2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)
July 6-11, 2014, Beijing, China

Parallel Mining of Fuzzy Association Rules on Dense Data Sets

Michal Burda, Viktor Pavliska, and Radek Valášek

***Abstract*— The aim of this paper is to present a scalable parallel algorithm for fuzzy association rules mining that is suitable for dense data sets. Unlike most of other approaches, we have based the algorithm on the Webb's OPUS search algorithm [1]. Having adopted the master/slave architecture, we propose a simple *recursion threshold* technique to allow load-balancing for high scalability.**

I. Introduction

SEARCHING FOR ASSOCIATION RULES [2], [3] is a broadly discussed, developed and accepted data mining technique. An association rule is commonly understood as an expression $X \to Y$, where antecedent X and consequent Y are conditions – the former usually in the form of elementary conjunction. Such rules are usually interpreted as implication "if X is satisfied, then Y is true very often, too". Two traditional measures of intensity of an association rule are often used, $support$ and $confidence$. A task of searching for association rules is the task of finding rules with their $support$ and $confidence$ above some user-defined thresholds.

The task of searching for association rules fits particularly well on binary or categorical data and many has been written on that topic [2], [3], [4], [5].

For association analysis on numeric data, a prior discretization is proposed by Srikant et al. [6]. Unfortunately, that may lead to danger of undiscovering important knowledge due to information loss caused by discretization.

The problem of information loss due to discretization is tightly connected to the fact that quantitative attribute is transformed to several categories given by *crisp* boundaries. A rational request to "soften" these boundaries leads us to the fuzzy sets theory. The use of fuzzy sets in connection with association rules has been motivated by many authors (see [7] for recent overview). By allowing for soft boundaries of discretization intervals, fuzzy sets can avoid certain undesirable threshold effects [8]. Fuzzy association rules are appealing also because of the use of vague linguistic terms such as "small", "very big" etc. [9], [10]. See also [11].

Association rules mining, especially of fuzzy rules where the amount of attributes multiplies by using different linguistic terms, may be a very computational time demanding task. Therefore, a substantial research effort is devoted to design of parallel algorithms.

Parallel algorithms for association rules focus mainly on processing large, but sparse categorical data sets. That data often do not fit into RAM, therefore reading of data from the disk is the most time consuming task. The approaches for parallelization are e.g. *count distribution* [12] where each processor computes counts of appearance of candidates in locally stored data set, which are then combined to obtain global results. The *data distribution algorithm* [12] partitions candidate itemsets among the processors. Also combinations of these approaches exist [13]. All these approaches are based on candidate-generation technique developed originally in the *Apriori algorithm* [3].

Other approaches are based on FP-growth mining algorithm [14], which avoids generating candidate sets. Hu and Yang-Li [15] propose FP-Forests.

In this paper, we present a novel implementation of fuzzy association rules mining algorithm, that is based on OPUS search [16], [1] that is suitable for dense data sets. We develop a parallel version of OPUS, implement it as a program suitable to run on a high-performance computing cluster with the Open MPI parallelization environment [17], and evaluate its scalability on real data.

The rest of this paper is organized as follows. First, some theoretical background is presented in section II. Next, we present the algorithms and implementation details in section III. A performance analysis is presented in section IV and some directions of future research are sketched in section V.

II. Theoretical Background

A. Fuzzy Sets and Fuzzy Association Rules

A fuzzy set F of an universe U is defined by a *membership function* that for each element $u \in U$ specifies the degree of membership of u in the fuzzy set. For us, the membership function is a mapping $U \to [0, 1]$. We will not distinguish between a fuzzy set and its membership function, that is, $F(u)$ denotes the degree of membership of the element u in the fuzzy set F. (Clearly ordinary sets are special cases of fuzzy sets such that their membership function maps U to $\{0, 1\}$.)

T-norm $\otimes$ is a generalized logical conjunction, i.e. a function $[0, 1] \times [0, 1] \to [0, 1]$ which is associative, commutative, monotone increasing (in both places) and which satisfies the boundary conditions $\alpha \otimes 0 = 0$ and $\alpha \otimes 1 = \alpha$ for each $\alpha \in [0, 1]$. Some well-known examples of t-norms are:

- minimum t-norm: $\otimes_{\min}(\alpha, \beta) = \min(\alpha, \beta)$;
- product t-norm: $\otimes_{\text{prod}}(\alpha, \beta) = \alpha\beta$;
- Łukasiewicz t-norm: $\otimes_{\text{Łuk}}(\alpha, \beta) = \max(0, \alpha + \beta - 1)$.

The authors are from the University of Ostrava, Institute for Research and Applications of Fuzzy Modeling, Centre of Excellence IT4Innovations, 30. dubna 22, 701 03 Ostrava, Czech Republic (Michal.Burda@osu.cz, Viktor.Pavliska@osu.cz, Radek.Valasek@osu.cz).

