Relational Data Mining and GUHA

Tomáš Karban

Department of Software Engineering, Faculty of Mathematics and Physics,
Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
tomas.karban@matfyz.cz

Abstract. This paper presents an extension of GUHA method for relational data mining of association rules. Because ILP methods are well established in the area of relational data mining, a feature comparison with GUHA is presented. Both methods suffer from the explosion of the hypotheses space. This paper shows heuristic approach for GUHA method to deal with it, as well as other methods helping with the relational data mining experience.

1 Introduction

Most data mining methods have been developed for data in the traditional single-table form: rows represent observations (objects) and columns represent variables (properties). However, real-world data are usually stored in relational databases and consist of many related tables. Data preparation methods then convert data stored in an arbitrary form to a single-table representation (e.g. through joins and aggregation). This step simplifies the data inherently and it is manually driven, so its success depends on the creativity of a KDD person.

Relational data mining studies methods for KDD that can use more than one data table directly, without transforming the data into a single table first and then looking for patterns in such an engineered table. In many scenarios, leaving the data in relational form can save a lot of information that can be later expressed in the form of relational patterns. Single-table patterns are simply less expressive. Relational data mining techniques have been mainly developed within the area of inductive logic programming (ILP). ILP was initially concerned with the synthesis of logic programs from examples and background knowledge. Recent development has expanded ILP to consider more data mining tasks, such as classification or association analysis.

On the other hand, GUHA method is an approach of generating and verifying the hypotheses. Deep theoretical background is in [1]. For example, GUHA procedure 4ft-Miner mines for association rules from a single table, while other GUHA procedures (see [4, 7, 8]) mine for other types of patterns. Its main principle is to generate all possible hypotheses based on user task setting, verify them and output the valid ones. In a typical effective implementation, it loads the database into bit strings (i.e. bitmap indexes), and verifies the hypotheses by processing the bit strings.

We can quite naturally extend this approach, so it allows us to search for association rules generalized to more than one data table. We can expect to find patterns that are more complex. Unfortunately, the hypotheses space grows rapidly and the results (valid hypotheses) are more numerous. A new system called Rel-Miner covering methods and ideas presented in this paper is under development.

This paper presents the extension of GUHA method for relational data mining and compares this approach with ILP, which is well established in this area. In the section 2 of this paper, we present the GUHA method, starting from the single-table case and

K. Richta, V. Snášel, J. Pokorný (Eds.): Dateso 2005, pp. 103–112, ISBN 80-01-03204-3.

extending it to relational case. The section 3 covers the ILP approach (mainly relational association rules), compares it to the GUHA and puts both concepts to a universal frame. In the fourth section, we provide more insight to heuristics and optimizations for relational version of GUHA. Section 5 is the conclusion.

2 GUHA Method

2.1 Single-Table Association Rules

An association rule is commonly understood as an expression of the form of X→Y, where X and Y are sets of items. The intuitive meaning is that transactions (e.g. supermarket baskets) containing set X of items tend to contain set Y of items as well. Two measures of intensity of association rule are used, *confidence* and *support*. The goal is to find all association rules of the form X→Y such that the support and confidence of X→Y are above the user-defined thresholds *minsup* and *minconf*. The basic algorithm for mining of association rules is APRIORI [2].

The papers [6, 7] draw an attention to an alternative approach for mining association rules based on representation of each possible value of each attribute by a single string of bits. The association rules have the form $\varphi \approx \psi$ and it is possible to mine for conditional association rules in the form $\varphi \approx \psi / \chi$ as well. Here φ, ψ and χ are conjunctions of boolean attributes automatically derived from many-valued attributes in various ways (called *antecedent*, *succedent* and *condition* respectively). The symbol $\approx$ is called a 4ft-quantifier. The association rule $\varphi \approx \psi$ means that boolean attributes φ and ψ are associated in the sense of the 4ft-quantifier $\approx$. A conditional association rule $\varphi \approx \psi / \chi$ means that φ and ψ are associated (in the sense of $\approx$) if the condition given by χ is satisfied (i.e. associated on a subset of data specified by χ).

$\mathcal{M}$	ψ	$\neg\psi$	
φ	a	b	r
$\neg\varphi$	c	d	s
	k	l	n

Fig. 1. A contingency table of the association rule $\varphi \approx \psi$ in data matrix $\mathcal{M}$

The 4ft-quantifier is formally a boolean condition concerning the four-fold contingency table with frequencies *a*, *b*, *c*, *d*. For every quantifier, we can also use a *natural value of hypothesis*. It is a real number based on the same expression. See the example below.

