Computing and Informatics, Vol. 26, 2007, 345–366

DRAP-INDEPENDENT: A DATA DISTRIBUTION ALGORITHM FOR MINING FIRST-ORDER FREQUENT PATTERNS

Jan Blaťák, Luboš Popelínský

KD Group at Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno, Czech Republic
e-mail: xblatak@mail.muni.cz, popel@fi.muni.cz

Revised manuscript received 8 December 2006

Abstract. In this paper we present dRAP-Independent, an algorithm for independent distributed mining of first-order frequent patterns. This system is based on RAP, an algorithm for finding maximal frequent patterns in first-order logic. dRAP-Independent utilizes a modified data partitioning schema introduced by Savasere et al. and offers good performance and low communication overhead. We analyze the performance of the algorithm on four different tasks: Mutagenicity prediction – a standard ILP benchmark, information extraction from biological texts, context-sensitive spelling correction, and morphological disambiguation of Czech. The results of the analysis show that the algorithm can generate more patterns than the serial algorithm RAP in the same overall time.

Keywords: Frequent patterns, inductive logic programming, parallel and distributed data mining, propositionalization

1 INTRODUCTION

Frequent pattern mining is one of the most important data mining tasks [1, 29]. A *frequent pattern* is a conjunction of literals that covers at least n instances, where n is a threshold value called the *minimal frequency threshold*, which is given by the user. The number of covered instances is usualy called the *support*. Frequent patterns have many potential applications, e.g. mining *association rules* [1], *feature*

construction [6, 16, 17, 40], *propositionalization* [27], and *classification* [28]. Because of the fact that the number of frequent patterns can be huge, several techniques have been developed to condense the theory. The two most important are *closed* [34, 42] and *maximal* [29] frequent patterns. *Closed frequent* patterns are the most specific patterns from classes obtained by splitting the set of all patterns frequent in a data set according to their support such that each class contains only those patterns which have the same support. In this paper we deal with the latter. A *maximal frequent pattern* is a frequent pattern which cannot be specialized (i.e., enlarged by adding a literal) without decreasing the coverage under the given threshold. Every frequent pattern is a *sub-pattern* of some maximal frequent pattern and therefore the number of maximal patterns is much lower than the number of all frequent patterns.

Many algorithms for finding frequent patterns from attribute-value or transactional data – *propositional data* – have been developed since the introduction of the *Apriori* algorithm [3], the first efficient algorithm for mining frequent patterns and assocation rules. The algorithms utilize different search strategies and introduce improvements that decrease the amount of memory used, the I/O operations performed, and the computer time needed.

For more complex data, e.g. multi-relational, object-oriented data – otherwise known as *relational data* [18] – only a few systems have been proposed, although mining in such data has recently become more important. In most situations the propositional representation is not expressive enough for describing complex data such as molecules, DNA strings, spatio-temporal data, and texts. The first algorithm for mining frequent patterns in relational data was WARMR [15] which was based on the Apriori algorithm [3]. Although WARMR used several enhancements (e.g., *query packs* [7] for pattern evaluation), it was inadequate for mining in really large or complex data (see, e.g., [12]). After WARMR, two new algorithms were presented that tried to solve the bottlenecks of WARMR. The first, FARMR [32], uses a *tree*-like data structure for storing patterns together with special data representation to speed up coverage computation. The second, RADAR [12], solves the problems with exorbitant memory requirements of both WARMR and FARMR. All these systems search for all frequent patterns.

In previous work [5, 6] we presented RAP, a system for mining first-order maximal frequent patterns. RAP exploits the fact that many frequent patterns are redundant, i.e. cover the same instances, and therefore are not important for the user. RAP mines "interesting" maximal first-order frequent patterns (i.e. patterns that are as different as possible) at first by utilizing different search strategies and several pruning methods. By relaxing the pruning criteria the user can obtain all frequent patterns.

Regardless of the search method or data structures used, no serial algorithm is able to cope with really large amounts of data in reasonable time. This is because of the limitations of computers, such as lack of operational memory or computational power of current processors. Moreover, with the systems mentioned above it is not possible to mine frequent patterns in distributed databases, e.g., data stored in several different branches of a company. Therefore a great deal of attention has

been paid recently to efficient parallel and distributed data mining algorithms.

In this paper dRAP-Independent (distributed RAP with independent computation), an algorithm for mining first-order frequent patterns in distributed data, is presented. A "distributed" environment or system denotes a parallel system – a collection of computers (computational nodes) which do not share any resources (e.g., memory, bus, hard drives, or time). We have two objectives: to speed up computation when a large amount of data is processed and to decrease the communication overhead. We show that dRAP-Independent can find more frequent patterns than RAP in the same overall time while the system uses only two rounds of communication.

The task of frequent pattern mining is outlined in the next section. Parallel and distributed algorithms for mining frequent patterns are described in Section 3. In Section 4 the dRAP-Independent algorithm is introduced. Section 5 gives information about experiments. A detailed analysis of dRAP-Independent performance is given in Section 6. Summary and directions of future research are in Section 7.

2 FREQUENT PATTERN MINING

Frequent pattern mining was introduced by Agrawal et al. [1] as a problem to find all subsets of $\mathcal{I}$ that are *frequent*. The symbol $\mathcal{I}$ denotes the set of all items that can occur in a transactional database $\mathcal{D}_T$. A subset of $\mathcal{I}$ is frequent if is subsumed by at least σ transactions from $\mathcal{D}_T$. The value σ is the *minimal frequency threshold*, which is given by the user. This definition is appropriate only for data represented as a transactional database or as a single relation. There are many problems, e.g. data in chemistry describing the structure of organic molecules, that are difficult to represent in this way. To be able to mine frequent patterns from such data it is necessary to utilize more complex schema for representing and accessing data and knowledge mined. Many techniques for processing complex data are already available, e.g. relational, object-relational, object-oriented, and deductive databases. The situation is more difficult in the case of the knowledge that is to be mined, because database systems use different techniques for accessing data, e.g., SQL or DATALOG queries.

Mannila and Toivonen [29] defined frequent pattern mining as a theory extraction problem. For a given database $\mathcal{D}$, a language $\mathcal{L}$ for expressing the properties or defining subgroups of the data, and an interestingness predicate q for evaluating whether a sentence $\varphi \in \mathcal{L}$ defines an interesting subclass of $\mathcal{D}$, the task is to extract a theory, $\mathcal{T}h(\mathcal{L}, \mathcal{D}, q)$, such that $\mathcal{T}h(\mathcal{L}, \mathcal{D}, q) = \{\varphi \in \mathcal{L} \mid q(\varphi, \mathcal{D}) \textit{ is true}\}$. This definition is general enough for describing many data mining tasks. Frequent pattern mining from a relational database may be defined as the task of finding all SQL queries that select at least σ instances from a relational database $\mathcal{D}$ (the criterion q).

