_min	Support_min	N_Rules	T_rules (sec.)	N nodes in Transaction tree
0,3	0,1	90	< 0,01	1 024
0,3	0,05	447	< 0,01	1 024
0,3	0,03	450	< 0,01	1 024
0,3	0,01	1290	0,03	44 717
0,15	0,03	450	< 0,01	1 024

The time of reading the database of 100,000 transactions is 0.05 seconds.

Tables 3 and 4 show similar results for the database containing 500,000 transactions.

Table 3

Model database 500,000. ConApriori algorithm

Confidence_min	Support_min	N_Rules	T_rules (sec.)
0,3	0,1	90	1,7
0,3	0,05	450	8,2
0,3	0,03	450	8,2
0,3	0,01	1290	26,4
0,15	0,03	704	10,4

The time of reading the database of 500,000 transactions is 0.6 seconds.

Table 4

Model database 500,000. Borglet's Apriori algorithm

Confidence_min	Support_min	N_Rules	T_rules (sec.)	N nodes in Transaction tree
0,3	0,1	90	< 0,01	1 024
0,3	0,05	447	< 0,01	1 024
0,3	0,03	450	< 0,01	1 024
0,3	0,01	1 290	0,11	157 741
0,15	0,03	450	< 0,01	1 024

The time of reading the database of 500,000 transactions is 0.24 seconds.

Let us analyze the obtained results. *ConApriori* algorithm demonstrates the efficiency acceptable for practical use. For a database of 100,000 transactions it detects 90 rules in less than one second. The time dependence of the algorithm on the database size and the number of constructed rules is linear.

Borglet's algorithm shows hyper efficiency. It builds and processes 157,000 nodes of the transactions tree in tenths of a second. This high efficiency is due to two factors, namely the quality of the algorithm and its implementation in the C language.

As is noted above, the algorithm does not build rules with multiple conclusions. The number of rules can be very different. For example, *ConApriori* algorithm builds 704 rules, whereas the number of the rules built with Borglet's algorithm is 450, being limited to the rules with a single conclusion. For the latter rules both algorithms build the same set of rules.

We can also note that a database of 100,000 transactions is as representative as the base of 500,000 records. In fact, both databases use the same set of rules, with only one case showing the minimal variance of three rules.

FIMI website databases. The FIMI website [18] is a database repository (Frequent Itemset Mining Dataset Repository) used for comparative analysis of efficiency of algorithms building frequent examples. The databases presented on the website contain records in *csv* format. The meaning of the elements in the records is not specified, each element is represented by a numeric code. The number of records in the database and the number of different elements can be quite large. For example, in *Kosarak* database the number of records is more than 990,000, the number of different elements being more than 10,000.

For algorithms such as ConApriori, whose task is to build confident Associative rules, the direct use of databases like *Kosarak* is hardly reasonable, since there is no point in looking for associations, the number of different elements being so great. Elements should be combined into categories, building multi-level Associative rules. This approach was applied to the *Kosarak* database analysis regarding the first two digits of the element code as the category code. In this case associative rules specify the connection between 99 categories of elements.

It took *ConApriori* algorithm about 0.6 seconds to find 13 rules with the minimum support of 0.1 and minimum confidence of 0.5.

In the same environment *Apriori* algorithm builds a tree of transactions from 512 nodes, in the time commensurate with the measurement error. It builds one rule fewer without detecting the rule with multiple conclusions.

Consider the results for another database from the website – the *accidents* database. The database built, as is noted, on real data, contains 340,183 transactions. The number of different elements is 468.

It took *ConApriori* algorithm 0.6 seconds to find 329 rules with the minimum support of 0.85 and minimum confidence of 0.9.

In the same environment *Apriori* algorithm builds a tree of transactions from 140 nodes, as usual, in the minimum time commensurate with the measurement error. Nevertheless, it builds 156 rules – much fewer than *ConApriori* does — as more than half of the 329 rules are those with multiple conclusions.