This work was supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033).

978-1-4799-2072-3/14/$31.00 ©2014 IEEE

TABLE I
LINGUISTIC HEDGES AND THEIR ABBREVIATIONS, SORTED BY EFFECT AND SPECIFICITY.

Narrowing effect	Widening effect
very (Ve)	more or less (ML)
significantly (Si)	roughly (Ro)
extremely (Ex)	quite roughly (QR)
–	very roughly (VR)

T-norms are used for defining the *intersection* of fuzzy sets F and G: $(F \cap G)(u) := F(u) \otimes G(u)$.

The *cardinality* of a fuzzy set F is defined as the sum of the membership degrees: $|F| := \sum_{\forall u \in U} F(u)$ [18].

Let $\mathcal{D}$ be a set of objects for which some properties are identified by the fuzzy subsets of $\mathcal{D}$ that we will call the *fuzzy attributes*; let there be n fuzzy attributes. For instance, if $\mathcal{D}$ was a set of people, then t could be a fuzzy subset of $\mathcal{D}$ of people that are "tall". I.e. for each $x \in \mathcal{D}$, $t(x)$ results in a membership degree of x being "tall". A fuzzy association rule is then a rule of the form $\mathcal{A} \to \mathcal{B}$, where $\mathcal{A}$ and $\mathcal{B}$ are sets of fuzzy attributes. For example:

$$\{leading\_position, middle\_age\} \to \{high\_income\};$$

i.e. "middle-aged people working on leading position have high income very often".

Let $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ be a set of m fuzzy attributes. A support of the set $\mathcal{A}$ is the cardinality of the intersection of all attributes $A_k \in \mathcal{A}$:

$$supp(\mathcal{A}) := \left| \bigcap_{k=1}^{m} A_k \right| = \sum_{\forall x \in \mathcal{D}} \Big(A_1(x) \otimes \ldots \otimes A_m(x) \Big). \quad (1)$$

Similarly, a support of a rule $\mathcal{A} \to \mathcal{B}$ is defined as a support of the set $\mathcal{A} \cup \mathcal{B}$:

$$supp(\mathcal{A} \to \mathcal{B}) := supp(\mathcal{A} \cup \mathcal{B}). \quad (2)$$

A confidence of the rule $\mathcal{A} \to \mathcal{B}$ is given by

$$conf(\mathcal{A} \to \mathcal{B}) := \frac{supp(\mathcal{A} \to \mathcal{B})}{supp(\mathcal{A})}. \quad (3)$$

Mining for fuzzy association rules means searching for all rules with support and confidence above some user-specified thresholds.

B. Evaluative Linguistic Expressions

Evaluative linguistic expressions [19] (or *evaluative expressions*, in short) are special expressions of natural language that are used for vague evaluation of intensity, in context of range of all reasonable values. Example of evaluative expression is: *very large*, *extremely hot*, *more or less correct* etc. Their importance and potential to be mathematically modelled has been pointed out by L. A. Zadeh, e.g. in [20].

Evaluative expressions keep the following structure:

$$\langle \text{linguistic hedge} \rangle \langle \text{atomic evaluative expression} \rangle. \quad (4)$$

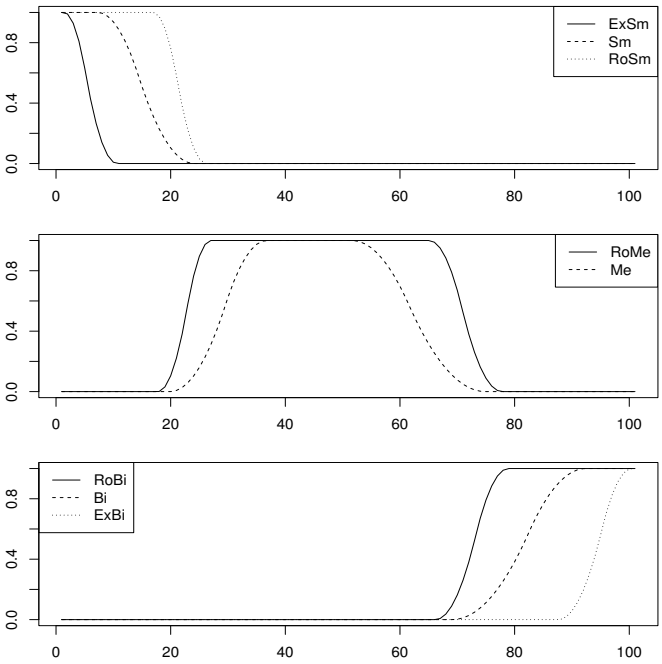

Fig. 1. Shapes of fuzzy sets of evaluative linguistic expressions for context $\langle 0, 42, 100 \rangle$.