In this work we use this definition. In our case the database $\mathcal{D}$ is a deductive database. We define the language $\mathcal{L}$ as a subset of first-order logic. It consists of

conjunctions of atoms without function symbols. Each conjunction φ can be seen as a DATALOG query and we say that φ covers an example $e \in \mathcal{D}$ (denoted as covers(φ, e)) if φ can be proved from $\mathcal{D}$. Further, we define *support* of the query φ as $\sup(\varphi, \mathcal{D}) = |\{e \in \mathcal{D} \mid \text{covers}(\varphi, e)\}|$, i.e., the number of examples covered by φ. The evaluation criterion q is satisfied for φ if support of the query is at least as high as a given value σ.

We define a specialization/generalization relation between patterns in accord with [29] and [15] as a partial order $\preceq$ on the patterns in $\mathcal{L}$. We say that φ_1 is more general than the pattern φ_2, if $\varphi_1 \preceq \varphi_2$. We also say that φ_2 is more specific than φ_1. In this paper the relation is defined as the θ-subsumption [31] and we say that φ_1 *subsumes* φ_2 or φ_1 is *sub-pattern* of φ_2. The relation $\preceq$ is a *monotonic specialization relation* with respect to q, i.e., if for a database $\mathcal{D}$ and $\varphi, \psi \in \mathcal{L}$ is $q(\psi, \mathcal{D})$ true and $\varphi \preceq \psi$, then $q(\varphi, \mathcal{D})$ is also true. This allows us to prune the search space by not considering any specialization of patterns which are not frequent (*infrequent* patterns). Finally, we can define *maximal frequent pattern* as a pattern whose specializations are not frequent. Each frequent pattern is subsumed by some maximal frequent pattern.

customer(*allen*).	*parent*(*allen*, *bill*).	*buys*(*allen*, *wine*).
customer(*bill*).	*parent*(*allen*, *carol*).	*buys*(*bill*, *cola*).
customer(*carol*).	*parent*(*bill*, *zoe*).	*buys*(*bill*, *pizza*).
customer(*diana*).	*parent*(*carol*, *diana*).	*buys*(*diana*, *pizza*).

Fig. 1. An example of a Prolog database

Example 1. Let $\mathcal{D}$ be a database as in Figure 1 ([15]) and the argument of the predicate *customer* be an example (instance) identifier in $\mathcal{D}$. The meaning of the predicate *parent*(X, Y) is that X is mother or father of Y, and *buys*(X, Y) means that X buys an item Y. Let us have a pattern φ defined as the formula *customer*(X), *parent*(X, Y), *buys*$(Y, pizza)$[1]. This pattern succeeds for each customer who is parent of the person Y that buys *pizza*. There are two such customers in $\mathcal{D}$ (*allen* and *bill*), such that support of φ is 2. As an example of a specialization of φ we can consider pattern *customer*(X), *parent*(X, Y), *buys*$(Y, pizza)$, *parent*(Y, Z). Its support is 1, because it holds only for customer *allen*.

3 PARALLEL AND DISTRIBUTED FREQUENT PATTERN MINING

Parallel and distributed algorithms for mining frequent patterns can be categorized into three different groups, algorithms for distributing data, algorithms for distributing frequent patterns, and those which distribute both data and patterns. Another

[1] The symbol "," is used instead of $\wedge$.

criterion is the communication schema and the architecture of the system. The great majority of systems use the *master-worker* architecture, in which a master assigns work to workers. In the rest of this section we give an overview of the principal algorithms for mining frequent patterns in propositional data. Further details can be found in [24]. We pay special attention to the Partition algorithm of Savasere et al. [38], which we have adapted for relational data. Then we describe the most important parallel ILP systems. Finally we describe the PolyFARM algorithm of Clare and King [11], the only algorithm for parallel minig of relational frequent patterns.

Agrawal and Schafer [2] adapted the Apriori algorithm [3] for mining association rules over distributed data and designed a Count Distribution (CD) algorithm. The idea is that the computation of coverage can be done very efficiently if the data is stored in the main memory. CD iterates candidate generation and evaluation steps until no frequent pattern found in a previous iteration can be specialized without decreasing its support under σ. In step $k \in \mathbb{N}$ each worker $i \in \{1, \ldots, P\}$, where P is the number of computational nodes, generates patterns of length k that are frequent in $\mathcal{D}_i$. Subsequently, each node sends its patterns to each other node, *global support* of all candidates is computed, and pruning is performed. Global support of a pattern φ is computed as $\sup(\varphi, \mathcal{D}) = \sum_{i=1}^{P} \sup(\varphi, \mathcal{D}_i)$. Several variants that slightly improve performance of CD have been proposed. None of them is appropriate for mining in dense data (data with enormous number of non-maximal frequent patterns), because all candidates have to be stored in the memory of each worker. It is also difficult to achieve high workload balance.

In order to cope with memory problems Agrawal and Shafer [2] designed a Data Distribution (DD) algorithm and Han et al. [23] proposed IDD (Intelligent Data Distribution) which addresses the main problems of DD. Instead of splitting data horizontally each computational node processes only a part of the set of candidates. In the first step of each iteration k the candidates of length k are distributed among nodes. Each node computes support of all its candidates and sends the information to the other nodes. Subsequently, pruning is performed and frequent patterns are processed in the next iteration. Good workload balance is achieved by assigning an equal number of candidates to each node. The algorithm performs much better than the CD algorithm when the number of computational nodes increases and very fast communication is available. For long patterns the communication overhead increases, which significantly decreases the performance of the algorithm.

Han et al. [23] developed a Hybrid Distribution (HD) algorithm which combines both strategies, distributing the data and the candidates. Nodes are split into several groups and the database is distributed over these groups. The candidates are partitioned and processed by one node in a group. The HD algorithm achieves much better performance than pure IDD and CD algorithms.

Konstantopoulos [26] adopted the ideas of the CD algorithm for learning first-order classification rules. He implemented a parallel version of the Aleph[2] classifier. This algorithm performs – when a simple background knowledge is used – better

[2] `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`

than the serial one. The ILP classifier which distributes the database and also the hypothesis space was proposed by Dehaspe and De Raedt [14]. Fonseca et al. [19] implemented and compared several strategies for parallel rule learning, e.g., parallel coverage testing (by analogy with CD), parallel exploration of search space (IDD), and data parallel algorithm. They proposed a new algorithm, a pipelined data-parallel algorithm [20], which achieved an excellent speed increase when it was used for classification of three benchmark data.