It should be noted that despite the high frequency and confidence of the constructed rules, they are informatively useless as the *lift* parameter for all discovered rules is practically equal to one, which indicates the absence of any correlation between the premise and conclusion of the rules. This has been expected, as the number of items contained in the transactions is great and detectable associations are random. In such situations it is wiser to search multi-level rules by combining the elements in groups.

Wiki datasets databases. Databases allowing us to test algorithms for constructing frequent examples and Associative rules are also available on the Wiki website [19]. Let us do the research on the databases containing real data about purchases of confectionery. The site contains four databases of this kind. The number of records in the databases are 1 000, 5 000, 20 000 and 75 000. The number of different elements found in the records is 50, among them 40 confectionery products and 10 drinks of different kinds. Elements in the records are represented by codes of the products, but a meaningful decryption of the codes is supplied including the name, price and other product characteristics.

The databases present a classic pattern of market baskets to search for Associative rules. They ideally suit the effective search for rules by our algorithms. All 50 elements found in the records belong to the same category, so the databases provide a good example of search for associations within one product category. The fact that not only one database but four bases with the same elements are presented, allows us not only to detect the rules, but also to test the hypothesis of stability of the rules, i.e., to see the degree of coincidence of the rules built on all four databases representing the same population.

Table 5 presents the results of applying *ConApriori* algorithm to the four databases.

Table 5

ConApriori algorithm

Confidence_ min	Support_ min	N_Rules	T_rules (sec.)	N transactions
0,45	0,03	75	0,05	1 000
0,45	0,03	65	0,08	5 000
0,45	0,03	45	0,2	20 000
0,45	0,03	42	0,6	75 000

Table 6 presents the results of applying *Apriori* algorithm to the same databases.

Table 6

Apriori algorithm

Confidence_ min	Support_ min	N_Rules	T_rules (sec.)	N transactions	N nodes
0,45	0,03	64	<0,01	1 000	874
0,45	0,03	55	<0,01	5 000	3 783
0,45	0,03	41	<0,01	20 000	12 551
0,45	0,03	41	0,01	75 000	38 842

The effectiveness of the search for the rules in the databases is high for both algorithms. Since Borglet's algorithm does not build rules with multiple conclusions, the number of detectable rules is somewhat less than for *ConApriori* algorithm. However, all the rules built coincide with those built by *ConApriori* algorithm.

What to say about the stability of the discovered rules? Out of the 42 rules discovered in the database of 75,000 records, 30 rules meet the 4 of 4 criteria, that is, they are present in the rules constructed for all four databases. The remaining 12 rules meet the 3 of 4 criteria. These rules are not found in the database of 1000 records. The database has more random associations.

The following table provides examples of stable rules occurring in all databases.

Table 7

Examples of stable rules

The number of records in database	Rules
1 000	33 => 42 support = 0,04 confidence = 0,49 lift = 5,94 3, 18 => 35 support = 0,04 confidence = 0,93 lift = 12,36
5 000	33 => 42 support = 0,04 confidence = 0,56 lift = 6,06 3, 18 => 35 support = 0,04 confidence = 0,94 lift = 10,49
20 000	33 => 42 support = 0,04 confidence = 0,54 lift = 5,82 3, 18 => 35 support = 0,04 confidence = 0,94 lift = 10,13
75 000	33 => 42 support = 0,04 confidence = 0,52 lift = 5,72 3, 18 => 35 support = 0,04 confidence = 0,95 lift = 10,24

We can see that stability of rules and coincidence of characteristics confirms the hypothesis that the constructed rules are a reflection of the real existing relations between elements.

Table 8 shows decrypting the codes of the items included in the rules presented in table 7.

Table 8

Characteristics of the products

The code of element	The code's interpretation
33	'Cheese','Croissant',1.75,'Food'
42	'Orange','Juice',2.75,'Drink'
3	'Opera','Cake',15.95,'Food'
18	'Cherry','Tart',3.25,'Food'
35	'Apricot','Danish',1.15,'Food'

One of the rules says that choosing a cheese croissant is often accompanied by choosing orange juice. Such information can be useful for providing the best possible service to the customers of the bakery.

Do we need the rules with multiple conclusions?