Among usual 4ft-quantifiers there is a founded implication, which is very similar to "classic" association rule meaning with parameters *minsup* and *minconf* (in the sense [2]). We define it by the following expression over the four-fold table:

$$\frac{a}{a+b} \geq p \wedge a \geq Base .$$

There are parameters $0 < p \leq 1$ and $Base \geq 0$. You can interpret a hypothesis $\varphi \Rightarrow_{p;Base} \psi$ as "φ implies ψ on the level of 100*p* percent and there are at least *Base* objects satisfying both φ and ψ". The natural value of this hypothesis is the expression $\frac{a}{a+b}$.

Other quantifiers describe double founded implication, founded equivalence, above average occurrence or tests of statistical hypotheses (chi-square test of independence, lower critical implication, etc.). You can find more information about this approach in [6, 7]. The project LISp-Miner [4, 5] (namely the 4ft-Miner procedure) is the implementation of this method.

Let us give an example of the association rule with founded implication quantifier:

$$\text{Smoking(> 20 cigs.) \& PhysicalActivity(high)} \Rightarrow_{85\%} \text{RespirationTroubles(yes)}$$

We can read in plain English that 85% of observed patients, who smoke daily more than 20 cigarettes and have a high physical activity, suffer from respiration troubles.

The basic principle behind an effective implementation is usage of the bit strings. We start with a bit string of heavy smokers and bit string of people with high physical activity (these are created directly from database values). We construct the antecedent by ANDing the two bit strings (bit by bit). Note that all bit strings have the same length, which is the total number of patients (objects of this particular database). The succedent is simply a bit string of people with respiration troubles. We compute the frequencies of the four-fold contingency table as the number of bits 1 in the following bit strings: (antecedent & succedent), (antecedent & ¬succedent), (¬antecedent & succedent) and finally (¬antecedent & ¬succedent).

Note that if the database contains objects with missing information (i.e. some null values), we must also maintain a bit string of these null values (as if this was another category) and use them for computation. More details about missing information handling are out of scope of this paper.

In general, we create bit strings out of every category (i.e. possible value) of every attribute. Then we combine some of these bit strings by very fast bitwise operations (namely AND, OR and NOT) into bit strings of antecedent, succedent and possibly condition. After that, we compute a contingency table by counting the number of bits with value 1 and check the value of quantifier. We do this for every hypothesis in the whole hypotheses space specified by the user as a task setting.

2.2 Relational Association Rules

At the beginning, it is important to note that even when we have more than one data table, we still consider one data table as “the main” which stores the basic information about every single object (the same as before). Additionally, we take additional tables with the 1:N relation to the main table. This situation is in database world usually called “master-detail”.

Let us start with a motivation example (see also [6]). We have a database of bank clients. The first table stores basic client information like age, gender, marital status, number of children, and the quality (status) of a running loan. The second table stores money transactions on clients’ accounts. Every transaction has a source and destination account number and amount.

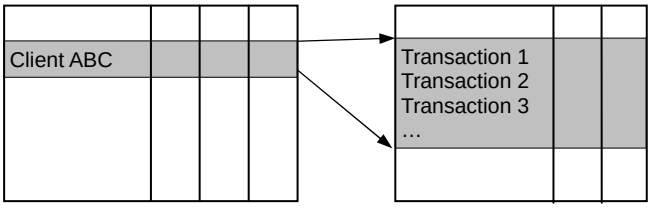

Fig. 2. A simple database schema with 1:N relation

The goal is to get relational association rules that will tell us interesting patterns about the loan quality with respect to both mentioned data tables. The target could be:

MaritalStatus(divorced) & Children(3) & SingleIncome(yes) & AvgIncome(< 1500) $\Rightarrow_{76\%}$ LoanQuality(bad)

In plain English, observed divorced people with single income less than $1500 a month and three children have the probability of 76% of having bad quality loan (troubles with repayments). Here, we used the attributes MaritalStatus and Children directly from master data table. Attributes SingleIncome and AvgIncome were derived from the transactions in the following way:

TransactionAmount(> 500) $\Rightarrow_{93\%}$ SourceAccount(acc345) / Client(ABC)
AVG(SELECT SUM(TransactionAmount) WHERE (TransactionAmount > 0) GROUP BY YearMonth)

We call the attributes derived from detail tables *virtual*. The former attribute is in the form of a founded implication on transaction data, the account number acc345 is a fixed constant for the client ABC. The attribute value for this specific client is 93%. This value is then "virtually added" to the master data table and can be discretized (e.g. yes = more than 90%) to a boolean attribute SingleIncome.