3.1 Partition

Partition [38] is an algorithm for mining in horizontally fragmented data. The algorithm splits data into partitions such that each partition can fit into main memory and exploits the property that each pattern that is frequent in the entire database has to be frequent in at least one partition.

The algorithm first splits the database $\mathcal{D}$ into P partitions. In the second step, each node utilizes a level-wise algorithm (e.g., Apriori) for finding all patterns such that $\sup(\varphi, \mathcal{D}_i) \geq \sigma \cdot \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$ in partition $\mathcal{D}_i$, where $i \in \{1, \ldots, P\}$ and $|\mathcal{D}_i|$ denotes the number of instances in partition i. These patterns are denoted as *locally frequent* patterns. After all locally frequent patterns are mined (all partitions are processed) the set of all potentially frequent patterns (*globally frequent* patterns) is generated. Lastly, a second pass over the database is performed to compute the support of these patterns.

The advantage of Partition is that only two passes over data are needed. High speed up can be obtained because the first step has no additional communication overhead. Moreover, any algorithm for mining locally frequent patterns can be utilized in Partition. To achieve the best performance it is necessary to minimize the effect of data skew across the partitions. Also the size of partitions plays an important role. Finding locally frequent patterns in a small data is fast, but a large number of patterns that are not globally frequent can be generated.

3.2 PolyFARM

PolyFARM (Poly-machine First-order Association Rule Miner[3]) by Clare and King [11] is a system for distributed mining in horizontally partitioned data. It utilizes the same strategy as the CD algorithm and unfortunately inherits also its disadvantages. PolyFARM is a prototype system written in the functional language Haskell and uses DATALOG for representing both the data and the found rules. It allows the definition of Is-A hierarchies for values. A Beowulf[4] cluster is utilized as an underlying parallel environment.

The patterns are generated in three steps – generation of candidate patterns, computation of local support of candidates in each data partition and merging the

[3] `http://www.aber.ac.uk/compsci/Research/bio/dss/polyfarm/`

[4] `http://www.beowulf.org/`

information about support to obtain global support for all patterns. Each step is performed by an independent module. There is one instance of a module *Farmer* for generating candidate patterns and one module *Merger* for merging local supports. The module *Farmer* acts as a master in the master-worker architecture. *Merger* communicates with nodes running a *Worker* module that computes coverage of candidates in a data partition. It prunes infrequent rules and sends the frequent ones back to the node running the *Farmer* module.

4 dRAP FRAMEWORK

Relational data mining has become more important in recent years, because we need to deal with very complex data which cannot be analyzed by propositional data mining tools. Great attention is therefore paid to the design of efficient algorithms for mining in multi-relational data. Nevertheless, there are still several problems that serial systems have not yet been able to solve. In fact, there is no efficient solution for mining long relational frequent patterns in large amounts of data or in dense data (data containing many frequent patterns) because:

- processing candidates and auxiliary information exhausts all accessible resources, especially memory.
- evaluation of candidates (coverage computation) consumes a great deal of time by testing all possible instantiations of variables (especially if the tested candidate does not cover the tested example).

Another problem is that data may be stored in a distributed database and cannot be stored on one node, e.g., data from several branches of some organization. In that case no serial algorithm can be used directly.

In Section 1 we described RAP, an any-time algorithm for mining maximal frequent patterns in first-order logic. RAP solves the first problem and can be used for finding long patterns in dense data, but it cannot mine in really large amount of data (gigabytes of data) in reasonable time. In the rest of this section we propose a solution to the second problem, an efficient evaluation of candidates. We describe dRAP-Independent an algorithm for distributed finding of maximal first-order frequent patterns. It is based on the Partition algorithm of Savasere et al. (cf. Section 3.1) and utilizes RAP for finding locally frequent patterns. dRAP-Independent is also intended for searching interesting patterns like the RAP system but by relaxing the pruning criteria it can generate all frequent patterns.

4.1 The dRAP-Independent Algorithm

The dRAP-Independent algorithm (cf. Figure 2) finds maximal frequent patterns in three steps. In the first step the database is split into partitions. The obtained partitions are used as input data for the serial RAP algorithm running on each worker, and locally frequent maximal patterns (LFMP) are generated. The main

difference between Partition and dRAP-Independent lies in the last step. While the former is designed to find all frequent patterns, the latter is intended to generate only the maximal ones.

There may be two problems with a pattern from LFMP – 1) it is not globally frequent, or 2) it is globally frequent but not maximal. Omitting the infrequent patterns, as it is done in Partition, would decrease performance. Therefore, each globally infrequent pattern generated in the second step is generalized – its longest frequent sub-pattern is found. Because of the incomplete search used it is not possible to decide whether the found pattern is maximal or not. Additional specialization steps need to be performed before the pattern is added to the set of globally frequent maximal patterns (GFMP).

Algorithm: dRAP-Independent
Input: database $\mathcal{D}$, list of computational nodes *Nodes* and the threshold σ
Structures: set of locally frequent maximal patterns (*LFMP*), set of globally frequent maximal patterns (*GFMP*)
Output: the set of globally frequent maximal patterns *GFMP*

for each $i \in$ *Nodes* **do**
  generate partition $\mathcal{D}_i$ of $\mathcal{D}$ and determine σ_i
end for
for each $i \in$ *Nodes* **do**
  generate locally frequent maximal patterns *LFMP*
  (run the serial RAP algorithm on $\mathcal{D}_i$ and σ_i)
end for
wait until all the nodes finish computation
for each pattern $\varphi \in$ *LFMP* **do**
  compute global support $\sup(\varphi, \mathcal{D})$ of φ
  while $\sup(\varphi, \mathcal{D}) < \sigma$ **do**
    generalize pattern φ
    (remove last literal from φ)
    compute $\sup(\varphi, \mathcal{D})$
  end while
  if ($\varphi \notin$ *GFMP*) **then**
    generate maximal pattern φ_s from φ
    (run RAP system on $\mathcal{D}$, σ and φ)
    add φ_s into *GFMP*
  end if
end for

Fig. 2. The dRAP-Independent algorithm

4.2 Partitioning the Data

The information related to an example in relational data can be spread over several relations, which makes splitting the data more difficult than in the propositional case. Moreover, the examples may depend on each other. This means that for counting coverage we need to process all examples. This setting is usually called *learning from entailment*. An efficient method for splitting such data is not known. Another setting is called *learning from interpretations* [13, 8]. Each example is seen as an independent part of the database. In this setting it also may be quite complicated to separate out all the information related to a particular example. This is because new complex intensional predicates can be defined in the given background knowledge. Therefore partitioning is often performed only on example identifiers. An example of such an identifier is primary key in a relational database (e.g., the predicate $customer(X)$ in Example 1). This solution is not appropriate for data which cannot fit into main memory.

