Backward Chaining Inference as a Database Stored Procedure — the Experiments on Real-world Knowledge Bases

Roman Simiński and Tomasz Xięski
Institute of Computer Science
University of Silesia
Będzińska 39, 41-200 Sosnowiec, Poland
Email: {roman.siminski|tomasz.xieski}@us.edu.pl

***Abstract*—In this work two approaches of backward chaining inference implementation were compared. The first approach uses a classical, goal driven inference running on the client device — the algorithm implemented within the `KBExpertLib` library was used. Inference was performed on a rule base buffered in memory structures. The second approach involves implementing inference as a stored procedure, run in the environment of the database server — an original, previously not published algorithm was introduced. Experiments were conducted on real-world knowledge bases with a relatively large number of rules. Experiments were prepared so that one could evaluate the pessimistic complexity of the inference algorithm.**

***Index Terms*—Expert systems, knowledge bases, backward chaining inference, databases**

I. Introduction

Rules are among the most popular forms of representing knowledge in the field of intelligent information systems, regardless of the development of different knowledge representations. Forward and backward chaining inference algorithms are also popular in the real-world applications [1]. The number of applications which utilise rule bases and methods of inference grows, but unfortunately the number of tools for building knowledge-based systems increase much more slowly [2]. The well-known systems like JESS [3], CLIPS [4], DROOLS [5] or EXSYS [6] are usually described as the tools for implementing domain knowledge based systems. What is more, commercial expert system development tools have been extended to offer web-based development capabilities.

A selected part of our research focused on the development of new methods and tools for building knowledge-based systems is presented in this work. In our previous work [7] we introduced the `KBExplorer` system — a WWW application which allows the user to create, edit and share rule-based knowledge bases. We also introduced (in a separate paper) the inference engine, which is provided by the `KBExpertLib` — a software library which allows programmers to implement domain knowledge-based systems using the Java programming language [8]. Next part of the realized project is the `KBExploratorDesktop` [9] — a desktop application which allows to analyse knowledge bases created by the `KBExplorer`. `KBExploratorDesktop` internally uses the `KBExpertLib` library and is implemented as an standard JavaFX GUI program. The prototype version of `KBExplorer` and the demo version of `KBExploratorDesktop` are available on-line at http://kbexplorer.ii.us.edu.pl. The fig. 1 presents the main software modules of the proposed distributed expert system shell. The system is still under development — enhanced versions of the software are in the test phase.

The motives behind this article are research and implementation works concerning on application of proposed tools for a weak hardware configuration — like in mobile and embedded devices or obsolete (but still frequently used) computers. The `KBExpertLib` library provides inference algorithms implemented in Java and working on the internal device's resources. The inference performed on such devices may be ineffective from the user's point of view, and for the large rule bases may be totally impossible. We propose a different approach: a client device sends the initial inference information to the server side over the Internet and receives inference results. The comparison of time efficiency of these two approaches is the main goal of the article. This article presents new (and not published before) implementation issues, focused on the backward chaining inference. Presented considerations are the continuation of the previous, forward inference dedicated studies [22]. This article is a part of a wider project involving both research and implementation works — in this work we analyse two different approaches mentioned earlier.

The first approach uses a classical, goal driven inference running on the client device. An algorithm implemented within the `KBExpertLib` is used. The content of a particular knowledge base is retrieved from a local XML file or a relational database and is stored in the device's RAM — fig. 2 illustrates the discussed solution. Inference is performed on a rule base (buffered in memory structures) and the effectiveness of this process depends on the hardware configuration of the local machine and will vary due to the size of the knowledge base. This type of inference uses local resources (memory, processor) of the client's device and may consume a significant amount of energy (possibly supplied with a battery).