The attribute AvgIncome is a kind of SQL aggregation; for every client it counts the monthly average of the sum of all income transactions.

2.3 Adaptation for Relational Data Mining

In the single-table case, the whole KDD process can be described very easily and precisely by the CRISP-DM methodology [3]. It has separate steps of data preparation and modeling (i.e. data mining). In the data preparation step, data must be discretized before they are used in modeling step. This stays the same even for the relational case; unfortunately, the discretization is needed in the modeling step as well. In the previous example, we have seen the conversion of the strength of implication (93%) into a boolean value, as well as making some categories out of the average income.

We may want the user to prepare such discretization in advance, but the number of such virtual attributes can be very large. From this point of view, we suggest a semi-interactive mining that lets the data mining engine suggest the discretization automatically (based on the distribution of values) with the possibility that the user will change (and fix for future use) some discretization later as he/she sees the attribute used in results.

In other words, the before-distinct steps of data preparation, modeling and evaluation blend. That may introduce new challenges and problems in the whole context of KDD.

At this point, note that the attribute SingleIncome in the previous example was created as an association rule effective on the set of transactions that belong to the specified client. In theory, we can run the search for the association rules recursively, and use their validity as a new attribute at higher level. In practice, we doubt that it is useful to do it for depth more than one, as it leads to result hypotheses, which can hardly be expressed in plain English and meaningfully used in practice. Furthermore, the general recursive approach would be extremely computationally intensive. That means we practically accept only star-schema of database, with one table being master and the rest being detail tables.

2.4 Types of virtual attributes

At this point, we can summarize all methods and possibilities, how we can construct a virtual attribute. We can define a set of aggregation functions that can be applied to attributes in a detail table. The generator can try to run all aggregation functions on all attributes. However, in practice there are semantic limits on usage of certain aggregations on particular attributes. For example, there is no point computing the sum of values that denote the current amount of goods in a warehouse in every month. Unfortunately, the user must specify these limits. Among the usual aggregates, we may consider sum, average value, mean value, standard deviation, minimum, maximum, first and third quartile, count and many others. More advanced aggregates may be integration or linear regression (e.g. for time series). It may prove useful to allow the user to provide user-defined aggregate functions as well.

In the previous example, we have seen the usage of an arbitrary association rule as a virtual attribute (see also [6]). It can describe (among others) implication, equivalence, statistical independence and even relations on general contingency tables [8].

The last category of virtual attributes is existential quantifiers. Within this category, it is necessary to use constants bound to a single object, or global constants specified by user. Note that we have seen (in the example above) the literal SourceAccount(acc345), which used a constant acc345 bound to a single bank client (other clients may have other sources of income). The existential quantifier can be used for example to define a boolean attribute PaysTaxes, such that there exists a transaction with target account number equal to government account (which is a constant defined by user).

3 Other Methods of Relational Data Mining

3.1 Inductive Logic Programming

The most widely adopted method for relational data mining is inductive logic programming (ILP). A summary of the methods and taxonomy is given in [11, 16]. In [15] the author notes the following. While approaches such as ILP inherently suffer from a large search space, integrating aggregation operators further increases the hypothesis space. Moreover, the search space is less well-behaved because of the problem of non-monotonicity, which renders traditional pruning methods in ILP inapplicable. Clearly, we must give up the idea of performing a systematic search for a good hypothesis and the search must be more heuristic.

As we are proposing the extension of GUHA method, which mines for association rules, we focus on the part of ILP that concerns relational association rules as well.

3.2 WARMR

The paper [12] presents an algorithm WARMR, which is an extension to APRIORI algorithm [2] for mining of relational association rules. You can find the implementation of WARMR in Aleph [17] or ACE [18].

APRIORI algorithm is a levelwise search through the lattice of itemsets. It works in two phases. First, we search for so-called frequent itemsets; we examine every transaction and count the number of transactions containing a given itemset (as a subset). In the second phase, we create association rules from frequent itemsets that meet minimum specified confidence and support (see paragraph 2.1).

This two-phase approach is the same for WARMR. The notion of itemsets is generalized to *atomsets*, which are sets of logical atoms constructed from database relations. We count the number of examples that are covered by a given atomset. This is implemented in Prolog such that an atomset is converted to an existentially qualified conjunction and then run as a Prolog query. If it succeeds, the atomset covers the specified example (a counter is increased); if it fails, the atomset does not cover the example.