In the past, it was believed that assigning examples into partitions randomly would be sufficient and that it would not be necessary to pay special attention to the method used for splitting data. This was because of the fact that by mining in large volume of data it was expected each partition would contain a similar sample of data. Cheung et al. [10] showed that this is not true. They presented the first in-depth study of the impact of the method utilized for partitioning data on performance of distributed frequent pattern mining. They defined two metrics for measuring dissimilarity of generated partitions (*skewness* and a *workload balance*). High skewness and workload balance resulted in higher performance of their Count Distribution like algorithm, because a much smaller number of candidate patterns that cannot be frequent were generated. They developed a fast simple algorithm which splits data into partitions such that skewness and workload balance are maximized.

In this work, we use learning from interpretations and we also split only the set of key identifiers. The algorithm assigns a subset of key identifiers to each node randomly such that it preserves the same class distribution in all partitions.

4.3 Finding LFMP

One of the advantages of the Partition algorithm is that an arbitrary system for finding locally frequent patterns can be utilized. This version of dRAP-Independent uses the serial RAP algorithm. Distributing the computation does not result in additional restrictions on the search and pruning strategy, so that it is possible to mine all frequent patterns in the same manner as in the case of RAP. For further details see [5, 6].

4.4 Generating GFMP

Merging information about support and dealing with globally infrequent patterns have a crucial impact on performance of the algorithm. As we mentioned in Sec-

tion 4.1 it is not a good idea to omit all globally infrequent patterns. The locally frequent maximal patterns are usually very long – they may consist of ten or more literals. To generate such patterns is quite expensive and finding them usually takes the major part of the time. While a pattern from LFMP may be globally infrequent, some of its generalizations can be frequent. It is often sufficient to remove just one or two literals from the pattern to increase its support over the given threshold. Another problem which makes the generation of GFMP difficult is that there are locally frequent maximal patterns that are not maximal in the whole data. Therefore it is necessary to specialize such patterns.

To generate the set of globally frequent maximal patterns dRAP-Independent performs both transformations – generalization and specialization of LFMP. In the first step all globally infrequent patterns that were generated in the second step of the algorithm are generalized (the set of patterns obtained is called GFMPc – GFMP candidates). In the second step the GFMPc set is given as an input to the serial RAP algorithm. It specializes all elements such that just one maximal frequent pattern is generated from each candidate pattern. Finally GFMP is created. This version of dRAP-Independent performs these two steps on a single node. This is inefficient and it decreases performance. To overcome this obstacle it is necessary to utilize a message-passing interface library for Prolog, which was not available in the time when dRAP-Independent was implemented. To integrate such a library into the algorithm is one of goals of our future work.

We should remark here that there are many tasks for which the maximality of a pattern is not important. An example of such a task is feature construction for classification. In this case shorter patterns would be more useful than maximal ones. Similarly, we can use all the patterns generated, not only the maximal ones (i.e., we relax the support constraint). For example, it is not crucial when a pattern covers 299 examples instead of 300 if it is used as a new feature for classification. Therefore the algorithm can skip both the generalization and the specialization steps.

4.5 Controlling the Execution

Because no communication among nodes is possible when locally frequent maximal patterns are generated a situation may arise in which almost all computational nodes have finished their work but the systems is waiting for a single node. The reason is that the node has to process more complex examples, so that much longer patterns or more patterns are generated. Moreover, in the first case the patterns found usually are not globally frequent.

To cope with such a situation the dRAP-Independent algorithm allows the user to define additional constraints, such a maximal length and number of patterns, or maximal execution time. These are adopted from the RAP system. We defined a new constraint, which handles the problem mentioned above. The user can specify the maximal number of waiting nodes. The execution of nodes running the RAP system is aborted if as many nodes have finished their work as the given value.

5 EXPERIMENTAL EVALUATION

We had two objectives in the experimental evaluation – to analyze performance of dRAP-Independent and to check whether distributed processing is useful for solving real-world problems. In the first step mutagenicity prediction [39], a well-known benchmark data, was used to analyze the performance of the algorithm. In the second phase, we utilized dRAP-Independent for propositionalization (cf. Section 5.1) of three text mining problems (cf. [4]). In the rest of this section the propositionalization problem is stated, an appropriate metric for measuring performance of the algorithm is defined, and the solved tasks are described. The results of the analysis are in Section 6.

5.1 Propositionalization

Learning discriminant functions can be realized by reformulating the relational problem into an attribute-value form and then applying an efficient propositional learner. This process is usually called *propositionalization* [27]. In the attribute-value representation, examples are usually represented as feature-vectors of a fixed size in the form $f_1 = v_1 \wedge f_2 = v_2 \wedge \ldots \wedge f_a = v_a$, where f_i are the features and v_i the values. A new feature is defined [27] as a conjunction of (possibly negated) literals

$$f_i(X) \ :- \ Lit_{i,1}, \ldots, Lit_{i,n_i} \tag{1}$$

where $i, n_i \in \mathbb{N}$, X refers to an individual (a line in a relational table or example) and $\forall k \in \mathbb{N}$ literal $Lit_{i,k}$ is defined in background knowledge $\mathcal{B}$. Dehaspe et al. [16] showed that frequent patterns, especially frequent Datalog queries, can be successfully used as new Boolean features. In that work, the body of the clause (the right-hand side) in Equation 1 was replaced with a frequent pattern. The value of a new feature was set to 1 for those examples which were covered by the frequent pattern used and to 0 for all others.

5.2 Evaluation Criteria

A number of metrics for measuring performance of parallel algorithms have been proposed (cf. [22]). The most common one is *speed up*, which captures the relative gain of solving the problem in parallel. This is defined as the ratio between the time consumed to solve the problem on a single processor (denoted as *serial runtime*, T_s) and the time taken to solve the same problem on p identical processors (*parallel runtime*, T_p). For both serial and parallel system the time elapsed between the beginning and the end of execution is measured.

This definition of speed up is not appropriate for this work, because it does not take into account the fact that the number of patterns mined may differ. We define the speed up as

$$S = \frac{T_s}{T_p} \cdot \frac{C_p}{C_s},$$

where T_s and T_p denote the serial and parallel runtime and C_s/C_p is the number of patterns found by using the serial/parallel algorithm.

5.3 Task Descriptions

It is known that the quality of a model generated by any ILP system strongly depends on the background knowledge used. If the background knowledge is too simple it is not possible to describe all the data dependencies well. Conversely, too complex background knowledge can slow down the execution and make the system unusable. We used different types of background knowledge to observe if and how the performance of dRAP-Independent depends on its complexity. In this section the data, background knowledge, and settings used are described.