Atomic evaluative expression comprises any of the adjectives *small* ("Sm"), *medium* ("Me"), or *big* ("Bi"). *Linguistic hedge* is an optional adverb that further determines the expression. It can have either narrowing effect, e.g. *very* ("Ve"), *extremely* ("Ex"), or widening effect such as *roughly* ("Ro"), *more or less* ("ML") etc. In our software, we use linguistic hedges listed in Table I.

Hedges with widening effect are applicable on all three atomic evaluative expressions (i.e. "Sm", "Me", "Bi"), whereas hedges with narrowing effect are typically used together with "Sm" and "Bi" only.

The notion of evaluative expression may be used to fuzzify numeric attributes. Given a numeric attribute A, one may generate 21 fuzzy attributes of the form

$$A \text{ is } \langle \text{evaluative expression} \rangle$$

by combining evaluative expressions with linguistic hedges from Table I as follows: ExSm, SiSm, VeSm, Sm, MLSm, RoSm, QRSm, VRSm, Me, MLMe, RoMe, QRMe, VRMe, ExBi, SiBi, VeBi, Bi, MLBi, RoBi, QRBi and VRBi.

For fuzzification, a numeric attribute's *context* denotes what value is small, medium, or big, respectively. *Context* is a triplet $\langle v_L, v_M, v_R \rangle \in \mathbb{R}^3$, $v_L < v_M < v_R$. It may be set manually or automatically by choosing A's minimal, middle, and maximal value of A, respectively. Linguistic hedges define the shape of the fuzzy sets around these values. Details on fuzzification are out of the scope of this paper. Please see [19] for more information. See also Fig. 1 for an example of fuzzy sets within the context $\langle 0, 42, 100 \rangle$.

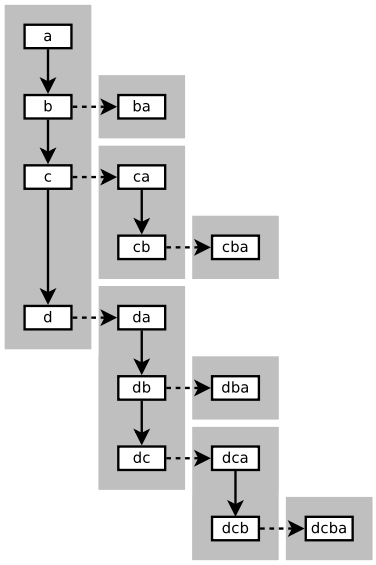

Fig. 2. Tree of antecedent combinations traversed by the OPUS search algorithm [16]. Each box represents a left-hand side of a rule. Hence, all available right-hand sides have to be added to create a well-formed association rule. Gray boxes represent *tasks*, i.e. a single loop through all available predicates with fixed *prefix*. Dashed arrows stand for recursive calls (for non-parallel version), or task enqueuing on master (for parallel version), respectively.

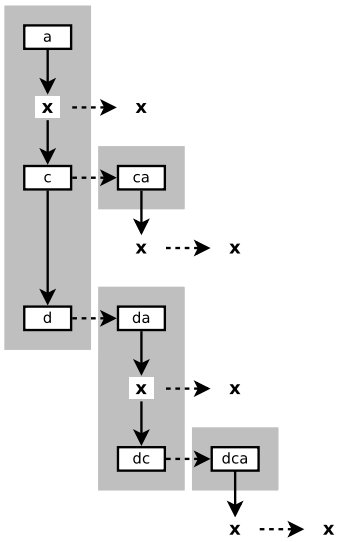

Fig. 3. Pruning a node (here: "b") causes not-evaluating all descendand nodes of the current *task* (gray box) that contain pruned value (i.e. pruned are also: "ba", "cb", "cba", "db", "dba", "dcb" and "dcba").

III. Association Rules Mining Algorithm

Searching for fuzzy linguistic association rules requires a dataset in which rows represent objects and columns features. For fuzzy rules, the features are in the form of fuzzy sets, i.e. for each object, a memberhip degree to each fuzzy set is defined.

Our parallel algorithm for fuzzy association rules mining is inspirred by Webb's OPUS algorithm [16], [1], that was initially developed as non-parallel search algorithm for rules with crisp antecedents. The details of OPUS adoption to fuzzy rules and parallelization is described in this section.

A. The OPUS Search

Webb's OPUS algorithm [16], [1] is based on recursive depth–first *branch and bound* search allowing efficient search space pruning with moderate space complexity, suitable to process dense data. We assume all data fit into RAM, hence no repeated disk file read is necessary.