Now having the set of frequent k-atomsets (atomsets containing exactly k atoms), we construct candidate (k+1)-atomsets. This is analogous to APRIORI, although new step is introduced to prune atomsets. This extra step involves running a theorem prover to verify whether the atomsets are contradictory or redundant with respect to a user-specified clausal theory. This theory provides the user with a powerful tool to specify taxonomies, mutual exclusion relations, and other types of background information known to hold in the database. It contributes to the efficiency of the algorithm as well as to the quality of the output.

The example of association rule over multiple relations can be the following:

$$\text{likes(KID, dogs)} \,\&\, \text{has(KID, fish)} \Rightarrow \text{prefers(KID, dogs, fish)};\ \text{conf. 65\%, supp. 20\%}$$

This association rule states that 65% of kids, who like dogs and have fish, prefer dogs to fish; furthermore, 20% of all kids like dogs, have fish and prefer dogs to fish. There can be variables instead of constants in association rules as well, like:

$$\text{likes(KID, dogs)} \,\&\, \text{has(KID, A)} \Rightarrow \text{prefers(KID, dogs, A)}$$

3.3 Comparison of GUHA and WARMR

We can separate all algorithms and methods of relational data mining to two basic categories, according to the first part of [14]. The methods using aggregation to bring the information from detail tables to the master table belong to the first category. These methods use the aggregates as a description of the whole set of data that corresponds to the examined object from the master table.

The second category comprises of algorithms that handle sets by looking at properties of their individual elements. For example, a part of the pattern can be of the form "there exists an x in the set such that P(x) holds", where P can be a relatively complicated condition. Most ILP systems follow this approach.

We can call methods of the first category *aggregating methods* and methods of the second category *selective methods*. To make this distinction clearer, see the two simple concepts in the following the example:

- people whose average account balance is above $10.000
- people who pay telephone by encashment
  (i.e. there exists a certain type of transaction)

Clearly, WARMR algorithm is using an existentially qualified conjunction to count the number of examples covered by a given atomset. That puts it into the category of selective methods. On the other hand, ideas presented in this paper as an extension to GUHA method cover both these categories. Aggregations create virtual attributes easily and are very suitable for subsequent processing. Furthermore, using association rules as a basis for virtual attributes is a more advanced concept than simple aggregation, but it can be placed into aggregating category as well, as association rules describe the whole set of detail data related to currently examined object.

It is very easy to cover the existentially qualified conjunction case as well. In fact, the principle is very similar to association rules because they have conjunctions in common. If we create a conjunction of literals (out of attributes in detail data table), we can create a corresponding bit string and look for the bit with value 1 (in the segments belonging to individual objects of master table). If we find the bit with value 1, the existential quantifier is satisfied. Note that it is not necessary to count the number of 1 as in the case of association rules, so we can implement it substantially faster.

Also, note that there is a major difference between virtual attributes created as aggregations and association rules. While aggregations are computed from the detail data directly (e.g. by calling SQL queries or running more complex user-defined functions), creating virtual attributes out of association rules implies discretization of detail data (if necessary), creating bit strings of discrete categories, running a kind of single-table data mining engine (GUHA-based) and finally discretization of the results to allow further use as a virtual attribute in the master table.

In the paper [9], authors also note a practical difference between aggregating and selective methods. While the former approach is more suitable to highly non-determinate domains (usually in business), the latter is geared more towards structurally complex domains, such as molecular biology or language learning. This holds in comparison of WARMR and proposed Rel-Miner as well. While Rel-Miner is supposed to work with simple database schema (master-detail) containing many-valued categories and even real numbers, WARMR works with arbitrarily complex data structures with only “simple data”, searching for frequent structural patterns.

Continuing this differentiation, WARMR produces association rules spanning data tables to an arbitrary depth, bound together by the unique identification of the basic object. It is also capable of working with recursive tables. The output is a structural pattern that holds frequently, looking to individual sub-databases created from the original one by selecting rows regarding a single object at a time from all tables. On the contrary, Rel-Miner produces association rules that you can always comprehend the same way as in the single-table case, only “enhanced” by virtual attributes, which were created at run time from detail tables.