5.3.1 Mutagenicity Prediction

The mutagenicity prediction task [39] is a well-known ILP benchmark used to measure and compare the performance of ILP systems. The goal is to build a model for predicting whether a given chemical compound causes mutagenicity or not. Such a model is usually called *SAR* (*Structure-Activity Relationships*). In our experiments we used $\mathcal{B}_4$ – the most complex type of background knowledge for mutagenicity prediction data. This contains information about atoms and their properties (partial charges, values of ϵ_{LUMO} and $LogP$), bonds, definition of 2-D structures (rings) and predicates for testing the connections between 2-D structures and atoms which does not belong to any structure. We generated patterns that cover at least 25 % (43 from 169) molecules. The best-first search was utilized and each candidate pattern φ was pruned whether a maximal pattern ψ such that φ was the prefix of ψ was found already. The continuous values were discretized with equal frequency discretization built in RAP.

5.3.2 Information Extraction from Biomedical Text

The information extraction (IE) task we solved was proposed at the Learning Language in Logic 2005 (LLL05[5]) workshop. The aim is to generate patterns which describe gene-protein interactions from biomedical text. The dataset consists of words, their lemmas, lists of morpho-syntactic relations in a sentence, agents, targets and lists of interactions (pairs agent-target[6]). The data also contains a dictionary of all named entities, i.e. candidate agents and targets. An example of a typical sentence analysed is: "*GerE stimulates cotD transcription and inhibits cotA transcription in vitro by sigma K RNA polymerase, as expected from in vivo studies, and, unexpectedly, profoundly inhibits in vitro transcription of the gene (sigK) that encode sigma K.*" It contains the following interactions: *GerE-cotD*, *GerE-cotA*,

[5] `http://www.cs.york.ac.uk/aig/lll/lll05/`

[6] `http://genome.jouy.inra.fr/texte/LLLchallenge/`

sigma K-cotA, *GerE-SigK* and *sigK-sigma K*. There are 576 examples (103 positive, 473 negative interactions) in the data.

Each generated pattern begins with a literal that introduces two entities (genes or proteins). The rest of the pattern consists of predicates for exploring words which are syntactically related to these genes. The value of the minimal frequency was 10 % and the best-first search was employed. All frequent patterns up to the length of four literals which were subsumed by some known maximal pattern were pruned. The time of execution was limited to 15 minutes. The tests were performed on 660 gene-protein pairs.

5.3.3 Context-Sensitive Text Correction

Classical spell checkers do not use context and therefore are not able to detect errors such as a correct word being used in the wrong place. For example, in the sentence "*I'd like a peace of cake.*", the word *peace* is a correct English word but the intended word was *piece*. The goal is to generate patterns which describe the context of the misspelled words. We used a variant[7] of the TDT2[8] corpus morphologically tagged by Carlson et al. [9] with the SNoW-based part-of-speech tagger. The whole corpus consists of about one million English sentences from six news sources. For our experiments we selected the words *among* (7 119 occurrences) and *between* (13 378 occurrences). The generated patterns consisted of literals which describe properties (morphological category, etc.) of the five nearest words to the left and right of the keyword. Two kinds of background knowledge were tested. In the first ($\mathcal{B}_1$) the absolute position of the words from context was considered. The second introduces words regardless of the absolute position of the word in the sentence. Defined predicates represent the relative relation between two words (e.g., a word w_1 is on the left/right of the word w_2). The latter background knowledge ($\mathcal{B}_2$) is more general, but computing the support of patterns is more time-consuming. Before finding patterns we split the data into a training set (16 398 examples) and a test set (4 999 examples). We set the minimal frequency threshold to the value of 10 % of the training set and restricted the mining time to 90 minutes.

5.3.4 Morphological Disambiguation of Czech

The last task we tackled was the morphological disambiguation of Czech [36]. For purposes of testing and explanation, only the word *je* was processed. This word has two readings[9], which are: the pronoun *them* (e.g. "*Vidím je.*", "*I see them.*") and the verb *to be* (e.g. "*On je řidič.*", "*He is a driver.*"). For classifying we used sentences from DESAM [33], an annotated corpus of Czech. Unambiguous data was used for mining of frequent patterns and two versions of this data – unambiguous

[7] `http://l2r.cs.uiuc.edu/~cogcomp/Data/Spell/`

[8] `http://www.ldc.upenn.edu/Projects/TDT2/`

[9] There is in fact a third reading of *je* – it can be an interjection. We ignore this, as it does not occur in our data.

and ambiguous – were used for evaluating patterns. The second one was generated by adding all possible grammatical readings to each word. We used the full range of tags from the tagged DESAM corpus, which covers part-of-speech, gender, number, and case. The contexts consisted of the two words closest to the word *je* to the left and right.

The value of the minimal frequency threshold was set to 10 % of the size of the learning set and all prefixes of frequent patterns which had already been shown to be frequent were pruned. We used 50 examples from each class (noun – k3, verb – k5) when RAP was used and 200 examples when distributed mining on 4 nodes was performed. Pruning criterion was relaxed for distributed mining. The pruning was applied only to the patterns which consisted of 5 or more literals. The time of execution of dRAP was shortened to be the same as in the serial case. None of the training data contained any ambiguity. The evaluation of the generated models has been performed on 300 unseen examples from each class. This data was ambiguous.

6 RESULTS

6.1 Mining Maximal Frequent Patterns

The experiments with mutagenicity prediction data were performed on 2, 4, 8 and 16 AMD Athlon^{TM}XP 1600+ computers with 512 MB of memory. The nodes were connected with a one-hundred megabit Ethernet. We tested the impact of additional time constraint to the speed up measure. The execution time was restricted to 10, 20, ..., 100 % of the time consumed by the serial RAP algorithm for mining patterns from the whole data.

The serial RAP system found 9 maximal patterns (3 of the length 4, 5 of the length 5 and 1 pattern containing 7 literals) with the given settings. It performed 155 discretization steps, which consumed 20 % of the execution time. The computation of support of the candidate patterns took 76 % of execution time. The rest 4 % were consumed by specialization of patterns (2 %), pruning the candidate set (1.2 %), and loading the data. The whole execution took 53 seconds.