The algorithm traverses a tree of combinations that will form an antecedent of a rule. Antecedents are recursively lengthened to obtain more complex rules, until some ending condition is reached (e.g. running below minimum support threshold). Pruning an antecedent combination causes not-evaluating all antecedents that are descendants of the pruned one – please see Algorithm 1 and compare Fig. 2 with Fig. 3. Hence, the algorithm is preferably suitable for classification rules.

A sequential (non-parallel) algorithm starts with the call

$$\text{SEARCH}(allLhs, \emptyset, allRhs, \text{SEARCH}),$$

where $allLhs$ (resp. $allRhs$) is a set of fuzzy attributes that may appear in the antecedent (resp. consequent) part of a rule.

The SEARCH procedure works with a set of attributes that may appear in antecedent ($avail$) and consequent ($conseq$). It is possible that $avail \cap conseq \neq \emptyset$.

The algorithm iterates through all fuzzy attributes $p \in avail$, creates an antecedent part of a rule (lhs) from p, and if rules with that antecedent are not prunable, it iterates through all attributes in $conseq$ to get consequent parts (rhs) of the rules. Iteration through all attributes in $avail$ is called a *search task* (or simply a *task*). Tasks are depicted as gray boxes in Fig. 2.

In its basic form, pruning is based on well-known *Apriori* condition: if some set $\mathcal{A}$ of attributes has support lower than user-defined threshold s, i.e. $supp(\mathcal{A}) < s$, then for any attribute a, $supp(\mathcal{A} \cup \{a\}) < s$ too. Hence, if we are searching for rules with support above s and $supp(\mathcal{A}) < s$ then we do not need to traverse through supersets of $\mathcal{A}$ at all. Therefore, antecedent $\mathcal{A}$ and all its descendants can be pruned.

All attributes of $avail$, that are not pruned-out, are used as a $prefix$ for antecedents that are traversed in a recursive call. That attributes are then also stored in $nextAvail$, which in the recursive call become the set of available antecedent attributes. Recursive calls are depicted by dashed arrows in Fig. 2.

There occurs $stepFunc$ parameter in the SEARCH procedure in Algorithm 1. It is a function that is used for steps into subtree of search tasks. For non-parallel version of the algorithm, we simply put $stepFunc = \text{SEARCH}$.

Algorithm 1 Non-parallel (recursive) version of the search algorithm

```
1: procedure SEARCH(avail, prefix, conseq, stepFunc)
2:     nextAvail ← ∅
3:     for p ∈ avail do
4:         lhs ← prefix ∪ {p}
5:         if lhs is not prunable then
6:             nextConseq ← ∅
7:             for rhs ∈ conseq do
8:                 if lhs → rhs is not prunable then
9:                     report new rule lhs → rhs
10:                    nextConseq ← nextConseq ∪ {rhs}
11:                end if
12:            end for
13:            if nextConseq ≠ ∅ then
14:                stepFunc(nextAvail, lhs, nextConseq, stepFunc)
15:                nextAvail ← nextAvail ∪ {p}
16:            end if
17:        end if
18:    end for
19: end procedure
```

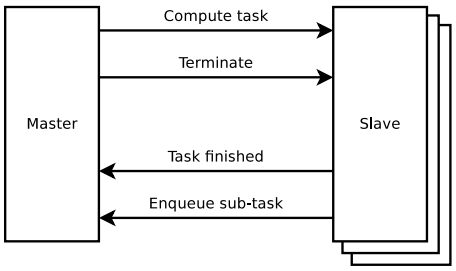

Fig. 4. Master sends tasks to be computed by slaves via the *"Compute task"* message. On the other hand, slaves send sub-tasks back to master (*"Enqueue sub-task"*) to be enqueued and then re-sent to other slaves. Also slaves inform master of their availability for next tasks by sending *"Task finished"* message. Master uses the *"Terminate"* message to inform slaves of the end of the algorithm.

B. Parallelized Version of the OPUS Search

Parallelized search for association rules is driven by the master/slave architecture and message-passing between them, see Fig. 4. Master sends tasks to slaves, slaves compute them and send sub-tasks back to master where the tasks are stored in a queue and later re-sent back to slaves to be computed.

Task is a triplet $\langle avail, prefix, conseq \rangle$ where $avail$ is a set of fuzzy attributes for antecedent part of the rules, $conseq$ is a set of attributes for consequent part of the rules, and $prefix$ is a set of antecedent attributes that will be common for all rules generated within the task.

Message msg is a record that is sent from master to slave or vice versa. Parts of a message will be addressed using a "dot notation" in the algorithm: e.g. $msg.type$ is for the type of the message, $msg.sender$ is for the ID of the sender of the message etc.

In detail, master proceeds as follows (see also Algorithm 2). Firstly, a $taskQueue$ is created with a root task $\langle allLhs, \emptyset, allRhs \rangle$ in it. Also a $slaveQueue$ is initialized and all k available slaves are started. The $slaveQueue$ contains slaves that are actually not working on any task.