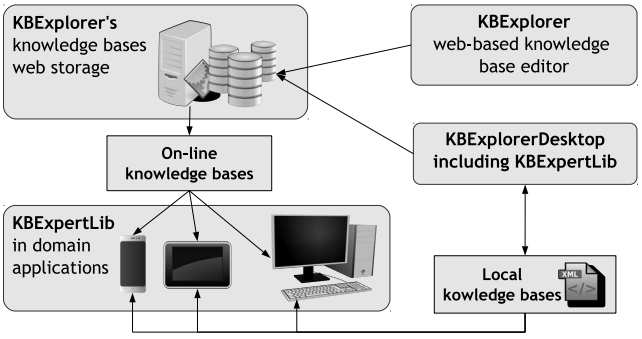

Fig. 1. Main components of the distributed expert systems shell

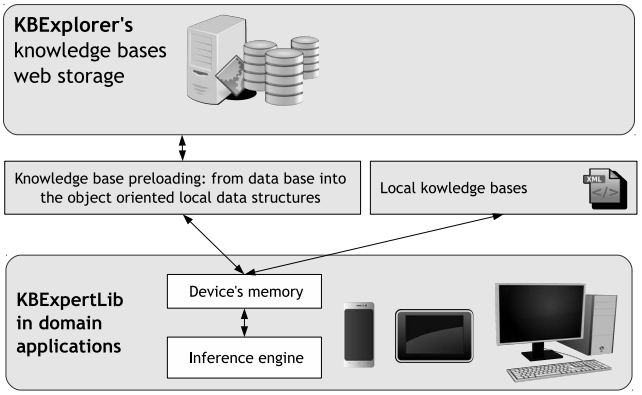

Fig. 2. First approach: inference as a local process

The second approach assumes that the inference process is being realized fully on the server side. A dedicated PHP implementation was previously analysed [10]. In this work utilization of stored procedures (within the database server) is considered. A client device uses Rest API services, sends the goal and facts to the server and receives information about the goal confirmation (and optional details) — fig. 3 illustrates the presented approach. The utilization of a server side implementation minimizes the network traffic, as only a single request is necessary. The usage of database server's stored procedures ensures independence from the used programming tools — only a connection to the database server and a simple API is required. The main research goal of this paper was the experimental evaluation of the backward chaining inference algorithm implementation as a stored procedure and a comparison (of such implementation) with inference performed on preloaded knowledge bases (on local devices). Considered approaches are quite different from the implementation point of view.

The organization of this paper is as follows. The next section presents background information – the formal knowledge base model, classical backward chaining inference algorithm and related works. Section 3 outlines the proposed methods and tools — the backward chaining inference algorithm as a stored procedure is introduced. Section 4 presents the experiments and their results. Finally, the conclusions section summarizes the paper.

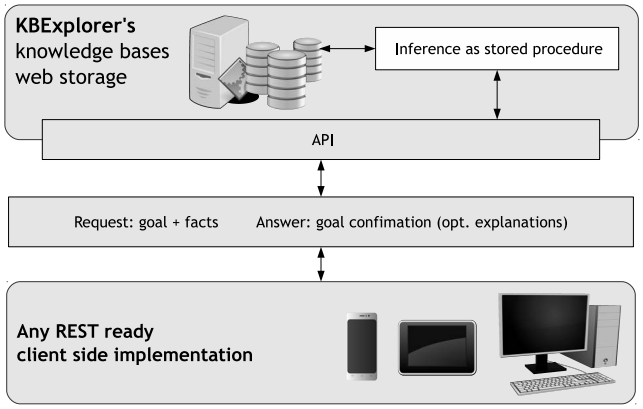

Fig. 3. Second approach: inference as a remote process

II. Background information and related works

This section presents the formal description of a knowledge base, the backward chaining inference algorithm, a brief review of software tools related with this work and a short description of our own software implementation.