		# of patterns			Speed up	
Limit	Interrupted	LFMP	GFP	GFFPc	gLFMP	gGFMP
10 %	8	30	16 (13)	6	18	6.7
20 %	8	43	24 (20)	7	13	4.4
30 %	8	54	29 (25)	9	11	3.1
40–50 %	6	62	32 (27)	11	9–7	2.9–2.8
60–100 %	2	73	35 (29)	12	6–4	2.6–2.2

Table 1. The results for mutagenicity prediction task solved on 8 nodes

The results obtained by running on 8 nodes are shown in Table 1. The first column (Limit) shows the given value of the time limit. In the second column the number of nodes constrainedly interrupted when the locally frequent maximal

patterns were generated is given. The number of locally generated maximal frequent patterns is in the column headed LFMP. The GFP column shows how many LFMP patterns were globally frequent and maximal (the number in brackets). By generalizing patterns from LFMP GFFPc patterns were obtained. All patterns found by the serial and distributed algorithm were different except for one. This was due to the built-in discretization method. There were many common patterns that covered the same structure of molecules, but they differed in the interval boundaries generated for continuous values. The last two columns show the speed up obtained. Its value for mining globally frequent maximal patterns – all the steps of dRAP-Independent performed – is in the gGFMP column. When the last phase of the algorithm – generalization and specialization steps – is omitted the speed up is much higher (see column gLFMP).

By partitioning the data, almost twice as many globally frequent patterns (the GFP column in Table 1) were found as by the serial algorithm in 10 % of the serial execution time. This means that the speed up factor is 18 for mining locally frequent maximal patterns. The speed up decreases when all the steps are performed. Non-linear speed up was achieved because the value of S is less than 8. The reason is that the serial RAP is used for generalization and specialization of LFMP. Also, by relaxing the time limit the speed up decreases. This is a result of the pruning criteria used. Almost all nodes generate all the allowed patterns in the given time and are left waiting for nodes processing dense data. By interrupting execution if at least 6 nodes finish finding LFMP or by relaxing the pruning criteria the speed up can increase.

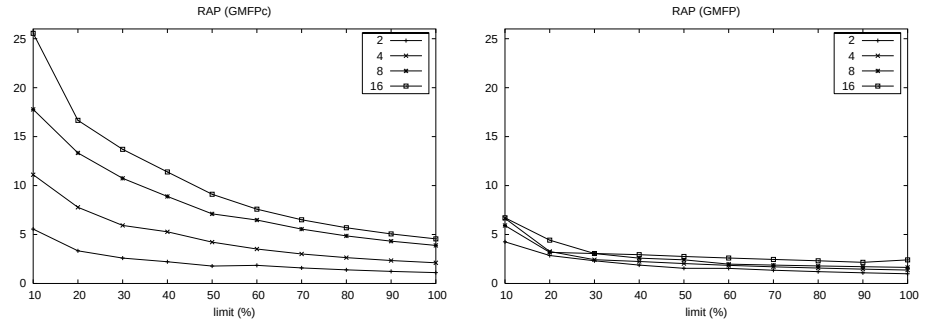

Fig. 3. The dependency of speed up S on the time limit

The dependency of the speed up on the time limit is shown in Figure 3. As we can see, the first phase (generating LFMP) speeds up linearly if the execution time is less than 30 % of the serial time. If less than 16 nodes were used linear speed up was obtained also for a 40 % time limit. The whole execution speeds up linearly only for two and four nodes (see the right graph in Figure 3). For large number of nodes it is lower than 8 because of serial processing of the second phase and the higher overhead of merging the patterns and their characteristics.

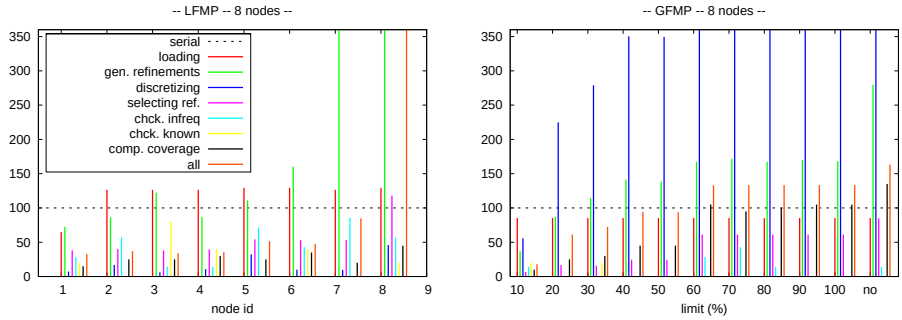

Fig. 4. The relative time of the single steps of RAP measured for mining LFMP (left) and GFMP (right) on 8 nodes. The value is compared to the time consumed by the serial execution (denoted as 100).

In Figure 4 the time consumed by each step of RAP algorithm is shown. Its value is relative to the time taken by the serial execution. The results given are for 8 nodes with no time limit. The results obtained by generating LFMPs are on the left side. We can see that both the most time-consuming steps (discretization and selecting refinement) are performed much faster on 7 of the 8 nodes. The explanation for the fact that the execution on the last node took so much longer is that a dense data set was assigned to this node. The second phase consumed the same or more time than the serial algorithm. This is as expected because all the data was analyzed on a single node and very long patterns (patterns containing more than 8 literals) were processed. As the execution time increased (because the limit was relaxed) a longer time for generating GFMPs was required. This is due to the large number of locally frequent candidate patterns.

To summarize these experiments, we can say that the first phase (generating locally frequent maximal patterns) speeds up linearly; but the performance of the system is not optimal because of the second phase, in which the patterns found are firstly generalized and then specialized.

6.2 Propositionalization with dRAP

As we remarked in Section 4.5, there are many tasks for which it is not important or convenient whether patterns are maximal or not. An example of such a task is propositionalization. dRAP-Independent allows the user to skip the second step (generalization and specialization of locally frequent maximal patterns); this results in better performance. The locally maximal frequent patterns were found and used directly for feature construction.

The experiments were performed on AMD Athlon^{TM} XP 2500+ computers with 756 MB of memory. The classifiers SMO (support vector machines [25]), J48 (an implementation of the Quinlan's decision trees algorithm C4.5 [37]), Naïve

Bayes (NB) [30], and instance-based learner (IB1) [30] from the Weka[10] system [41] were used with default parameters. An accuracy metric [21] is used for measuring quality of generated models. Distributed mining of frequent patterns was performed on four computers. A time limit was set to 90 and 15 minutes for the text correction task and to 15 minutes for information extraction.

Table 2 displays the speed up of the first phase (finding LFMP) achieved by using distributed processing on four nodes with a given maximal time (see the column called Limit). We can see that the algorithm speeds up linearly only for text correction task with $\mathcal{B}_2$ background knowledge. Its value is about 3.5 for three of four tasks and lower than three for two settings. On the other hand, the real speed up for the disambiguation of Czech is much higher than 3.4, because 400 examples and relaxed pruning were used for the distributed algorithm, while for the serial algorithm the data consisted of only 100 examples. For other tasks a small value of speed up was achieved because of the length of generated locally frequent patterns. For example, the serial algorithm created only four patterns containing more than four literals, while distributed algorithm processed ten such patterns. This was for text correction with $\mathcal{B}_1$ background knowledge, but the same holds for other data too.