In contrast to the known tools (Borglet's algorithm, Microsoft Toolkit), *ConApriori* builds rules with a multiple conclusion.

Explaining why his algorithm does not build rules with multiple conclusions, Christian Borglet writes: *"Multiple items in the consequents of association rules come at a considerable cost, which must be counterbalanced by some added benefit in order to justify them"*.

He rightly notes that the existence of the confident rule with a multiple conclusion, $x \rightarrow y, z$ implies the existence of four confident rules with a single conclusion: $x \rightarrow y; x \rightarrow z; x, y \rightarrow z; x, z \rightarrow y$. Further he writes: *"Given that a customer placed the item x into its shopping cart, we first use the rule x => y to infer that the customer is likely interested in y, then hypothesize that the customer will place item y into the shopping, and apply the rule x, y → z to the resulting situation to suggest z as well. In this way we achieve the same result, without any loss of information or approximation, that would result from x → y, z"*.

The latter statement is hard to accept. The existence of confident rules with a single conclusion does not imply the existence of a confident rule with multiple conclusions. Meanwhile, in practice, such rules may be of particular interest. When a Bank or some other organization offers the customers a set of services, it is important to understand not only which customers prefer a particular service, but also which customers choose a package of services.

Currently together with a group of physicians we are conducting a research of the risk factors affecting the emergence of children's diseases. It is important to discover the rules giving evidence of the factors influencing the possibility of occurrence of a certain set of diseases.

Similar situations (customers and the services offered, patients and possible diseases) can be considered as the tasks of classification. For them the reduction in the number of constructed rules is due to the fact that the premise of the rules is chosen from one set, and the conclusion is taken from another one.

Conclusion

ConApriori algorithm is based on presenting the database transaction using enumeration given by scale. Database records in this case can be presented by one or more integers. The basic algorithm for the *Support* function is computed in linear time proportional to the size of the database, with the minimum constant. Using capabilities of parallelization we can sufficiently reduce the computation time of this function.

The algorithm provides high performance, practically sufficient for building associative rules in large databases.

It would be of interest to study the possibility of combining the approach used in the algorithm with methods of compressing the database into a tree of transactions.

Acknowledgements. I am grateful to Michael Dekhtyar and Alexander Dekhtyar for their careful reading of the manuscript and important comments for improving the text of the article. My thanks to my sons, Ilya and Yuly Billig for their valuable comments. Many thanks to Nicholas Siabro – we worked together on the experiments on comparative performance analysis of various algorithms. And finally, I would like to specially thank Christian Borglet for the possibility of having free access to the code of his excellent algorithm which he has been developing for many years.

References

1. Agrawal R., Imielinski T., Swami A. Mining association rules between sets of items in large databases. *Proc. of the ACM SIGMOD Conf. on Management of Data*. 1993.

2. Agrawal R., Srikant R. Fast algorithms for mining association rules in large databases. *Proc. 20th Int. Conf. on Very Large Data Bases (VLDB)*. 1994.

3. Agrawal R., Srikant R. Mining sequential patterns. *Proc. 11th Int. Conf. Data Engineering.* IEEE Press, 1995.

4. Bodon F. Fast Apriori Implementation. *Proc. IEEE ICDM Workshop on Frequent Itemset Mining Implementations.* 2003.

5. Borgelt Ch., Kruse R. Induction of Association Rules: Apriori Implementation. *Proc. 15th Conf. on Computational Statistics (Compstat 2002).* Physica Verlag Publ., Heidelberg, Germany, 2002, pp. 395–400.

6. Borgelt Ch. Efficient Implementations of Apriori and Eclat. *Workshop of Frequent Item Set Mining Implementations (FIMI 2003).* Melbourne, FL, USA, 2003.

7. Borgelt Ch. Recursion Pruning for the Apriori Algorithm. *2nd Workshop of Frequent Item Set Mining Implementations (FIMI 2004).* Brighton, UK, 2004.

8. *Ch. Borglet's web pages*. Available at: http://www.borgelt.net/apriori.html (accessed March 23, 2017).

9. Han J., Fu Y. Mining Multiple-Level Association Rules in Large Database. *IEEE transactions on knowledge & data engineering in 1999*, pp. 1–12.

10. Shrivastava1 A., Jain R.C. Performance Analysis of Modified Algorithm for Finding Multilevel Association Rules. *Computer Science & Engineering: An Int.l Jour. (CSEIJ).* 2013, vol. 3, no. 4.

11. Borgelt Ch. Frequent Item Set Mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*. 2012, no. 2(6), pp. 437–456, J. Wiley & Sons Publ., Chichester, UK.

12. Han J., Kamber M., Pei J. *Data Mining. Concepts and Techniques*. Elsevier Publ., 2012.

13. Billig V.A., Ivanova O.V., Tsaregorodtsev N.A. Building association rules in medical diagnosis. *Programmnye produkty i sistemy* [Software & Systems]. 2016, no. 2. Available at: http://www.swsys.ru/index.php?page=article&id=4162 (accessed March 23, 2017).

14. Billig V.A., Korneeva E.I., Syabro N.A. Association rules. Compared analysis of the tools. *Programmnye produkty, sistemy i algoritmy* [Software Products, Systems and Algorithms]. 2016, no. 2. Available at: http://swsys-web.ru/association-rules.html (accessed March 23, 2017).

15. MacLennan J., Tang Z., Crivat B. *Data Mining with Microsoft SQL Server 2008*. Wiley Publ., 2009, 676 p.

16. *Search associations Microsoft*. Available at: https://msdn.microsoft.com/ru-ru/library/ms174916(v=sql.105).aspx (accessed March 23, 2017).

17. Ihaka R., Gentleman R. R: A Language for Data Analysis and Graphics. *Jour. of Computational and Graphical Statistics*. 1996, vol. 5, no. 3.

18. *Examples of the databases*. Available at: http://fimi.ua.ac.be/data/ (accessed March 23, 2017).

19. *Examples of the databases*. Available at: https://wiki.csc.calpoly.edu/datasets/wiki/WikiStart (accessed March 23, 2017).

УДК 004.89
DOI: 10.15827/0236-235X.118.196-206

Дата подачи статьи: 16.03.17
2017. Т. 30. № 2. С. 196–206

ЭФФЕКТИВНЫЙ АЛГОРИТМ КОНСТРУИРОВАНИЯ АССОЦИАТИВНЫХ ПРАВИЛ

В.А. Биллиг, к.т.н., старший научный сотрудник, доцент, Vladimir-Billig@yandex.ru
(Тверской государственный технический университет,
наб. Аф. Никитина, 22, г. Тверь, 170026, Россия)

Конструирование ассоциативных правил является одним из наиболее важных алгоритмов извлечения знаний из БД. Все современные алгоритмы так или иначе связаны с алгоритмом Apriori, предложенным в работах Р. Агравала и его соавторов, опубликованных более чем 20 лет назад и ставших сегодня классикой. Известные эффективные реализации алгоритма связаны со сжатием БД и представлением структуры данных в виде дерева, что позволяет эффективно вычислять поддержку ассоциативных правил и другие их характеристики.

Предлагаемый алгоритм ConApriori не использует вышеназванную идею. Транзакции БД рассматриваются как перечисление, заданное шкалой. Это позволяет практически мгновенно вычислять базисную для алгоритма операцию, определяющую, является ли некоторое множество подмножеством другого множества. Вычисления сводятся к не-

скольким логическим командам компьютера. Перечисление позволяет также рассматривать транзакцию во внутреннем представлении как одно или несколько чисел, сохраняя в то же время смысл элементов транзакции в их внешнем представлении.

Другая идея, используемая в алгоритме, позволяет конструировать большинство достоверных правил на основе ранее построенных достоверных правил.

В статье дается обоснование корректности алгоритма и приводится оценка его сложности. Анализируется эффективность алгоритма в сравнении с другими известными реализациями. Рассматривается также возможность распараллеливания алгоритма.

Ключевые слова: *Data Mining, априори, ассоциативные правила, поддержка, достоверность, лифт, перечисление, шкала, БД, транзакция, обнаружение знаний, параллельные вычисления, корректность, сложность, антимонотонность.*

Литература

1. Agrawal R., Imielinski T., Swami A. Mining association rules between sets of items in large databases. Proc. of the ACM SIGMOD Conf. on Management of Data. 1993.

2. Agrawal R., Srikant R. Fast algorithms for mining association rules in large databases. Proc. 20th Int. Conf. on Very Large Data Bases (VLDB). 1994.

3. Agrawal R., Srikant R. Mining sequential patterns. Proc. 11th Int. Conf. Data Engineering. IEEE Press, 1995.

4. Bodon F. Fast Apriori Implementation. Proc. IEEE ICDM Workshop on Frequent Itemset Mining Implementations. 2003.

5. Borgelt Ch., Kruse R. Induction of Association Rules: Apriori Implementation. Proc. 15th Conf. on Computational Statistics (Compstat 2002). Physica Verlag Publ., Heidelberg, Germany, 2002, pp. 395–400.

6. Borgelt Ch. Efficient Implementations of Apriori and Eclat. Workshop of Frequent Item Set Mining Implementations (FIMI 2003). Melbourne, FL, USA, 2003.

7. Borgelt Ch. Recursion Pruning for the Apriori Algorithm. 2nd Workshop of Frequent Item Set Mining Implementations (FIMI 2004). Brighton, UK, 2004.

8. Ch. Borglet's web pages. URL: http://www.borgelt.net/apriori.html (дата обращения: 23.03.2017).

9. Han J., Fu Y. Mining Multiple-Level Association Rules in Large Database. IEEE transactions on knowledge & data engineering in 1999, pp. 1–12.

10. Shrivastava1 A., Jain R.C. Performance Analysis of Modified Algorithm for Finding Multilevel Association Rules. Computer Science & Engineering: An Int. Jour. (CSEIJ). 2013, vol. 3, no. 4.

11. Borgelt Ch. Frequent Item Set Mining. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2012, no. 2 (6), pp. 437–456, J. Wiley & Sons Publ., Chichester, UK.

12. Han J., Kamber M., Pei J. Data Mining. Concepts and Techniques. Elsevier Publ., 2012.

13. Биллиг В.А., Иванова О.В., Царегородцев Н.А. Построение ассоциативных правил в задаче медицинской диагностики // Программные продукты и системы. 2016. № 2. URL: http://www.swsys.ru/index.php?page=article&id=4162 (дата обращения: 23.03.2017).

14. Биллиг В.А., Корнеева Е.И., Сябро Н.А. Ассоциативные правила. Сравнительный анализ инструментария // Программные продукты, системы и алгоритмы. 2016. № 2. URL: http://swsys-web.ru/association-rules.html (дата обращения: 23.03.2017).

15. MacLennan J., Tang Z., Crivat B. Data Mining with Microsoft SQL Server 2008. Wiley Publ., 2009, 676 p.

16. Search associations Microsoft. URL: https://msdn.microsoft.com/ru-ru/library/ms174916(v=sql.105).aspx (дата обращения: 23.03.2017).

17. Ihaka R., Gentleman R. R: A Language for Data Analysis and Graphics. Jour. of Computational and Graphical Statistics. 1996, vol. 5, no. 3.

18. Examples of the databases. URL: http://fimi.ua.ac.be/data/ (дата обращения: 23.03.2017).

19. Examples of the databases. URL: https://wiki.csc.calpoly.edu/datasets/wiki/WikiStart (дата обращения: 23.03.2017).

Примеры библиографического описания статьи

1. Billig V.A. Effective algorithm for constructing associative rules // Программные продукты и системы. 2017. Т. 30. № 2. С. 196–206; DOI: 10.15827/0236-235X.118.196-206.

2. Billig V.A. Effective algorithm for constructing associative rules. *Programmnye produkty i sistemy* [Software & Systems]. 2017, vol. 30, no. 2, pp. 196–206 (in Russ.); DOI: 10.15827/0236-235X.118.196-206.