A. Knowledge base

The following formal description of a knowledge base is assumed in this work: a knowledge base is a pair $\mathcal{KB} = (\mathcal{R}, \mathcal{F})$ where $\mathcal{R}$ is a non-empty finite set of rules and $\mathcal{F}$ is a finite set of facts. Furthermore, $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ and each rule $r \in \mathcal{R}$ will be represented in the form of Horn's clause: $r : p_1 \wedge p_2 \wedge \ldots \wedge p_m \rightarrow c$, where m is the number of literals in the conditional part of rule r ($m \geq 0$), p_i is the i-th literal in the conditional part of rule r ($i = 1 \ldots m$) and c denotes the literal of the decisional part of rule r. For each rule $r \in \mathcal{R}$ we define the following functions: $concl(r)$ — returns the conclusion literal of rule r: $concl(r) = c$. We will also consider *facts* as clauses without any conditional literals. The set of all such clauses f will be called *set of facts* and will be denoted by $\mathcal{F}$: $\mathcal{F} = \{f : \forall_{f \in \mathcal{F}}\ cond(f) = \emptyset \wedge f = concl(f)\}$.

B. Backward chaining inference

Backward chaining inference is a bottom-up procedure which starts with a main goal and queries the fact base about information which may satisfy the conditions contained in the rules. We basically go through the rules in the knowledge base looking for conclusions which match the query and if we find them, we can create new queries (adding new facts if necessary). Backward chaining inference is goal-driven and appropriate for problem-solving. Its complexity can be linear or less (in the size of the knowledge base), depending on the implementation. The main idea and general description of a classical backward chaining inference algorithm have been repeatedly published, for example in [11], [12]. But it is hard to find a detailed, step-by-step algorithm and for this reason we present the pseudo-code of such an algorithm bellow.

The presented backward chaining inference algorithm is a recursive one — recursion is used to confirm the sub-goals of inference. Typically the user is the source of facts, but

Algorithm 1: Backward chaining inference

```
Require: R: the set of rules, F: the set of facts, g: the goal
Ensure: F
  function backwardInference(R, F, g) : boolean
  var truePremise : boolean
  var R : RulesSet {Working rules set}
  var r : Rule {Working rule}
  var w : Literal {Working condition}
  begin
  if g ∈ F ∨ ¬g ∈ F then
     return g ∈ F
  end if
  truePremise ← false
  R ← select R ⊆ R where ∀r ∈ R : concl(r) = g
  while ¬truePremise ∧ ¬R ∈ ∅ do
     r ← select r ∈ R using selection strategy
     for each w ∈ cond(r) do
        truePremise ← w ∈ F
        if ¬truePremise then
           truePremise ← backwardInference(R, F, w)
        end if
        if ¬truePremise then
           truePremise ← environmentConfirmsFact(w)
        end if
        if ¬truePremise then
           break
        end if
     end for
     if ¬truePremise then
        R ← R − {r}
     end if
  end while
  if truePremise then
     F ← F − {concl(r)}
  end if
  return truePremise
  end function
```

in the context of embedded systems, facts can be provided by any technical equipment. In the experiments discussed further in this work, this step of the algorithm is omitted to obtain a pessimistic time complexity of the backward chaining inference.

C. Related works

The knowledge based systems were typically developed as desktop applications. Meanwhile, web applications have grown rapidly and have had a significant impact on the application of a traditional expert system. The detailed discussion and comparison of modern knowledge-based systems building tools goes beyond the scope of this study. Selected aspects of such review can be found in [13], [14] and also in [15]–[17]. In this work only a basic overview is presented. The JESS is a well-known and popular tool. It is the skeleton of expert systems developed by Sandia National Laboratories. JESS is written in Java and it is possible to run code in this language using JESS. It uses a syntax similar to Lisp [3]. It is compatible with both Windows and Unix systems. Rules written using JESS are saved in the form of an XML file which must contain a *rule-execution-set* element [18]. JESS is a rule engine as well as a scripting language, which provides a console for programming and enables basic input/output operations. JESS is a forward chaining inference engine, it provides mechanisms that „simulate" backward chaining.

EXSYS Corvid [6] is a software tool for building and fielding knowledge automation expert system applications. It is designed to be easy to learn and aimed at non-programmers. The Java based EXSYS Inference Engine makes it simple to deploy systems on different platforms and to integrate them with external programs. The rules (in the knowledge base) are described simply in English and Algebra. Tree-structured logic diagrams are used to describe individual sections of the process. It's a combination of „Logic Block" structures (which describe rules) and "Command Blocks" that provide procedural control on system execution. Corvid's Inference Engine uses both backward and forward chaining algorithms.