		# of patterns		Time (s)		
Task	Limit (s)	RAP	dRAP	RAP	dRAP	Speed up
LLL05	900	10	22	901	901	2.2
Spell ($\mathcal{B}_1$)	900	2	8	938	907	2.1
	5400	18	40	5 438	3 608	3.4
Spell ($\mathcal{B}_2$)	900	2	16	938	907	4.1
	5400	15	55	5 438	5 438	3.7
Disamb[11]	*no*	42	72	234	119	3.4

Table 2. The speed up obtained by mining locally maximal frequent patterns on 4 nodes

The impact of distributed mining on the accuracy of propositional learners is shown in Table 3.

The accuracy increased for the text correction task and morphological disambiguation when distributed mining and SVM and J48 learners were used. The accuracy of classification by the Naïve Bayes classifier was higher for disambiguation, but lower for text correction. When the patterns found were used for solving information extraction task, the precision, recall, and F_1 increased when dRAP-Independent was used. We used not only the frequent patterns, but also the infrequent patterns generated by the RAP system, because the frequent patterns were not sufficient for learning a reasonable model. This resulted in an increase of all three measures. For more details see [35].

[10] `http://www.cs.waikato.ac.nz/~ml/weka/`

[11] We used 400 examples instead of 100 for distributed mining.

	SVM		J48		NB	
Task	RAP	dRAP	RAP	dRAP	RAP	dRAP
Spell ($\mathcal{B}_1$)	75.0%	**79.0%**	75.6%	**80.7%**	**74.0%**	72.6%
Spell ($\mathcal{B}_2$)	80.2%	**80.4%**	80.2%	**80.9%**	**76.8%**	73.5%
Disamb	76.0%	**83.5%**	73.0%	**73.8%**	77.5%	**83.0%**

Table 3. Accuracy obtained by propositionalization with the serial (RAP) and distributed (dRAP) mining

7 CONCLUSION

We described dRAP-Independent, a system for distributed mining of first-order frequent patterns in horizontally partitioned data. The algorithm adopts the ideas of the well-known propositional algorithm Partition. The main advantages of dRAP-Independent are its minimal communication overhead and the possibility to utilize any system for mining locally frequent patterns. The system has been shown to be very useful for solving several data mining tasks, e.g., information extraction from biological texts, context-sensitive text correction or morphological disambiguation.

We analyzed the performance of the algorithm in two ways – the speed up of the generation phase was measured and the usefulness of the generated patterns was studied. The experimental results showed that by distributed mining a large number of patterns can be found in the same overall time. We showed that the new patterns improve performance (precision, recall and accuracy) of propositional learners when used as new features.

We plan to integrate a message-passing interface for solving problems with the last two steps of dRAP-Independent. We would like to implement another parallel strategy for mining frequent patterns, e.g., those which distribute both data and hypothesis space. Because the strategy used for splitting the data plays a crucial role, we plan to pay special attention to adapting Cheung' algorithm for relational data and to desining new strategies for partitioning the data.

Acknowledgement

We are grateful to Patrick Hanks and anonymous referees for their comments. This work has been partially supported by the Faculty of Informatics, Masaryk University and by the Grant Agency of the Czech Republic under the Grant No. MSM0021622418 Dynamic Geo-visualization in Crisis Management.

REFERENCES

[1] Agrawal, R.—Imielinski, T.—Swami, A. N.: Mining Association Rules Between Sets of Items in Large Databases. In Proceedings of the 1993 ACM SIGMOD Inter-

national Conference on Management of Data, Washington, D. C., May 26–28, 1993, P. Buneman and S. Jajodia, Eds., ACM Press, pp. 207–216.

[2] Agrawal, R.—Shafer, J. C.: Parallel Mining of Association Rules. IEEE Transactions on Knowledge and Data Engineering, Vol. 8, 1996, No. 6, pp. 962–969.

[3] Agrawal, R.—Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In VLDB '94, Proceedings of 20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago de Chile, Chile, 1994, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds., Morgan Kaufmann, pp. 487–499.

[4] Blaťák, J.: First-Order Frequent Patterns in Text Mining. In Proceedings of the 12th Portuguese Conference on Artificial Intelligence, EPIA '05, (December 2005), Institute of Electrical and Electronics Engineers, Inc., pp. 344–350.

[5] Blaťák, J.—Popelínský, L.: Feature Construction with RAP. In Proceedings of the Work-in-Progress Track at the 13th International Conference on Inductive Logic Programming, (September 29–October 1, 2003), T. Horváth and A. Yamamota, Eds., University of Szeged, pp. 1–11.

[6] Blaťák, J.—Popelínský, L.: Mining First-Order Maximal Frequent Patterns. Neural Network World 5, 2004, pp. 381–390.

[7] Blockeel, H.—Dehaspe, L.—Demoen, B.—Janssens, G.—Ramon, J.—Vandecasteele, H.: Executing Query Packs in ILP. Lecture Notes in Computer Science, Vol. 1866, 2000.

[8] Blockeel, H.—Raedt, L. D.—Jacobs, N.—Demoen, B.: Scaling Up Inductive Logic Programming by Learning From Interpretations. Data Mining and Knowledge Discovery, Vol. 3, 1999, No. 1, pp. 59–93.

[9] Carlson, A. J.—Rosen, J.—Roth, D.: Scaling Up Context-Sensitive Text Correction. In Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence Conference, 2001, AAAI Press, pp. 45–50.

[10] Cheung, D. W.-L.—Lee, S. D.—Xiao, Y.: Effect of Data Skewness and Workload Balance in Parallel Data Mining. IEEE Transactions on Knowledge and Data Engineering, Vol. 14, 2002, No. 3, pp. 498–514.

[11] Clare, A.—King, R. D.: Data Mining The Yeast Genome in a Lazy Functional Language. In Practical Aspects of Declarative Languages, 5th International Symposium, PADL 2003, New Orleans, LA, USA, January 13–14, 2003, Proceedings, V. Dahl and P. Wadler, Eds., Vol. 2562 of Lecture Notes in Computer Science, Springer, pp. 19–36.

[12] Clare, A.—Williams, H. E.—Lester, N.: Scalable Multi-Relational Association Mining. In ICDM, 2004, IEEE Computer Society, pp. 355–358.

[13] de Raedt, L.—Džeroski, S.: First-Order jk-Clausal Theories are Pac-Learnable. Artif. Intell., Vol. 70, 1994, No. 1–2, pp. 375–392.

[14] Dehaspe, L.—De Raedt, L.: Parallel Inductive Logic Programming. In Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases, 1995.

[15] Dehaspe, L.—Toivonen, H.: Discovery of Relational Association Rules. In Relational Data Mining, S. Džeroski and N. Lavrač, Eds. Springer-Verlag, 2001, pp. 189–212.

[16] Dehaspe, L.—Toivonen, H.—King, R. D.: Finding Frequent Substructures in Chemical Compounds. In 4th International Conference on Knowledge Discovery and Data Mining, 1998, R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, Eds., AAAI Press., pp. 30–36.

[17] Deshpande, M.—Karypis, G.: Using Conjunction of Attribute Values for Classification. In Proceedings of the eleventh International Conference on Information and Knowledge Management, 2002, ACM Press, pp. 356–364.

[18] Džeroski, S.—Lavrač, N. eds.: Relational Data Mining. Springer-Verlag, Berlin, September 2001.

[19] Fonseca, N. A.—Silva, F.—Camacho, R.: Strategies to Parallelize ILP Systems. In Proceedings of the 15th International Conference on Inductive Logic Programming (ILP 2005) (Bonn, Germany, August 2005), S. Kramer and B. Pfahringer, Eds., Vol. 3625 of Lecture Notes in Artificial Intelligence, Springer-Verlag, pp. 136–153.

[20] Fonseca, N. A.—Silva, F.—Costa, V. S.—Camacho, R.: A Pipelined Data-Parallel Algorithm for ILP. In Proceedings of 2005 IEEE International Conference on Cluster Computing, (Boston, Massachusetts, USA, September 2005), IEEE.

[21] Forman, G.: An Extensive Empirical Study of Feature Selection Metrics for Text Classification. Journal of Machine Learning Research, Vol. 3, 2003, pp. 1289–1305.

[22] Grama, A.—Gupta, A.—Karypis, G.—Kumar, V.: An Introduction to Parallel Computing: Design and Analysis of Algorithms. Second ed., Addison Wesley, January 2003.

[23] Han, E.-H.—Karypis, G.—Kumar, V.: Scalable Parallel Data Mining for Association Rules. In SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13–15, 1997, Tucson, Arizona, USA, 1997, J. Peckham, Ed., ACM Press, pp. 277–288.

[24] Joshi, M. V.—Han, E.-H.—Karypis, G.—Kumar, V.: Efficient Parallel Algorithms for Mining Associations. In Large-Scale Parallel Data Mining, 1999, pp. 83–126.

[25] Keerthi, S. S.—Shevade, S. K.—Bhattacharyya, C.—Murthy, K. R. K.: Improvements to Platt's SMO Algorithm for SVM Classifier Design. Neural Computation, Vol. 13, 2001, No. 3, pp. 637–649.

[26] Konstantopoulos, S. T.: A Data-Parallel Version of Aleph. In Parallel and Distributed Computing for Machine Learning. In conjunction with the 14th European Conference on Machine Learning (ECML'03) and 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03), Cavtat-Dubrovnik, Croatia, September 2003, Springer.

[27] Kramer, S.—Lavrač, N.—Flach, P.: Propositionalization Approaches to Relational Data Mining. In Relational Data Mining, S. Džeroski and N. Lavrač, Eds. Springer-Verlag, September 2001, pp. 262–291.

[28] Liu, B.—Hsu, W.—Ma, Y.: Integrating Classification and Association Rule Mining. In Knowledge Discovery and Data Mining, 1998, pp. 80–86.

[29] Mannila, H.—Toivonen, H.: An Algorithm for Finding All Interesting Sentences. In Proceedings of the 6th Internation Conference on Database Theory, 1996, pp. 215–229.

[30] Mitchell, T. M.: Machine Learning. McGraw Hill, 1997.

[31] Nienhuys-Cheng, S.-H.—de Wolf, R.: Foundations of Inductive Logic Programming. Springer-Verlag, 1997.

[32] Nijssen, S.—Kok, J. N.: Efficient Frequent Query Discovery in Farmer. In 7^{th} European Conference on Principles and Practice of Knowledge Discovery in Databases, 2003, N. Lavrac, D. Gamberger, H. Blockeel, and L. Todorovski, Eds., Vol. 2838 of Lecture Notes in Computer Science, Springer, pp. 350–362.

[33] Pala, K.—Rychlý, P.—Smrž, P.: DESAM – Annotated Corpus for Czech. In Proceedings of SOFSEM 97, 1997, Springer Verlag, pp. 523–530.

[34] Pasquier, N.—Bastide, Y.—Taouil, R.—Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules. In ICDT, 1999, C. Beeri and P. Buneman, Eds., Vol. 1540 of Lecture Notes in Computer Science, Springer, pp. 398–416.

[35] Popelínský, L.—Blaťák, J.: Learning Genic Interactions without Expert Domain Knowledge: Comparison of different ILP Algorithms. In Proceedings of the 4^{th} Learning Language in Logic Workshop (LLL 05), 2005, Fraunhofer Institute, pp. 21–30.

[36] Popelínský, L.—Pavelek, T.: Mining Lemma Disambiguation Rules from Czech Corpora. In PKDD, 1999, J. M. Zytkow and J. Rauch, Eds., Vol. 1704 of Lecture Notes in Computer Science, Springer, pp. 498–503.

[37] Quinlan, J. R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

[38] Savasere, A.—Omiecinski, E.—Navathe, S. B.: An Efficient Algorithm for Mining Association Rules in Large Databases. In VLDB, 1995, U. Dayal, P. M. D. Gray, and S. Nishio, Eds., Morgan Kaufmann, pp. 432–444.

[39] Srinivasan, A.—Muggleton, S.—King, R.—Sternberg, M.: Theories for Mutagenecity: A Study of First-Order and Feature Based Induciton. Tech. rep., PRG-TR-8-95 Oxford University Computing Laboratory, 1995.

[40] Terabe, M.—Washio, T.—Motoda, H.—Katai, O.—Sawaragi, T.: Attribute Generation Based on Association Rules. Knowledge and Information Systems, Vol. 4, 2002, No. 3, pp. 329–349.

[41] Witten, I. H.—Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. second ed., Morgan Kaufmann, June 2005.

[42] Zaki, M. J.: Generating Non-Redundant Association Rules. In KDD '00: Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2000, ACM Press, pp. 34–43.

Jan Blaťák is a Ph. D. student at Faculty of Informatics, Masaryk University, Brno, Czech Republic. He is concerned with Relational Data Mining (RDM), especially Inductive Logic Programing (ILP) and its use in text mining tasks. He is developing serial and parallel methods for finding long first-order frequent patterns and for utilizing found patterns in data preprocessing and classification tasks.

Luboš Popelínský is an Associated Professor at the Faculty of Informatics, Masaryk University. He is concerned with machine learning, data and text mining and logic.