4 Making Rel-Miner Feasible

4.1 Complexity of relational hypotheses

As we already mentioned, the hypotheses space for the relational association rules is enormous. The total number of hypotheses grows with combinatorial numbers considering the number of attributes of the master table. While the number of possible aggregations is linear to the number of attributes in detail tables, the number of virtual attributes made of association rules suffers the combinatorial growth.

We may want to limit this growth considerably by limiting the number of virtual attributes (made of association rules in particular) that can be used simultaneously. The rationale is that every such an attribute strongly contributes to the complexity of the result. The process of data mining must keep in mind that it should deliver novel, potentially useful and ultimately understandable patterns. Using complex association rules as a virtual attributes makes the interpretation actually very difficult, so there is no point in pushing too hard with it.

4.2 Reordering of Hypotheses Evaluation

Another useful concept is a suitable reordering of the execution. The idea is to verify the hypotheses starting with the simple ones and going through more and more complex ones. Even if we admit the impossibility to process the entire hypotheses space, we want to be able to get the reasonable sample of results. Evaluation of the complexity of a hypothesis is inherently a heuristic function and the system should allow the user to fine-tune it to his/her needs.

This reordering implies that in the case of early stopping of the computation at any time, we are guaranteed to have finished the evaluation of all simpler hypotheses, which makes the partial result still somehow usable.

Unfortunately, this concept makes the process of verification rather inefficient. In the paper [13], the authors recommend using of the query-packs. That means to evaluate hypotheses in such order that similar hypotheses (having many attributes in conjunction in common) are evaluated together. It saves computing of the common part repeatedly. Analogous idea is in literal and cedent traces that were implemented in LISp-Miner project (see [7]).

We suggest a compromise solution for Rel-Miner, where the whole hypotheses space is divided into jobs. Every job is restricted in the hypothesis complexity (see previous paragraph) and is “local” in terms of cedent traces. Individual jobs can be evaluated optimally, even though there is some overhead among the jobs.

4.3 Distributed Computing

Thinking about the jobs, we can naturally make Rel-Miner distributed to many computers/processors. It is not difficult to design a “job manager” that will assign individual jobs to different computers. The only difficulty is with an excessive network communication; if we take into account the expected size of data in a bit string representation (see [7]), we can establish a bit string cache at every computer, so it will fetch every bit string at most once and store it for later use.

4.4 Limiting the Amount of Output

Together with the hypotheses space explosion, we can expect increased number of valid hypotheses that will go to output. We must keep in mind that our data mining is supposed to provide potentially useful results. Hence, the number of hypotheses (as results) that the user can review is limited. It is certainly possible to employ post-processing methods, such as filtering and sorting (by user-provided criteria). The possibility to limit the total size of output is nonetheless convenient. This limitation must preserve the most important hypotheses, based on their natural value.

4.5 Importance of Post-Processing

To continue with notes from the previous paragraphs, it is important to point out that the post-processing step (i.e. the evaluation according to CRISP-DM methodology [3]) is more important than in a single-table case. We also mentioned that blending the steps of data preparation, modeling and evaluation makes the overall usability worse.

We suggest creating a visual browser tool that will display the hypotheses lattice as a dynamic graph. In this graph, nodes are individual hypotheses and edges connect nodes that have something in common. The strength of the edge (displayed as length and/or thickness) denotes the measure of similarity between the connected hypotheses. The size (and/or color) of the node denotes its natural value. Unfortunately, making of such a tool is a task for a separate and generally independent research.

4.6 Possibility of User Interaction during Computation

With respect to possibly “endless” task run, it is necessary to allow viewing of interim results. During browsing of the results, the user should be able to make slight modifications to the task setting easily. When the user sees for instance an unreasonable hypothesis, because it describes semantically incorrect aggregation or an unsuitable combination of attributes, he/she can prevent the generator to work further on it (recall that although “local”, more complex hypotheses are left for future computations). On the contrary, boosting the priority of some hypothesis can mitigate its complexity, so the hypotheses in the close neighborhood will be computed sooner. Research on this topic is again out of scope of this paper.

4.7 Hypotheses in Natural Language

The paper [9] shows, how to formulate mechanically association rules to reasonable sentences in natural language. You can make a limited language model, which mainly consists of formulation patterns independent of the application domain. Afterward, you supply domain vocabulary that is closely related to attributes in the database. Another area of research is to extend this method for use in relational data mining.

5 Conclusion

We have presented the extension to GUHA method for relational data mining. This approach is different from the ILP approach in many aspects, namely the target application domain. Association rules produced by Rel-Miner can contain aggregations, inner association rules (valid on detail data tables) or existential attributes. This makes them more general compared to association rules produced by WARMR algorithm. On the other hand, Rel-Miner is meant to run on a simple “master-detail” database schema, not going to depth more than 1. ILP methods in general are capable

of mining in arbitrarily complex database structures, focusing more on the structure of the database itself, than on complex computations with attribute values.

Both methods suffer from an explosion of the hypotheses space and both have unique ways to deal with it. Among the most important, Rel-Miner uses heuristics to prune the space and chooses the verification of hypotheses in the order from simple to more complex.

Reference

1. Hájek, P. – Havránek, T.: *Mechanizing Hypothesis Formation – Mathematical Foundations for a General Theory*. Springer-Verlag, 1978
2. Aggraval, R. et al.: *Fast Discovery of Association Rules*. In Fayyad, U.M. et al.: Advances in Knowledge Discovery and Data Mining, pp. 307-328, AAAI Press / MIT Press, 1996
3. CRISP-DM Methodology – http://www.crisp-dm.org/
4. Šimůnek, M.: *Academic KDD Project LISp-Miner*. In Abraham, A. – Franke, K. – Köppen, M. (eds.): Intelligent Systems Desing and Applications (Advances in Soft Computing series), ISBN 3-540-40426-0, Springer-Verlag, 2003, pp. 263–272
5. Academic KDD software system – LISp-Miner: http://lispminer.vse.cz/
6. Rauch, J.: *Interesting Association Rules and Multi-relational Association Rules*. In Communications of Institute of Information and Computing Machinery, Taiwan, Vol. 5, No. 2, May 2002, pp. 77-82
7. Rauch J. – Šimůnek, M.: *An Alternative Approach to Mining Association Rules.* In Lin, T.Y. – Ohsuga, S. – Liau, C.J. – Tsumoto, S. (eds.): Data Mining: Foundations, Methods, and Applications, Springer-Verlag, 2005
8. Rauch, J. – Šimůnek, M. – Lín, V.: *Mining for Patterns Based on Contingency Tables by KL-Miner – First Experience*. In Lin, T.Y. – Ohsuga, S. – Liau, C.J. – Hu, X. (eds.): Foundations and Novel Approaches in Data Mining, Springer-Verlag, 2005
9. Strossa, P. – Černý, Z. – Rauch, J.: *Reporting Data Mining Results in a Natural Language*. In Lin, T.Y. – Ohsuga, S. – Liau, C.J. – Tsumoto, S. (eds.): Data Mining: Foundations, Methods, and Applications, Springer-Verlag, 2005
10. Džeroski, S. – Lavrač, N. (eds.): *Relational Data Mining*, Springer 2001, ISBN: 3-540-42289-7
11. Džeroski, S.: *Multi-Relational Data Mining: An Introduction.* In ACM SIGKDD Explorations Newsletter, Vol. 5, Issue 1, 2003, pp. 1-16
12. Dehaspe, L. – De Raedt, L.: *Mining Association Rules in Multiple Relations*. In Proceedings of the 7th International Workshop on Inductive Logic Programming, Volume 1297, LNAI, pp. 125-132, Springer-Verlag, 1997.
13. Blockeel, H. – Dehaspe, L. – Demoen, B. – Janssens, G. – Ramon, J. – Vandecasteele, H.: *Improving the Efficiency of Inductive Logic Programming Through the Use of Query Packs*. In Journal of Artificial Intelligence Research, volume 16, 2002, pp. 135-166
14. Blockeel, H. – Bruynooghe, M.: *Aggregation Versus Selection Bias, and Relational Neural Networks*. In IJCAI-2003 Workshop on Learning Statistical Models from Relational Data, SRL-2003, Acapulco, Mexico
15. Assche, A. van – Vens, C. – Blockeel, H. – Džeroski, S.: *A Random Forest Approach to Relational Learning*. In2 Dietterich, T. – Getoor, L. – Murphy, K. (eds.): ICML 2004 Workshop on Statistical Relational Learning and its Connections to Other Fields, pp. 110-116
16. Blockeel, H. – Sebag, M.: *Scalability and Efficiency in Multi-Relational Data Mining*. In ACM SIGKDD Explorations Newsletter, Vol. 5, Issue 1, 2003, pp. 17-30
17. A Learning Engine for Proposing Hypotheses (Aleph), ILP system http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html
18. The ACE Datamining system, http://www.cs.kuleuven.ac.be/~dtai/ACE/