Another commercial expert system building tool is XpertRule [19], which offers a Knowledge Builder Rules Authoring Studio. The XpertRule KBS interfaces over the Web with a thin client using the Microsoft's Active Server Page technology. Applications developed using the Knowledge Builder Rules Authoring Studio can be generated as Java Script/HTML files for deployment as Web applications. The Web Deployment Engine is a JavaScript rules runtime engine which runs within a browser.

Similar concepts share the eXpertise2Go's Rule-Based Expert System, which provides free building and delivery tools that implement expert systems as Java applets, Java applications and Android apps [20]. Next interesting system is Drools — a Business Rules Management System solution. It provides a core Business Rules Engine, a web authoring and rules management application (Drools Workbench) and an Eclipse IDE plugin for core development [5].

This work introduces an another decision support system building tool — the `KBExplorator` system [7]. It is a web application and it allows the user to create, edit and share rule knowledge bases. It is also connected with the `KBExpertLib` — a software library, which allows programmers to use different kinds of inference within any software projects implemented in the Java programming language. The `KBExplorer` works on the client side and requires only the usage of a typical modern web browser. Knowledge bases created by the user are stored in a relational database, and may be shared between the registered system's users [21]. Moreover, it is also possible to download the knowledge base as an XML file (for further analysis). The `KBExpertLib` is a software library, which allows programmers to implement domain knowledge based systems using the Java programming language. This library makes it possible to run different kinds of inference (classical and modified forward and backward chaining algorithms) on rule-based knowledge bases stored in the `KBExplorer` database or saved locally in the form of XML files [22] (see fig. 1), the `KBExpertLib` is also considered as a tool for implementation of systems described in [23], [24].

III. Methods and tools

This section presents two approaches for backward chaining inference (illustrated on the fig. 1 and 2) described earlier in the introduction.

A. Backward chaining inference as Java code

The `KBExpertLib` provides a backward inference presented by algorithm 1 above, algorithm is implemented as a member function of the `KBBackwardConsoleInferer` class. In this work we consider inference without interaction with the system's environment, we want to analyse the worst possible case in terms of inference time. The `KBBackwardConsoleInferer` is a base class for a specialized derived class, which has to provide a fact confirmation function (algorithm 1: `environmentConfirmsFact`). Implementation of this function depends on a particular fact confirmation process. In research described later in the article, the implementation of this function is trivial — it always returns a *false* value, because we want to obtain the pessimistic inference algorithm complexity. Implementation of a backward chaining inference algorithm is typical and obvious — much more interesting is the implementation as stored procedure, described in the next section.

B. Backward chaining inference as the stored procedure

This section presents the second of the earlier mentioned approaches of inference realization. This approach works in the database server layer as a stored procedure, which uses native properties of the MySQL engine.

Algorithm 2: Backward chaining inference as the stored procedure

Require: $\mathcal{R}$: the set of rules, $\mathcal{F}$: the set of facts, g: the goal
Ensure: $\mathcal{F}$
procedure `backwardInference`($\mathcal{R}$, $\mathcal{F}$)
1: **var** $tempFacts$: FactSet
2: **var** $subGoals$: GoalSet
3: **var** $curG$: CurrentGoal
4: **begin**
5: $subGoals \leftarrow g$
6: **while** $subGoals \neq \emptyset$ **do**
7: $curG \leftarrow$ **select first** $g \in subGoals$
8: $tempFacts \leftarrow tempFacts \cup \{curG\}$
9: $subGoals \leftarrow subGoals \backslash \{curG\}$
10: **if** $\neg curG \in \mathcal{F}$ **then**
11: **select** $R \subseteq \mathcal{R} : r \in R, concl(r) = curG$
12: **if** $r = \emptyset$ **then**
13: **return** Inference failed
14: **end if**
15: $subGoals \leftarrow subGoals \cup \{cond(r) : cond(r) \not\subseteq tempFacts \wedge cond(r) \not\subseteq subGoals\}$
16: **end if**
17: **end while**
18: $\mathcal{F} \leftarrow \mathcal{F} \cup \{tempFacts\}$
19: **return** Inference succeeded
20: **end procedure**

The presented inference algorithm assumes that any selected goal will be confirmed after testing enough subgoals, because we are interested only in the worst-case scenario. What is more, the algorithm was repeated in the experiments sections, for all possible inference goals (distinct rule's conclusions). The pseudocode of the described approach is presented in algorithm 2 and the exemplary implementation of the stored procedure is as follows:

```
main_infer:BEGIN
  DECLARE rId INT;
  DECLARE FK_subgoalAttributeID INT;
  DECLARE FK_subgoalValueID INT;
  DECLARE subgoalOperator CHAR(3);
  DECLARE subgoalRowID BIGINT;
  DECLARE subgoalContinousValue VARCHAR(45);
  DECLARE exitLoop BIT(1);
  DECLARE CONTINUE HANDLER FOR 1329 SET exitLoop = 1;
  SET exitLoop = 0; SET kbID = 1;
  #Clear temporary tables
  TRUNCATE TABLE subgoals;
  TRUNCATE TABLE tempfacts;

  #Insert the main inference goal to the subgoals table
  INSERT INTO subgoals(FK_attributeID, operator, FK_valueID,
  continousValue, FK_knowledgeBase) VALUES (3, '==', 3, NULL,
  kbID);

  read_loop: LOOP

  #Select first subgoal
  SELECT subgoalID, FK_attributeID, operator, FK_valueID,
  continousValue INTO subgoalRowID, FK_subgoalAttributeID,
  subgoalOperator, FK_subgoalValueID, subgoalContinousValue
  FROM subgoals ORDER BY subgoalID DESC LIMIT 1;

  IF exitLoop = 1 THEN
    LEAVE read_loop;
  END  IF;

  #Insert the current subgoal to the temporary facts table
  INSERT INTO tempfacts SELECT NULL, FK_attributeID, operator,
  FK_valueID, continousValue, kbID FROM subgoals WHERE
  subgoalID = subgoalRowID;

  #Delete the current subgoal
  DELETE FROM subgoals WHERE subgoalID = subgoalRowID;

  #Check if the current subgoal is a fact
  IF NOT EXISTS (SELECT * FROM facts WHERE FK_attributeID =
  FK_subgoalAttributeID AND operator = subgoalOperator AND
  FK_valueID = FK_subgoalValueID AND continousValue <=>
  subgoalContinousValue) THEN

    #Get the ID and later on all premises of the rule which
    #conclusion in the selected subgoal
    SET rId = (SELECT FK_ruleId FROM attributeValue WHERE
    FK_attributeID = FK_subgoalAttributeID AND
    operator = subgoalOperator AND FK_valueID = FK_subgoalValueID
    AND continousValue <=> subgoalContinousValue AND
    isConclusion = 1 LIMIT 1);

    #If there is no such rule, inference cannot complete
    IF rId IS NULL THEN
      SELECT "INFERENCE FAILURE";
      LEAVE read_loop;
    END  IF;

    #Insert selected premises to the subgoals table,
    #excluding those which are facts or were already processed
    INSERT INTO subgoals
    (FK_attributeID, operator, FK_valueID, continousValue)
    SELECT FK_attributeID, operator, FK_valueID, continousValue
    FROM attributeValue t1 WHERE FK_ruleId = rId AND
    isConclusion IS NULL AND NOT EXISTS
    (SELECT 1 FROM tempfacts t2 WHERE t2.FK_attributeID =
    t1.FK_attributeID AND t2.operator = t1.operator AND
    t2.FK_valueID = t1.FK_valueID AND t2.continousValue <=>
    t1.continousValue)AND NOT EXISTS
    (SELECT 1 FROM subgoals t3 WHERE t3.FK_attributeID =
    t1.FK_attributeID AND t3.operator = t1.operator AND
    t3.FK_valueID = t1.FK_valueID AND t3.continousValue <=>
    t1.continousValue );

  END IF;
  END LOOP;

  #If all subgoals were confirmed inference is a success
  IF exitLoop = 1 THEN
   SELECT "INFERENCE SUCCESS";
  END IF;
END
```

The above-mentioned implementation assumes that the initial set of facts is already known and the user has set the

inference goal to a descriptor expressed internally as a pair of $attributeID = 3$ and $valueID = 3$ (which is also the conclusion of the first rule in the knowledge base). What is more, two temporary tables ($subgoals$ and $tempfacts$ storing respectively the current set of inference subgoals and a temporary set of facts, which will be valid if the inference succeeds) use internally the *MEMORY Storage Engine*. The *MEMORY Storage Engine* which is a part of the MySQL DBMS ensures that data in such tables will be kept in the server's RAM memory – this is required due to many insert and delete operations.

The main loop starts by selecting the first subgoal from the $subgoals$ table – currently there is only the main inference goal (line 7 of algorithm 2). The subgoal is inserted into the temporary set of facts (represented by the $tempfacts$ table) and removed from the $subgoals$ table (lines 8 and 9). If the current subgoal of the inference does not belong to the set of initial facts, the procedure selects the ID of the first (not analysed) rule from the knowledge base which conclusion corresponds to the subgoal (line 11). If such a rule does not exist, the inference ends with failure. This is not the case, and so premises of the selected rule are added to the subgoals table (line 15), excluding those which are in the set of temporary facts ($tempfacts$) or were already processed (are included in the $subgoals$ table). If all subgoals are confirmed, the main loop ends and inference is considered as a success. Entries from the $tempfacts$ table can now be treated as new facts.

IV. Experiments

This section presents the experiments performed on four real-world knowledge bases. The first used knowledge base (`bud4438`) is used by a builder company in Poland and currently consists of 4438 rules. There are 2802 symbolic attributes and 51 numeric. The formed (by domain experts) rules were generally very short and the structure of the knowledge base was flat. The second base (`bud22190`) was generated by duplicating the first one five times with random modifications (because gaining access to large, real-world knowledge bases is very difficult). The third (`eval416`) and fourth (`eval1199`) knowledge bases regarded the topic of effectiveness evaluation of sales representatives and consisted of 416 and 1119 rules. More information about the structure and experimental evaluation of rules partitioning concerning these knowledge bases can be found in [25].

The aim of the first experiment was to evaluate the data loading times from a relational database. The results of this experiment (shown in table I) were needed to confirm the usefulness of the proposed backward chaining inference algorithm implemented directly on the database and should be interpreted in the context of the second experiment.

Results presented in table I clearly indicate, that loading times from the database can be regarded as significant. It is especially visible for the `bud4438` and `bud22190` knowledge bases, because they have a lot of attributes which definitions (and a list of possible values) are also stored besides the rule set. That is why the authors wanted to check if performing operations directly on the database server can be an alternative to having to wait for the data loading phase to finish in order to perform e.g. inference in the client's application. The memory usage for data structures which hold the rules seems to be reasonable. For 22190 rules the data structures occupy less than 5MB of memory.

TABLE I
Knowledge base average loading times

Knowledge base	Rules count	Loading time from database [s]	Memory occupation [KB]
eval416	416	2,092	96
eval1199	1199	5.185	285
bud4438	4438	29.238	1197
bud22190	22190	188.195	4646

The goal of the second experiment was to compare two (previously described in this work) methods of backward chaining inference — a classical version implemented in the Java programming language and one that operates directly on the database server. The results of this experiment are presented in table II.

TABLE II
Analysis of rules' retrieval time (in seconds)

Knowledge base	Data source	Min	Max	Mean	Median
eval416	Database	16.890	17.234	17.016	16.946
eval416	RAM	0.002	0.267	0.079	0.075
eval1119	Database	52.172	53.781	52.844	52.828
eval1119	RAM	0.002	1.377	0.383	0.367
bud4438	Database	13.063	14.969	13.961	13.899
bud4438	RAM	0.001	0.753	0.543	0.478
bud22190	Database	136.000	137.000	136.333	136.000
bud22190	RAM	0.001	2.462	1.567	1.342

It is obvious that inference realized on objects stored in the RAM of a client's computer will be faster than performing the same process directly in the database layer (even when using proper column indexes). But the results in table II should be interpreted in the context of knowledge base loading times (see table I). Then one can observe an advantage of the stored procedure method (in case of knowledge bases `bud4438` and `bud22190`) compared to running inference on the desktop application. In case of the Java program the user has to wait a time period of about 29 (in case of `bud4438`) or 188 (in case of `bud22190`) seconds for the database to load and additionally 0.2 or 2.5 seconds for the inference process to finish.

When performing inference directly on the database this time is reduced to a bit over 13 or 136 seconds respectively. This means that the user will be able to perform inference at least one to two times (directly on the database) whereas the data would still be loading in the desktop application. Of course when the user plans to perform inference multiple times it is still better to load the data into the desktop application, because it's a one-time operation only. As far as the `eval416` and `eval1119` knowledge bases are concerned, the direct database approach performs worse than the traditional one.

This is caused by the structure of these knowledge bases. Although they store less rules, they form a hierarchical structure, and a chain of rules is activated and analysed instead of only a single rule like in the case of the `bud4438` and `bud22190` bases which have a flat structure.

The results from the second experiment can also have practical impacts — a „smart" software dispatcher, which can choose the best inference scenario according to the current user's device properties — hardware configuration, internet availability, type of power supply — can be made. We are going to include this solution into the `KBExplorer` system and the `KBExpertLib` library in the future.

V. Conclusions

In this work two approaches for inference implementation, which uses knowledge bases stored in the form of a relational database, were introduced. The first approach requires a priori loading of the knowledge base contents to the RAM of the client's computer. The inference process is performed later on using only data stored in the RAM. The second approach involves implementing inference as a stored procedure, run in the environment of the database server.

Experiments were conducted on real-world knowledge bases with a relatively large number of rules. Experiments were prepared so that one could evaluate the pessimistic complexity of the inference algorithm. The results confirmed, that the inference implemented in object-oriented data structures loaded into memory is effective. The times of inference in the worst-case scenario did not exceed 2.5 seconds. However, loading the contents of the knowledge base proved to be time consuming. For small bases these times were about a couple of seconds, but for the largest one they reached over 3 minutes. Such a case is acceptable in systems where the knowledge base is reloaded rarely, and the waiting time (for data loading) is tolerable from the user's point of view. For applications where there is need for frequent reloading of the knowledge base, this solution is inconvenient and may be cumbersome for the user. This may be the case for knowledge bases which are frequently updated, e.g. by programs that use specific algorithms to automatically generate rules.

The inference implemented in the form of a stored procedure runs significantly slower than the solution described previously, which is not a surprising result. However, when comparing the inference times and adding the time of loading data from the knowledge base, the solution using a stored procedure turns out to be faster (in specific conditions). This solution is especially convenient when the knowledge base is updated frequently.

Implementation of the inference in the form of a stored procedure is an interesting solution and will be permanently included into the described `KBExplorer` system and the `KBExpertLib` library. This will allow the programmer implementing a domain expert system to select the desired mode of inference. We consider the implementation of an „intelligent" inference dispatcher, which will be able to select the inference method, according to the device resources (processor speed, free memory) as well as taking into account the speed of the used internet connection. This solution will be analysed as the next stage of research.

References

[1] R. Akerkar and P. Sajja, *Knowledge-based systems*. Jones & Bartlett Publishers, 2010.