The master's main loop runs until the $taskQueue$ is empty and all slaves are not computing. Inside of the main loop,

Algorithm 2 Parallel (1 master, many slaves) version of the search algorithm

```
1: procedure MASTER(allLhs, allRhs)
2:     taskQueue ← {⟨allLhs, ∅, allRhs⟩}
3:     slaveQueue ← {s_1, s_2, ..., s_k}
4:     start SLAVE on each slave s_1, s_2, ..., s_k
5:     while taskQueue ≠ ∅ or |slaveQueue| < k do
6:         while taskQueue ≠ ∅ and slaveQueue ≠ ∅ do
7:             s ← pop slave from slaveQueue
8:             t ← pop task from taskQueue
9:             msg.type ← "compute task"
10:            msg.task ← t
11:            send msg to slave s
12:        end while
13:        msg ← wait for message from some slave
14:        if msg.type = "enqueue sub-task" then
15:            push msg.task to taskQueue
16:        else if msg.type = "task finished" then
17:            push msg.sender to slaveQueue
18:        end if
19:    end while
20:    msg.type ← "terminate"
21:    send msg to each slave in slaveQueue
22: end procedure

23: procedure SLAVE
24:    while true do
25:        msg ← wait for message from master
26:        if msg.type = "terminate" then
27:            return
28:        else if msg.type = "compute task" then
29:            ⟨avail, prefix, conseq⟩ ← msg.task
30:            SEARCH(avail, prefix, conseq, STEPINTO)
31:            msg.type ← "task finished"
32:            send msg to master
33:        end if
34:    end while
35: end procedure

36: procedure STEPINTO((avail, prefix, conseq, stepFunc))
37:    if |avail| < recursionThreshold then
38:        SEARCH(avail, prefix, conseq, SEARCH)
39:    else
40:        msg.type ← "enqueue sub-task"
41:        msg.task ← ⟨avail, prefix, conseq⟩
42:        send msg to master
43:    end if
44: end procedure
```

as much tasks in the $taskQueue$ as possible are sent to unoccupied slaves inside of the *"compute task"* messages. After that, master waits for a message from some slave, which could be:

1) *"enqueue sub-task"* – slave sent a task, which is put into $taskQueue$ (and possibly re-sent to some unoccupied slave in the beginning of next main loop's iteration);
2) *"task finished"* – a slave informs master that it has stopped working, so it is free to be assigned another task; hence master puts that slave into $slaveQueue$ of non-working slaves.

After the end of the main loop, master sends a *"terminate"* message to all slaves to inform them of the end of the program.

Slaves wait for a message from master in an infinite loop. If they receive the *"terminate"* message, they exit. If they

receive the *"compute task"* message, they start the SEARCH procedure as in non-parallel variant, with $stepFunc$ set to STEPINTO.

As described above, $stepFunc$ is called when it is needed to compute sub-task, i.e. a task with current antecedent (lhs) being the prefix to longer antecedents. Non-parallel version of the algorithm simply calls SEARCH to do that (i.e. $stepFunc =$ SEARCH).

Parallelized approach either calls recursive SEARCH or sends the task encapsulated into the *"enqueue sub-task"* message to the master. Decision between those variants is made inside of the STEPINTO procedure.

Why to alternate between both recursion and sending? As indicaded by our experiments described later, a sufficiently large amount of slaves processing data with relatively small number of rows may cause the master to be overwhelmed by the amount of messages, which destroys scalability of the whole solution. Therefore, an optimal load of work has to be given to the slaves in order to not to communicate with master too often.

In our approach, *simple* sub-tasks are not sent to the master, but directly computed. A task is *simple*, if the potential search sub-tree is not so large, which is the case for $avail$ containing relatively small number of attributes. We use a constant $recursionThreshold$ to decide: if $|avail| < recursionThreshold$ then a sub-task is computed recursively, else it is sent to the master.

Another approach may use some other criterion of load-balancing, or somehow dynamically control the $recursionThreshold$. We left these advanced techniques for the future.

IV. PERFORMANCE ANALYSIS

The parallel algorithm was tested on high performance computer *Anselm* of the National Supercomputing Center IT4Innovations (Czech Republic). Each node of the cluster contains 2x8 CPU cores, 64 GB RAM, nodes are connected with InfiniBand QDR network and communicate with the OpenMPI library. Tests were performed on 2 to 64 MPI nodes (1 master with 1 to 63 slaves).

First, an optimal value for the $recursionThreshold$ was estimated. With $recursionThreshold$, amount of work may be balanced between slaves and task queue that is managed by master. The higher the $recursionThreshold$ is, the more search tree branches are computed on a slave within a single task. Also the less communication is needed with the master, because instead of enqueuing, many tasks are computed on slave, recursively. On the other hand, the higher the $recursionThreshold$ is, the less tasks appear in the task queue on the master, and therefore many slave nodes may be unoccupied for a long time (especially during the start and finish period). Hence an optimal value should balance the utilization of slave nodes together with the overheads on the master caused by management of the task queue.

Figure 5 shows overall algorithm run time on sample data for different values of the $recursionThreshold$. For

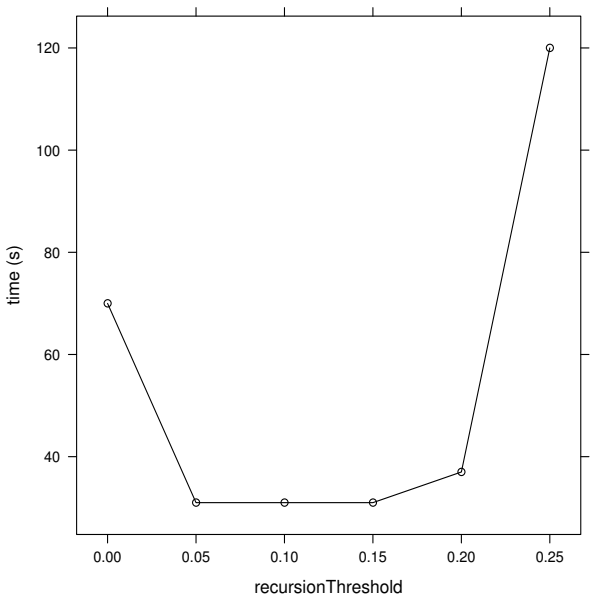

Fig. 5. Change of parallel algorithm's run time in dependence of the value of $recursionThreshold$.

$recursionThreshold = 0.1$, we have reached the shortest time. Therefore, for the analysis of scalability, the $recursionThreshold$ was set to that value (with single exception described below).

Scalability analysis was performed on some data sets from the UCI Machine Learning Repository [21]. For our interest in fuzzy rules, we have selected data sets with numerical attributes only. See Table II for details on data sets being used. First column of that table is for data set name. Next is number of rows and columns of the original data following with the number of fuzzy attributes generated from them and used for antecedents and consequents. Last four columns of Table II are settings and results of rule mining: minimum support threshold, maximum allowed length of the rule (i.e. number of fuzzy attributes in antecedent), $recursionThreshold$ and average scale ratio r_d defined below.

Each numeric attribute of the original dataset was transformed into 7 fuzzy attributes by using linguistic expressions, as described in Section II-B. One numeric attribute (i.e. 7 fuzzy attributes) was set as a consequent, the rest of the attributes were used in the antecedent part of the rules.

Figure 6 shows a graph with logarithmic axes, where an execution time of the parallel algorithm is apparent if multiplying the number of slave nodes. See also Table II for average scale ratio r_d of a data set d that is computed as follows:

$$r_d = \frac{1}{|n|-1} \sum_{i=1}^{|n|-1} \frac{t_d(n_i)}{t_d(n_{i+1})},$$

where $n = \{3, 5, 9, 17, 33, 64\}$ is a set of numbers of nodes that were used, and $t_d(n_i)$ is a run time of the algorithm on data d and n_i nodes. (The arrangement of nodes was: 1 master and 2, 4, 8, ... slaves.)

TABLE II
DATA SETS USED FOR PERFORMANCE TESTING.

Data Set d	Original		Generated Fuzzy Attributes		Minimum	Maximum	Recursion	Avg. Scale
	Rows	Columns	Antecedents	Consequents	Support	Length	Threshold	Ratio r_d
WDBC	569	32	217	7	0.010	5	0.10	1.9658
Wine	178	13	84	7	0.001	8	0.10	1.7024
Communities	1994	128	889	7	0.050	3	0.10	1.8121
KEGG	53414	24	161	7	0.050	3	0.10	1.8423
YearPredictionMSD	515345	90	623	7	0.050	3	0.10	1.3964
YearPredictionMSD.2	515345	90	623	7	0.050	3	0.05	1.7856