[2] P. S. Sajja and R. Akerkar, "Knowledge-based systems for development," *Advanced knowledge based systems: Model, Application & Research*, pp. 1–11, 2010.

[3] JESS, "JESS Information," http://herzberg.ca.sandia.gov, accessed November, 2016.

[4] CLIPS, "CLIPS NASA Home Page," http://www.siliconvalleyone.com/founder/clips/index.htm, accessed November, 2016.

[5] DROOLS, "DROOLS Home Page," https://www.drools.org, accessed November, 2016.

[6] EXSYS, "EXSYS Home Page," http://www.exsys.com, accessed November, 2016.

[7] R. Simiński and A. Nowak-Brzezińska, "Kbexplorator and kbexpertlib as the tools for building medical decision support systems," in *International Conference on Computational Collective Intelligence*. Springer, 2016, pp. 494–503.

[8] R. Simiński, "The kbexpertlib software library for java–functionality properties and performance study," *Studia Informatica*, vol. 37, no. 1, pp. 125–134, 2016.

[9] A. Nowak-Brzezińska and R. Simiński, "Goal-driven inference for web knowledge based system," in *International Conference on Information Systems Architecture and Technology ISAT'2015*. Springer, 2015.

[10] R. Simiński and M. Manaj, "Implementation of expert subsystem in the web application–selected practical issues," *Studia Informatica*, vol. 36, no. 1, pp. 131–143, 2015.

[11] J. W. Grzymala-Busse, *Managing uncertainty in expert systems*. Springer Science & Business Media, 2012, vol. 143.

[12] A. Ligeza, *Logical foundations for rule-based systems*. Springer, 2006, vol. 11.

[13] B. Ruiz-Mezcua, A. Garcia-Crespo, J. Lopez-Cuadrado, and I. Gonzalez-Carrasco, "An expert system development tool for non ai experts," *Expert Systems with Applications*, vol. 38, no. 1, pp. 597–609, 2011.

[14] H. Mathkour, I. Al-Turaiki, and A. Touir, "The development of a bilingual fuzzy expert system shell," *Journal of King Saud University-Computer and Information Sciences*, vol. 21, pp. 27–44, 2009.

[15] R. Grove, "Internet-based expert systems," *Expert systems*, vol. 17, no. 3, pp. 129–135, 2000.

[16] Y. Duan, J. S. Edwards, and M. Xu, "Web-based expert systems: benefits and challenges," *Information & Management*, vol. 42, no. 6, pp. 799–811, 2005.

[17] D. Huntington, "Web-based expert systems are on the way: Java-based web delivery," *PC AI*, vol. 14, no. 6, pp. 34–36, 2000.

[18] J. Canadas, J. Palma, and S. Túnez, "A tool for mdd of rule-based web applications based on owl and swrl," *Knowledge Engineering and Software Engineering (KESE6)*, p. 1, 2010.

[19] Xpert Rule, "Xpert Rule Home Page," http://www.xpertrule.com, accessed November, 2016.

[20] eXpertise2Go, "eXpertise2Go Home Page," http://expertise2go.com, accessed November, 2016.

[21] R. Simiński and T. Xięski, "Physical knowledge base representation for web expert system shell," in *International Conference: Beyond Databases, Architectures and Structures*. Springer, 2015, pp. 558–570.

[22] T. Xięski and R. Simiński, "A performance study of two inference algorithms for a distributed expert system shell," in *International Conference: Beyond Databases, Architectures and Structures, in print*. Springer, 2017.

[23] M. Przybyła-Kasperek and A. Wakulicz-Deja, "Global decision-making system with dynamically generated clusters," *Information Sciences*, vol. 270, pp. 172–191, 2014.

[24] A. Nowak-Brzezińska, "Mining rule-based knowledge bases inspired by rough set theory," *Fundamenta Informaticae*, vol. 148, no. 1-2, pp. 35–50, 2016.

[25] R. Simiński, "The experimental evaluation of rules partitioning conception for knowledge base systems," in *Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology–ISAT 2016–Part I*. Springer, 2017, pp. 79–89.