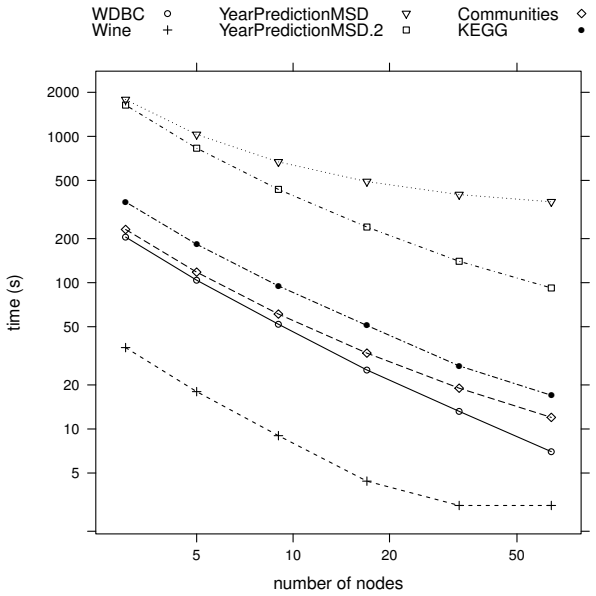

Fig. 6. Scalability of the parallel algorithm. Axes of the graph are logarithmic. Note that running time nearly halves if doubling the number of nodes being used, which indicates excellent scalability.

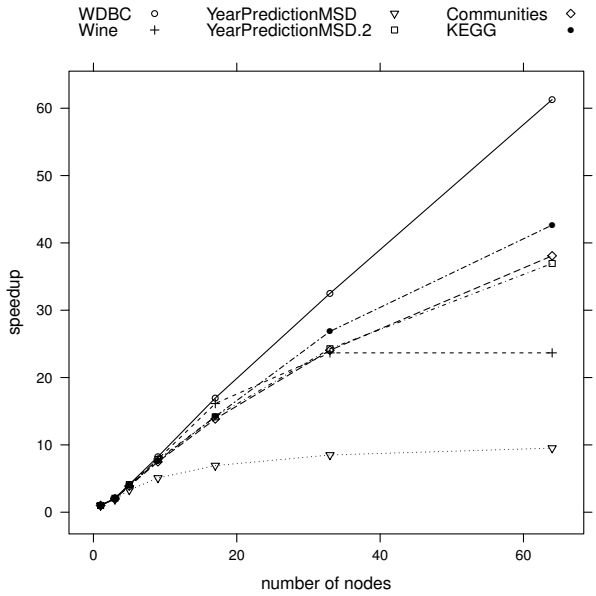

Fig. 7. Speedup of a parallel algorithm, i.e. a ratio of run time of a sequential algorithm to run time of the parallel version running on given number of nodes. Linear speedup (i.e. speedup equal to the number of nodes) is "ideal".

A scale ratio $r_d = 2$ means ideal scaling: doubling the number of nodes causes halving the run time. As can be seen in Table II, excellent scale ratio was obtained for WDBC, KEGG, and Communities data sets. Wine data set scaled very well, too.

Figure 7 shows the dependency of speedup on the number of nodes involved in computation. Speedup S_n is defined as

$$S_n = \frac{T_1}{T_n},$$

where n is the number of nodes, T_1 is the execution time of the sequential algorithm, and T_n is the execution time of the parallel algorithm with n nodes. Ideal speedup is $S_n = n$.

Figure 8 shows efficiency of our parallel algorithm on tested datasets. Efficiency E_n is defined as

$$E_n = \frac{S_n}{n} = \frac{T_1}{nT_n}.$$

Efficiency measures how well are the nodes utilized in solving the problem, in contrast to the execution time spent by communication and waiting on locks for exclusive access to shared resources. Ideal efficiency is $E_n = 1$.

As can be seen, YearPredictionMSD scaled rather poorly for $recursionThreshold = 0.1$. Therefore, another run was performed with $recursionThreshold = 0.05$ which ended with very good scale ratio of 1.7856. This experiment indicates the need to somehow dynamically estimate or adjust the optimal $recursionThreshold$, which we will address in the future research.

Rather poor efficiency was observed on 64 nodes for the Wine dataset. However, Wine dataset is very small, having very few columns and rows. On 64 nodes, the computation lasted no more than 3 seconds. It is clear that in contrast to the association rules mining process itself, a large portion of execution time was spent by initializing the parallel environment, starting the slave processes, and so on. Accordingly to Amdahl's law, we have reached a maximum speed-up in this case, and further addition of nodes could not affect the total execution time significantly.

V. CONCLUSION

In this paper, a novel technique was presented for parallel search of fuzzy association rules. Based on the OPUS al-

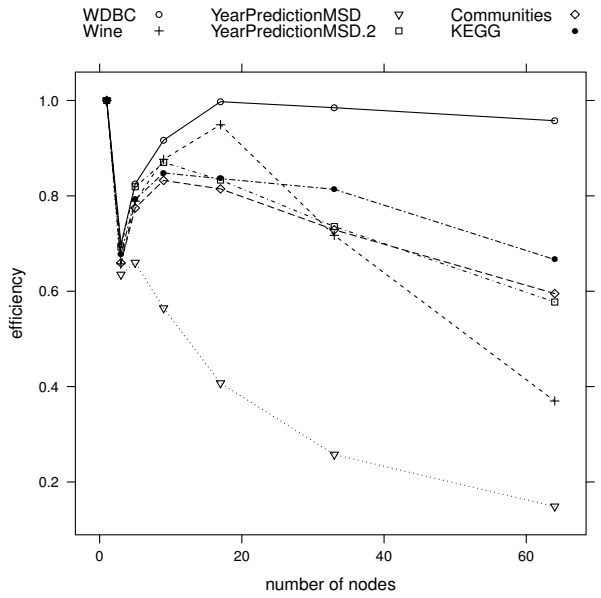

Fig. 8. Efficiency measures how well the nodes utilize the execution time for work on the problem, compared to the time spent on synchronization and communication. Ideal efficiency is 1.

gorithm [16], [1], we have designed a solution with parallel master/slave architecture, implemented it with the use of the OpenMPI framework and tested on the high performance computer.

Performance analysis shows very good scalability of the algorithm. We have indicated that load-balancing may be achieved by setting optimally the $recursionThreshold$, whose value was only rough guess in the presented performance analysis. Future work will therefore address elaboration of better load balancing by dynamically changing the $recursionThreshold$ based e.g. on the $taskQueue$ size, amount of available slaves, average time to compute the task etc.

REFERENCES

[1] G. I. Webb, "Opus: An efficient admissible algorithm for unordered search," *Journal of Artificial intelligence Research*, vol. 3, pp. 431–465, 1995.

[2] P. Hájek, I. Havel, and M. Chytil, "The GUHA method of automatic hypotheses determination," in *Computing 1*, 1966, pp. 293–308.

[3] R. Agrawal, T. Imielinski, and A. Swami, "Mining associations between sets of items in massive databases," in *ACM SIGMOD 1993 Int. Conference on Management of Data*, Washington D.C., 1993, pp. 207–216.

[4] A. Berrado and G. C. Runger, "Using metarules to organize and group discovered association rules," *Data Min. Knowl. Discov.*, vol. 14, no. 3, pp. 409–431, 2007.

[5] G. I. Webb, "Discovering significant patterns," *Mach. Learn.*, vol. 68, no. 1, pp. 1–33, 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1265347

[6] R. Srikant and R. Agrawal, "Mining quantitative association rules in large relational tables," *SIGMOD Rec.*, vol. 25, no. 2, pp. 1–12, 1996.

[7] H. Kalia, S. Dehuri, and A. Ghosh, "A survey on fuzzy association rule mining," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 9, no. 1, pp. 1–27, 2013.

[8] T. Sudkamp, "Examples, counterexamples, and measuring fuzzy associations," *Fuzzy Sets and Systems*, vol. 149, no. 1, pp. 57–71, 2005.

[9] K. C. Chan and W.-H. Au, "Mining fuzzy association rules," 1997.

[10] V. Novák, I. Perfilieva, A. Dvořák, G. Chen, Q. Wei, and P. Yan, "Mining pure linguistic associations from numerical data," *Int. J. Approx. Reasoning*, vol. 48, no. 1, pp. 4–22, Apr. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.ijar.2007.06.005

[11] M. Burda, "Fast evaluation of t-norms for fuzzy association rules mining," in *14th IEEE International Symposium on Computational Intelligence and Informatics (CINTI 2013)*. Budapest: IEEE, 2013, pp. 465–470.

[12] R. Agrawal and J. C. Shafer, "Parallel mining of association rules," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, pp. 962–969, 1996.

[13] E.-H. Han, G. Karypis, and V. Kumar, "Scalable parallel data mining for association rules," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 377–352, 2000.

[14] O. R. Zaane, M. El-hajj, and P. Lu, "Fast parallel association rule mining without candidacy generation," in *In ICDM*, 2001, pp. 665–668.

[15] J. Hu and X. Yang-Li, "A fast parallel association rules mining algorithm based on fp-forest." in *ISNN (2)*, ser. Lecture Notes in Computer Science, F. Sun, J. Zhang, Y. Tan, J. Cao, and W. Y. 0001, Eds., vol. 5264. Springer, 2008, pp. 40–49.

[16] G. I. Webb, "Discovering associations with numeric variables," in *Knowledge Discovery and Data Mining*, 2001, pp. 383–388. [Online]. Available: http://citeseer.ist.psu.edu/rey01discovering.html

[17] E. Gabriel et al., "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.

[18] V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*. Massachusetts, USA: Kluwer, 1999.

[19] V. Novák, "A comprehensive theory of trichotomous evaluative linguistic expressions." *Fuzzy Sets and Systems*, vol. 159, no. 22, pp. 2939–2969, 2008.

[20] L. A. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning - I, II, III," *Inf. Sci.*, vol. 8-9, pp. 199–249, 301–357, 43–80, 1975.

[21] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml