Journal of Universal Computer Science, vol. 14, no. 14 (2008), 2263-2287
submitted: 30/9/07, accepted: 30/4/08, appeared: 28/7/08 © *J.UCS*

The APS Framework For Incremental Learning of Software Agents

Damian Dudek
(The University of Information Technology and Management "Copernicus"
ul. Inowroclawska 56, 53-648 Wroclaw, Poland
ddudek@wsiz.wroc.pl)

Abstract: Adaptive behavior and learning are required of software agents in many application domains. At the same time agents are often supposed to be resource-bounded systems, which do not consume much CPU time, memory or disk space. In attempt to satisfy both requirements, we propose a novel framework, called APS (standing for *Analysis of Past States*), which provides agent with learning capabilities with respect to saving system resources. The new solution is based on incremental association rule mining and maintenance. The APS process runs periodically in a cycle, in which phases of agent's normal performance intertwine with learning phases. During the former ones an agent stores observations in a history. After a learning phase has been triggered, the history facts are analyzed to yield new association rules, which are added to the knowledge base by the maintenance algorithm. Then the old observations are removed from the history, so that in the next learning runs only recent facts are processed in search of new association rules. Keeping the history small can save both processing time and disk space as compared to batch learning approaches.

Key Words: statistical learning, incremental methods, software agents, web browsing assistant
Category: I.2.6, I.5.0, M.0, M.3

1 Introduction

Adaptivity and learning plays an important role in agent-related research. Since Patti Maes defined the notion of adaptive agents [Maes, 1994], numerous approaches have been developed both for learning of a single agent (*Single Agent Learning*, SAL) and multi-agent learning (MAL) [Sen and Weiss, 1999]. In many of these works general machine learning methods were adapted to work within agent architectures, e.g.: model-based learning [Maes, 1994, Yao et al. 2002] or reinforcement learning [Ribeiro, 2002]. There are also solutions, where the learning process is an integral part of an agent architecture, such as *chunking* in Soar [Laird et al. 1987, Newell, 1990].

We propose a new, general framework called APS (standing for *Analysis of Past States*) for incremental, statistical learning of an agent. It uses data mining algorithms in order to discover association rules in a large set of observations, which are collected by an agent while interacting with an information-rich environment. The APS method first of all is meant to be used within a learning

agent, which operates in a cycle with interlaced phases of performance and being idle. During the former phase the agent works on current tasks: it conducts reasoning, performs actions and perceives information from the outside world. At the same time agent stores in its history some of observed data (e.g. results of performing an action in a given situation). As the observed facts are often too numerous to be efficiently processed on-line (during the performance phase), they are just accumulated as experience for further analysis and learning. A good opportunity for processing the history observations occurs during a stand-by phase, when the agent is not charged with any current task, consequently not utilizing much system resources. Then the agent can switch to learning - analyzing the history facts, discovering new rules and adding them to the rule base with appropriate maintenance of previously existing knowledge.

Consider the following example of a personal web browsing recommender agent, which helps a user choose relevant pages, using a model of preferences, based on accumulated experience. The agent is a silent background process, running at a PC, observing WWW browsing activity and being able to log events, which indicate user's interest (such as bookmarking, saving or printing a web page). Moreover, the agent provides a simple interface for explicit evaluation of viewed pages (e.g. relevant or irrelevant), if the user wishes so. Every time a user responds to a viewed page in one of the above ways, the agent adds a new fact into the history, with page information (index terms which are frequent in the text, query context etc.) and user's response (*bookmarked*, *saved*, *printed*, *relevant* etc.). Depending on browsing activity, the agent can store even hundreds or thousands of such observations weekly. When the history contains a sufficient, representative number of facts[1] and system resources are not loaded heavily, the agent analyses the history, generalizing accumulated knowledge into association rules (e.g. $architecture \wedge development \wedge schema \Rightarrow relevant \wedge stored\_yes$), which can be used later for recommending potentially relevant pages to the user. After the rule mining has been completed, the history is cleared of all the processed facts and newly discovered rules are added to the existing ones using such a maintenance procedure, that the resulting rule base reflects not only the recent analysis, but learning in a longer period. Then the agent can return to normal performance phase, working on tasks and storing new observations for further learning runs. As the history is regularly emptied, its size is kept relatively small, which not only saves disk space, but also CPU time used for learning as compared to batch processing of all the facts stored during the agent's life-time. Moreover, proper maintenance of the rule base makes it contain stable and statistically reliable rules as if they were discovered in the whole observation history, stored during agent's life.

[1] Setting a threshold of representative history size is up to a system designer or administrator.

The remainder of the paper is as follows. In section 2 we present a high-level view of the APS learning procedure, while in section 3 we introduce the formal model of knowledge structures and specification of data processing algorithms. In section 4 we show how APS works on an example of a web browsing assistant. The next section contains the plan of experimental evaluation, test data description and discussion of the results. In section 6 the proposed method is compared to related research. Finally, we conclude the contributed work and outline directions of its further development.

2 APS Learning cycle

The key idea of the APS method is to incorporate a general data mining process [Mannila, 1997] into an agent architecture, in order to achieve a rule-based, statistical learning technique. It is based on the procedure, running in a cycle and interacting with the agent's knowledge base, which consists of four functional modules (see Figure 1). The history KB_H is a log of collected observations, potentially very numerous. The rule base KB_R stores rules produced from history facts as their generalization, ready for use in reasoning and decision making. The temporal knowledge module KB_T is a kind of short term memory, which contains operational data necessary within a single learning run, whereas the general knowledge KB_G is a long-term memory with information needed across many learning runs. The beneath *APS Learning* procedure shows the details of the learning process and its interaction with knowledge base modules (input and output data flow).

The key steps of data processing are performed by the algorithms: FSEL, HTRANS, HFILL, ENV, ARM and RMAIN, which are original contribution of this work. The whole loop reflects agent's life cycle with repeatable performance phases (steps 2-3) and learning phases (steps 4-13). While being in the performance phase an agent works towards its objectives, saving in the history KB_H selected information about its interaction with the outer world. The designer needs to decide what kind of event is to be captured as a *new fact event* and what kind of information is to be stored. In the previous web browsing scenario such an event can be bookmarking, saving, printing or evaluating a web page by a user, while the registered information could be frequent index terms describing the page, query context and user's response. Every new fact put into the history is assigned a unique identifier and timestamp of the very moment of observation. When the number of facts in the history is sufficient (due to some threshold set by a designer) and the agent has no current tasks to perform, the *analyze facts event* is raised and the APS procedure enters the learning phase (steps 4-13). First, the agent retrieves parameter values from previous learning runs, stored in the KB_G general knowledge module (step 5). They are necessary for providing continuous processing of incremental data portions. In the next step the

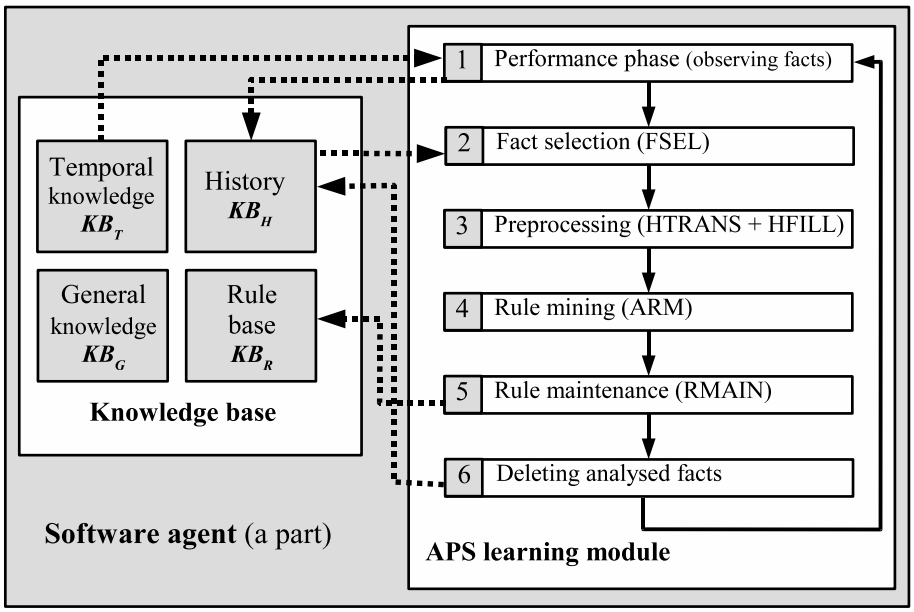

Figure 1: The APS learning cycle in interaction with knowledge base modules.

Procedure 1 APS Learning

Input: KB_G, KB_T, KB_H, KB_R

Output: updated KB_G, KB_T, KB_H, KB_R

```
1: while not Event_Close_Agent_Process do
2:   if Event_New_Fact then
3:     Store_Fact(input KB_H, output KB_H)
4:   else if Event_Analyse_Facts then
5:     Get_Parameters (input KB_G, output KB_T)
6:     Select_Facts (input KB_H, KB_T, output KB_H, KB_T)
7:     Transform_History_Schema (input KB_H, output KB_H, KB_T)
8:     Fill_New_Schema (input KB_H, output KB_H)
9:     Remove_N_Values (input KB_H, KB_T), output KB_H, KB_T)
10:    Mine_Rules (input KB_H, KB_T, output KB_T)
11:    Update_Rule_Base (input KB_R, KB_T, KB_G, output KB_R, KB_G)
12:    Delete_Facts (input KB_H, KB_T, output KB_H)
13:  end if
14: end while
```

FSEL algorithm chooses a set of recent history facts for rule discovery in the current run. By default all the observations in KB_H are retrieved. Afterwards, the selected facts are transformed to the acceptable input format for a data mining algorithm - namely transactions with binary attributes. First, a new history schema is created, using the HTRANS algorithm (step 7) and then it is filled with transformed facts by the HFILL algorithm (step 8). During further preprocessing transactions with unknown values are replaced by all the possible facts, using the ENV algorithm (step 9). The prepared history facts are then analyzed by an association rule mining algorithm (e.g. Apriori [Agrawal et al. 1994]), incorporated into the ARM algorithm (step 10), which plays a role of an interface between a mining algorithm and the APS procedure. After a set of new association rules R has been discovered, it is combined with the former rule base in KB_R (step 11) so that the resulting rule set reflected all the facts analyzed so far (i.e. in the current and previous learning runs). The necessary rule maintenance is done by the RMAIN algorithm, which is truly essential for the whole method. Then, the recently processed history observations are disposed (step 12), updated parameter values needed for further learning runs are saved in the general knowledge KB_G, while the temporal knowledge module KB_T is cleared of all the information concerning the recent run and the agent switches back to the performance phase.

The presented approach of periodic clearing the history can be classified as *learning with imperfect recall* [Dudek et al. 2005, Fagin et al. 1995] and *partial memory learning* [Maloof and Michalski, 2004]. Nevertheless, although the agent *forgets* elementary past events, it keeps and maintains rules, which are general conclusions about observations, potentially useful for reasoning.

3 Formal model

After the overview of the APS method given in the previous section, we present formal specification of knowledge base structures and data processing algorithms. Some preliminary elements of the APS model were introduced briefly in the previous work [Dudek et al. 2005], while a general presentation of the APS method without detailed formalism was introduced in [Dudek, 2007].

3.1 Knowledge base structure

Below we formally define data structures and some auxiliary elements, which are used in the APS learning cycle. The presented model is partly based on the relational database notation by Pankowski [Pankowski, 1992]. It also refers to models of transaction databases and association rules provided in [Agrawal et al. 1993,

Agrawal et al. 1994]. However, our model is significantly richer, covering non-binary and multi-valued attributes, extended rule representation, time depreciation functions and original APS agent's knowledge modules.

Definition 1 Fact. Let $S_H = \{K, T, U\}$ be called *a schema*, where the symbol K is *a key*, the symbol T is *time*, and U is a finite set of attributes $U = \{A_1^S, \ldots, A_n^S, A_1^M, \ldots, A_k^M\}$. The elements $A_i^S \in U$, for $i = 1, \ldots, n$ ($n \in \mathbb{N}$) are called *single-valued attributes*, and $A_j^M \in U$, for $j = 1, \ldots, k$ ($k \in \mathbb{N}$) are called *multi-valued attributes*. Let K and T be assigned respectively a set $D_K \subseteq \mathbb{N}$ called the *key domain* and a countable set of time points D_T, which is ordered by a strong linear order relation $<$. Let each single-valued attribute A_i^S, for $i = 1, \ldots, n$ be assigned *a domain*, which is a finite set of values D_i^S. Let each multi-valued attributed A_j^M, for $j = 1, \ldots, k$, be assigned *a domain*, which is a set $D_j^M = 2^{V_j}$ of all possible subsets of a finite set V_j. A *fact of a schema* $S_H = \{K, T, U\}$ is defined to be any function s, such that: $s\colon \{K, T, U\} \rightarrow D_K \cup D_T \cup \bigcup\{D_i^S\colon A_i^S \in U, i = 1, \ldots, n\} \cup \bigcup\{D_i^M\colon A_j^M \in U, j = 1, \ldots, k\}$.

Multi-valued attributes are very useful for description of real-life objects. For instance a web page can be characterized by a multi-valued attribute *term*, which is assigned a collection of values – terms describing this document. We denote a fact s of a schema $S_H = \{K, T, U\}$ as a set of attribute-value pairs $s = \{(K, k), (T, t), (A_1^S, a_1^S), \ldots, (A_n^S, a_n^S), (A_1^M, a_1^M), \ldots, (A_k^M, a_k^M)\}$, where $A_i^S, A_j^M \in U$, for $i = 1, \ldots, n$ and $j = 1, \ldots, k$.

Values of the key K, time T, a single-valued attribute A_i^S and a multi-valued attribute A_j^M in a fact s are denoted as: $K(s) = k$, $T(s) = t$, $A_i^S(s) = a_i^S$, and $A_j^M(s) = a_j^M$, respectively. We denote an unknown value of an attribute (single- or multi-valued) by a symbol N. A set of all single-valued attributes $A_i^S \in U$ for $i = 1, \ldots, n$ in a schema $S_H = \{K, T, U\}$ is denoted by S_H^S, while similar set of all multi-valued attributes $A_j^M \in U$ for $j = 1, \ldots, k$ is denoted by S_H^M. The following properties are satisfied: $S_H^S \subseteq S_H \backslash \{K, T\}$, $S_H^M \subseteq S_H \backslash \{K, T\}$, and $S_H^S \cap S_H^M \equiv \varnothing$. A set of all the facts of a schema S_H is denoted by a symbol $FACT(S_H)$. We allow solely discrete attribute domains, which potentially implies necessity of discretization for continuous domains (e.g. real numbers). However, dealing with problem is strongly domain-dependent and goes beyond the scope of this paper.

Definition 2 History. Let a schema $S_H = \{K, T, U\}$ be given. *A history* KB_H *of a schema* S_H is any subset of the set of all facts of the schema S_H. Formally: $KB_H \subseteq FACT(S_H)$.

We denote: the $k-th$ time point belonging to the set D_T by t_k, the time point of the first fact in the history by t_0, the time point referring to the presence by t_{now}, and the time interval from the moment t_i until t_j, where $t_i \leqslant t_j$, by $[t_i; t_j]$. For each fact $s \in KB_H$ the following dependency is satisfied: $t_0 \leqslant T(s) \leqslant t_{now}$.

Definition 3 Fact portion. Let a history KB_H of a schema $S_H = \{K, T, U\}$ be given. *A fact portion* of the history KB_H for the time interval $[t_1; t_2]$, where $t_1, t_2 \in D_T \wedge t_1 \leqslant t_2$, is the set: $KB_H(t_1, t_2) = \{s \in KB_H : t_1 \leqslant T(s) \leqslant t_2\}$.

Definition 4 Proper domain of a single-valued attribute. Let a history KB_H of a schema $S_H = \{K, T, U\}$ be given. *A proper domain of a single-valued attribute* $A_i^S \in S_H^S$ in the history KB_H for the time interval $[t_1; t_2]$, where $t_1, t_2 \in D_T \wedge t_1 \leqslant t_2$, is a set: $D_i^W(KB_H(t_1, t_2)) = \{A_i^S(s) \neq N \colon s \in KB_H \wedge (t_1 \leqslant T(s) \leqslant t_2)\}$.

Definition 5 Proper domain of a multi-valued attribute. Let a history KB_H of a schema $S_H = \{K, T, U\}$ be given. *A proper domain of a multi-valued attribute* $A_j^M \in S_H^M$ in the history KB_H for the time interval $[t_1; t_2]$, where $t_1, t_2 \in D_T \wedge t_1 \leqslant t_2$, is a set: $D_j^W(KB_H(t_1, t_2)) = \bigcup\{A_j^M(s) \neq \varnothing \,.\, s \in KB_H \wedge (t_1 \leqslant T(s) \leqslant t_2)\} \setminus \{N\}$.

The proper domain of an attribute is a set of all its values (except for the unknown value N), which are actually present in a given subset of history facts. Narrowing attribute values to their proper domains (as opposed to all the possible values) allows to speed up transformation of the history to the format with binary attributes and *N-values* $\{0, 1, N\}$.

Definition 6 Rule. Let a set $U' = \{A_1, A_2, \ldots, A_m\}$ of binary attributes be given. Each attribute A_i where $i = 1, 2, \ldots, m$ is assigned a set of values $D_b = \{0, 1, N\}$. Let a countable set of time points D_T be given, ordered by the relation $<$ of strong linear order. *A rule* r is defined as a tuple: $r = (X, Y, sup, con, b, t_m) \in 2^{U'} \times 2^{U'} \times [0; 1] \times [0; 1] \times \mathbb{N} \times D_T$, where: X is called antecedent, Y – consequent, sup – support, con – confidence, b – base number, t_m – mean time of the rule.

The above rule representation extends a classical association rule definition, used in many works (such as the one proposed in [Agrawal et al. 1993, Agrawal et al. 1994]). The novel elements needed for the APS incremental learning process are: (i) the base number b – the number of history facts, based on which the rule r was discovered, (ii) mean time t_m of these facts.

Definition 7 Syntactical equity of rules. Two rules $r_1 = (X_1, Y_1, sup_1, con_1, b_1, t_{m1})$ and $r_2 = (X_2, Y_2, sup_2, con_2, b_2, t_{m2})$, where $r_1, r_2 \in 2^{U'} \times 2^{U'} \times [0; 1] \times [0; 1] \times \mathbb{N} \times D_T$, are said to be *syntactically equal*, denoted by $r_1 \equiv r_2$, iff $X_1 \equiv X_2 \wedge Y_1 \equiv Y_2$.

Definition 8 Rule base. Let a set $U' = \{A_1, A_2, \ldots, A_m\}$ of binary attributes be given. Each attribute A_i for $i = 1, 2, \ldots, m$ is assigned a set of values $D_b = \{0, 1, N\}$. Let a countable set of time points D_T be given. A rule base KB_R is the following set:

$KB_R \subseteq \{r = (X, Y, sup, con, b, t_m) \in 2^{U'} \times 2^{U'} \times [0;1] \times [0;1] \times \mathbb{N} \times D_T\}$, where $\neg\exists r_1, r_2 \in KB_R.\ r_1 \equiv r_2$.

Definition 9 Vector of current run parameters. Let U and D_T be defined as before. *A vector of current run parameters* is a tuple:
$v_c = (id_c, X_c, Y_c, b_{max}, b_c, \eta_c, \sigma_c, \gamma_c, \hat{\sigma}_c, \hat{\gamma}_c, m_x, m_y, t_{rc}, t_{mc}, t_{sc}, t_{ec}, k_{ec}) \in$
$\mathbb{N} \times 2^U \times 2^U \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times [0;1] \times [0;1] \times [0;1] \times [0;1] \times \mathbb{N} \times \mathbb{N} \times D_T \times D_T \times D_T \times D_T \times \mathbb{N}$
where $\hat{\sigma}_c \in [0; \sigma_c] \wedge \hat{\gamma}_c \in [0; \gamma_c] \wedge t_{sc} \leqslant t_{mc} \leqslant t_{ec}$.

The symbol id_c is a unique run identifier, X_c and Y_c – a set of attributes, which are allowed in the rule antecedent and consequent, respectively; b_{max} – the maximal number of facts, which can be analyzed during the current run; b_c – the number of facts actually analyzed in the current run; η_c – the maximal number of allowed *N-values* in a single fact; σ_c and γ_c – thresholds of minimum rule support and confidence, respectively; $\hat{\sigma}_c$ and $\hat{\gamma}_c$ – expected support and confidence of rejected rules; m_x and m_y – the maximal number of attributes in the rule antecedent and consequent, respectively; t_{rc} – time of starting the current run; t_{mc}, t_{sc}, t_{ec} – the mean, the earliest and the latest time of facts analyzed in the current run; k_{ec} – the key of the last analyzed fact. In the APS framework the v_c vector of current run parameters belongs to the *temporal knowledge module* KB_T, i.e. $v_c \in KB_T$. We do not provide a formal definition of KB_T, as its structure depends on an application domain.

Definition 10 Time depreciation function. The *time depreciation function* is any function $f_T : [0; +\infty) \to [0; 1]$, such that: (i) $f_T(0) = 1$ and (ii) $\forall x_1, x_2 \in [0; +\infty).\ x_1 < x_2 \Rightarrow f_T(x_1) \geqslant f_T(x_2)$.

We denote the family of all f_T functions by $\mathcal{F}$. The argument of the f_T is time, which has passed since a given past moment t until t_{now}, i.e. $(t_{now} - t)$, for time points t, $t_{now} \in D_T$. For the current moment $t = t_{now}$ the f_T function achieves the maximal value 1. The actual formula and parameters of a f_T function depend on a given application (see Figure 2 for example shapes).

Definition 11 Vector of global parameters. Let $\mathcal{F}$ and D_T be defined as before. A vector of global parameters is a tuple:
$v_g = (b_g, \sigma_g, \gamma_g, \hat{\sigma}_g, \hat{\gamma}_g, t_{mg}, t_{sg}, t_{eg}, k_{eg}, f_T) \in \mathbb{N} \times [0;1] \times [0;1] \times [0;1] \times [0;1] \times$
$D_T \times D_T \times D_T \times \mathbb{N} \times \mathcal{F}$, where: $\hat{\sigma}_g \in [0; \sigma_g] \wedge \hat{\gamma}_g \in [0; \gamma_g] \wedge t_{sg} \leqslant t_{mg} \leqslant t_{eg}$.

The components of the vector concern all the learning runs performed so far: b_g – the number of facts which were analyzed; σ_g and γ_g – thresholds of minimal support and confidence; $\hat{\sigma}_g$ and $\hat{\gamma}_g$ – expected support and confidence of rejected rules; t_{mg} mean time of all processed facts; t_{sg} and t_{eg} – time of the first and the last analyzed fact; k_{eg} – the key of the last analyzed fact;

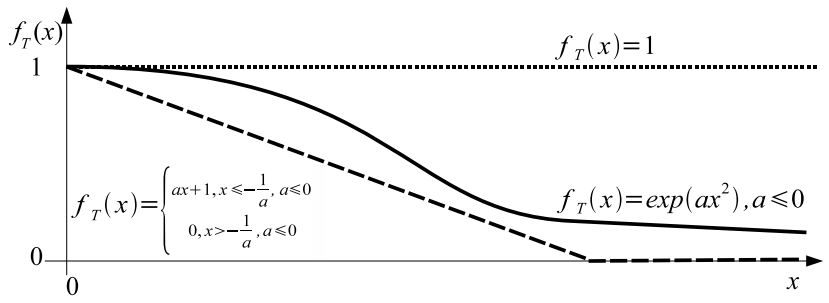

Figure 2: Example shapes of the f_T time depreciation function.

f_T – time depreciation function (formula and parameters). The defined vector of global parameters belongs to the *general knowledge module* (long-term memory), denoted by KB_G, together with the history schema, i.e. $v_g \in KB_G \wedge S_H \in KB_G$.

3.2 Data processing algorithms

In the current subsection we introduce original algorithms for data processing inside the APS cycle[1]. We omit two procedures *Store_Facts* (step 3) and *Get_Parameters* (step 5), which are domain-dependent, but also quite simple and straightforward. An intuitive example of how the proposed APS algorithms work is presented in Section 4.

3.2.1 Selecting facts for analysis

In the step 6 the APS procedure selects history facts, which are to be processed in the current learning cycle. The task can be described as follows. Given the history KB_H with a schema $S_H = \{K, T, U\}$ and the maximal number of facts to be retrieved b_{max}, find a portion of facts $KB_H(t_{sc}; t_{ec})$, such that: $|KB_H(t_{sc}; t_{ec})| \leqslant b_{max}$ and $\forall s \in KB_H(t_{sc}; t_{ec}).\ t_{sc} \leqslant T(s) \leqslant t_{ec} \wedge \neg\exists s' \in KB_H \setminus KB_H(t_{sc}; t_{ec}). T(s') < T(s)$. In other words, at most b_{max} of the earliest facts are to be selected, whose registration time lies within the given interval. The name of the solution algorithm FSEL comes from *Fact SELection*.

Provided that the history KB_H is managed by a relational database engine (RDBMS), selecting facts is a simple operation, which can be effectively realized by SQL commands such as SELECT TOP N...FROM...ORDER BY, together with aggregation functions: MIN, MAX, AVERAGE. The computational complexity of FSEL is linear: $O(n)$, where n is the number of facts in KB_H.

[1] Some elements and preliminary versions of the algorithms were introduced in the previous works [Dudek and Zgrzywa, 2005, Dudek et al. 2005], while their thorough analysis (including computational complexity) was done in [Dudek, 2005].

Algorithm 1 FSEL: Choosing facts for analysis

Input: history KB_H; maximal number of facts $b_{max}(v_c)$, where $v_c \in KB_T$
Output: selected fact portion $KB_H(t_{sc}, t_{ec})$; updated $v_c \in KB_T$

1: $KB_H(t_{sc}, t_{ec}) := \varnothing$
2: **if** $|KB_H| \leqslant b_{max}$ **then**
3: $\quad KB_H(t_{sc}, t_{ec}) := KB_H$
4: **end if**
5: **if** $|KB_H| > b_{max}$ **then**
6: $\quad KB_H(t_{sc}, t_{ec}) := \{s_1, s_2, \ldots, s_n \in KB_H \colon n = b_{max}$
$\quad \wedge \forall i, j \in \{1, 2, \ldots, n\}.(i < j \Leftrightarrow T(s_i) < T(s_j)) \wedge \forall s \in KB_H.T(s_1) < T(s)\}$
7: **end if**
8: $b_c(v_c) := |KB_H(t_{sc}, t_{ec})|$
9: $t_{sc}(v_c) := MIN\{T(s) \colon s \in KB_H(t_{sc}, t_{ec})\}$
10: $t_{ec}(v_c) := MAX\{T(s) \colon s \in KB_H(t_{sc}, t_{ec})\}$
11: $t_{mc}(v_c) := \frac{1}{n}\sum_{s \in KB_H(t_{sc}, t_{ec})} T(s)$
12: $k_e(v_c) := K(s)$, where $T(s) = t_{ec}(v_c)$
13: **return** $KB_H(t_{sc}, t_{ec})$, updated $v_c \in KB_T$

3.2.2 Transforming the history schema

After history facts are selected, the APS procedure needs to transform them into a form with binary attributes, which would be acceptable for a data mining process. First we need a new attribute schema, which can be further filled with facts. This is done by the *Transform_History_Schema* procedure (step 7) of the APS cycle. Formally, the problem can be stated as follows. Given the history schema $S_H = \{K, T, U\}$ and the selected fact portion $KB_H(t_1, t_2)$, find a new schema $S^{\#} = \{K, T, U'\}$, such that $\forall A_i \in U'.\ D_i = \{0, 1, N\}$. The only non-binary attributes allowed in the new schema are special attributes: the key K and the timestamp T, which remain unchanged as compared to the previous history schema. As a solution we propose the following HTRANS (*History schema TRANSformation*) algorithm. Attributes ${A_i}^{(v)}$ of the new $S^{\#}$ schema are found based on proper domains of initial attributes A_i (both single and multi-valued) that is all distinct values (except for N), which occurred in the processed fact portion. Observe that transforming the history schema in every APS learning run potentially narrows the number of resulting binary attributes, as compared to building a huge, general schema suitable for all runs (thus possibly containing many redundant attributes for a single run). Moreover, in our approach different history attributes can be used during different periods of storing observations. The complexity of the HTRANS algorithm is $O(k^2mn)$, where n is the number of facts in the portion $KB_H(t_{sc}, t_{ec})$, m is the number of attributes in U, and k is average number of attribute's distinct values in that portion.

Algorithm 2 HTRANS: Transforming the history schema

Input: history schema $S_H = \{K, T, U\}$, selected fact portion $KB_H(t_{sc}, t_{ec})$
Output: the new schema $S^{\#}$

1: $S^{\#} := \varnothing$
2: $S^{\#} := S^{\#} \cup \{K, T\}$
3: **for all** $A_i \in U$, where $i \in [1; |U|]$ **do**
4: find $D_i^W(KB_H(t_{sc}, t_{ec}))$.
5: $S^{\#} := S^{\#} \cup \{A_i^{(v)}\colon A_i \in S_H \wedge v \in D_i^W(KB_H(t_{sc}, t_{ec}))\}$.
6: **end for**
7: **return** $S^{\#}$

3.2.3 Populating the new schema

We transform on the fly the facts from the initial portion $KB_H(t_{sc}, t_{sc})$ into the new $S^{\#}$ schema, using the following HFILL algorithm (*History FILLing*). The resulting portion $KB_H^{\#}(t_{sc}, t_{ec})$ consists of the same number of facts as $KB_H(t_{sc}, t_{sc})$. Single-valued attributes are processed through straightforward value comparison (steps 4–12), while for multi-valued ones set membership needs to be tested (steps 13–21). Assume that the portion $KB_H(t_{sc}, t_{ec})$ consists of n facts and the set U' of the schema $S^{\#} = \{K, T, U'\}$ contains m attributes, $n, m \in \mathbb{N}$. Then the HFILL algorithm has computational complexity of $O(mn)$.

3.2.4 Eliminating unknown values

The facts transformed to the schema with binary attributes may be still unacceptable for many association rule mining algorithms as long as they contain unknown values N. We propose the ENV algorithm (*Elimination of N Values*) to deal with this problem. Essentially, the idea is to replace a fact containing one or more N values, with all the possible scenarios without N values. We use here the *random worlds approach*, treating all the possibilities equally probable [Bacchus et al. 1996]. This solution, even though simplified, decreases complexity of N value removal process and still provides good reliability of reasoning in information rich environments, as shown in [Bacchus et al. 1996]. In order to ensure consistency of generated facts, a complex condition is used in step 4. It guarantees that only one binary attribute $A_i^{(v_j)} \in S^{\#}$ coming from a single-valued attribute $A_i \in S_H^S$ can have value 1, while for multi-valued attributes $A_w \in S_H^M$ there can be more than one derived binary attribute $A_w^{(v_j)} \in S^{\#}$ with value 1. If there are more than η_c unknown values in a fact, it is considered too little informative and deleted without replacement[2]. After all N values have

[2] At the same time, reducing the number of facts decreases computational complexity of ENV.

Algorithm 3 HFILL: Filling the new history schema

Input: selected fact portion $KB_H(t_{sc}; t_{ec})$; attribute sets $S_H^S, S_H^M \subseteq S_H \setminus \{K, T\}$, where $S_H^S \cap S_H^M \equiv \varnothing$; new history schema $S^{\#}$

Output: set of transformed facts $KB_H^{\#}(t_{sc}; t_{ec})$

1: $KB_H^{\#}(t_{sc}, t_{ec}) := \varnothing$
2: **for all** facts $s \in KB_H(t_{sc}, t_{ec})$ **do**
3: create a new fact s^* of a schema $S^{\#}$:
$s^* = \{(K, K(s)), (T, T(s)), (A_1^{(v_1)}, N), (A_1^{(v_2)}, N), \ldots, (A_1^{(v_k)}, N), \ldots, (A_n^{(v_l)}, N), (A_n^{(v_{l+1})}, N), \ldots, (A_n^{(v_m)}, N)\}$,
where $A_i^{(v_j)} \in S^{\#}$, $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, m\}$
4: **for all** $A_i^{(v_j)} \in S^{\#}$, such that $A_i \in S_H^S$, $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, m\}$ **do**
5: **if** $A_i(s) = v_j$ **then**
6: $A_i^{(v_j)}(s^*) := 1$
7: **else if** $A_i(s) \neq v_j \wedge A_i(s) \neq N$ **then**
8: $A_i^{(v_j)}(s^*) := 0$
9: **else if** $A_i(s) = N$ **then**
10: $A_i^{(v_j)}(s^*) := N$
11: **end if**
12: **end for**
13: **for all** $A_i^{(v_j)} \in S^{\#}$, such that $A_i \in S_H^M$, $i \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, m\}$ **do**
14: **if** $v_j \in A_i(s)$ **then**
15: $A_i^{(v_j)}(s^*) := 1$
16: **else if** $v_j \notin A_i(s) \cup \{N\}$ **then**
17: $A_i^{(v_j)}(s^*) := 0$
18: **else if** $A_i(s) \in \{N\}$ **then**
19: $A_i^{(vj)}(s^*) := N$
20: **end if**
21: **end for**
22: $KB_H^{\#}(t_{sc}, t_{ec}) := KB_H^{\#}(t_{sc}, t_{ec}) \cup \{s^*\}$
23: **end for**
24: **return** $KB_H^{\#}(t_{sc}, t_{ec})$

been removed, time parameters t_{sc} and t_{ec} of the v_c vector are adjusted to the new set of facts[3].

The ENV algorithm has high, non-polynomial complexity of $O(n2^m)$, pro-

[3] Either the first or the last fact could have been removed, if it contained too many N values.

Algorithm 4 ENV: Removing unknown values

Input: transformed fact portion $KB_H^{\#}(t_{sc}; t_{ec})$, vector $v_c \in KB_T$ including the maximal number of unknown values in a single fact $\eta_c(v_c)$; attribute sets $S_H^S, S_H^M \subseteq S_H \setminus \{K, T\}$, where $S_H^S \cap S_H^M \equiv \varnothing$; schema $S^{\#} = \{K, T, U'\}$

Output: fact portion $KB_H^{\#}(t_{sc}; t_{ec})$ without N values; updated vector $v_c \in KB_T$

1: **for all** facts $s^* \in KB_H^{\#}(t_{sc}, t_{ec})$ **do**
2: $m := \left|\{A_i^{(v)} \in S^{\#} \colon A_i^{(v)}(s^*) = N\}\right|$
3: **if** $0 < m \leqslant \eta_c$ **then**
4: $KB_H^{\#}(t_{sc}, t_{ec}) := KB_H^{\#}(t_{sc}, t_{ec}) \cup \{s' =$
$\{(K, K(s^*)), (T, T(s^*)), (A_1^{(v_1)}, a_1^{(v_1)}), (A_1^{(v_2)}, a_1^{(v_2)}), \ldots, (A_1^{(v_k)}, a_1^{(v_k)}),$
$\ldots, (A_n^{(v_l)}, a_n^{(v_l)}), (A_n^{(v_{l+1})}, a_n^{(v_{l+1})}), \ldots, (A_n^{(v_p)}, a_n^{(v_p)})\} \colon A_i^{(v_j)} \in S^{\#}$
$\wedge\, i \in \{1, 2, \ldots, n\} \wedge j \in \{1, 2, \ldots, p\}$
$\wedge\, \forall A_i^{(v_j)} \in S^{\#}$, where $A_i \in S_H^M$.
$(A_i^{(v_j)}(s') \in \{0, 1\}) \wedge (A_i^{(v_j)}(s^*) \neq N \Leftrightarrow A_i^{(v_j)}(s^*) = A_i^{(v_j)}(s')))$
$\wedge\, \forall A_i^{(v_j)} \in S^{\#}$, where $A_i \in S_H^S$.
$(A_i^{(v_j)}(s') \in \{0, 1\}) \wedge (A_i^{(v_j)}(s^*) \neq N \Leftrightarrow A_i^{(v_j)}(s^*) = A_i^{(v_j)}(s'))$
$\wedge\, \forall A_i \in S_H^S. \left|\{A_i^{(v_j)} \in S^{\#} \colon A_i^{(v_j)}(s') = 1\}\right| = 1))\}$
5: **end if**
6: **if** $m > 0$ **then**
7: $KB_H^{\#}(t_{sc}; t_{ec}) := KB_H^{\#}(t_{sc}; t_{ec}) \setminus \{s^*\}$
8: **end if**
9: **end for**
10: $t_{sc}(v_c) := MIN\{T(s') \colon s' \in KB_H^{\#}(t_{sc}; t_{ec})\}$
11: $t_{ec}(v_c) := MAX\{T(s') \colon s' \in KB_H^{\#}(t_{sc}; t_{ec})\}$
12: **return** $KB_H^{\#}(t_{sc}; t_{ec})$, v_c

vided that $KB_H^{\#}(t_{sc}, t_{ec})$ contain n facts, k of them have at least one unknown value, and let the maximal number of N values in a single fact be m, where $k, m, n \in \mathbb{N}$ and $k \leqslant n$. Hence, this algorithm should be used carefully and parameterized with possibly low η_c threshold values.

3.2.5 Association rules discovery

After the history facts have been preprocessed, they can be analyzed by the rule mining procedure, using an algorithm chosen by the system developer (e.g. Apriori). The association discovery alone follows classical problem statement presented in other works [Agrawal et al. 1993, Agrawal et al. 1994, Goethals, 2002], but the novel solution is to place a rule mining algorithm inside the ARM algo-

rithm (*Association Rule Mining*), which plays a role of an interface – making use of the extended rule model and providing interoperation of the mining process with the APS architecture. There are two cases considered in the ARM algorithm. The first path (steps 2–3) is chosen, when the incorporated rule mining algorithm, apart from standard thresholds of minimal support σ_c and confidence γ_c, is able to handle additional constraints concerning the rule antecedent and consequent: X_c, Y_c, m_x and m_y (see explanation in the Definition 9). Then the mining algorithm takes all the constraints as input and yields a set of newly found rules R. In the second path (steps 4–6) the rule mining algorithm can accept solely σ_c and γ_c, so the resulting rule set R needs to be filtered outside this algorithm (step 6), using constraints X_c, Y_c, m_x and m_y. By handling both cases we achieve higher flexibility, being able to employ in the APS process a wide class of existing association mining algorithms. Afterwards, all the recent rules in R are assigned the same base numbers $b(r)$ and timestamps $t_m(r)$.

Algorithm 5 ARM: Association rules discovery

Input: preprocessed fact portion $KB_H^{\#}(t_{sc}; t_{ec})$, vector $v_c \in KB_T$, including constraints $X_c(v_c)$, $Y_c(v_c)$, $\sigma_c(v_c)$, $\gamma_c(v_c)$, $m_x(v_c)$, $m_y(v_c)$, $b_c(v_c)$, $t_{mc}(v_c)$
Output: set of new rules R

1: $R := \varnothing$
2: **if** the rule mining algorithm accepts constraints concerning the rule antecedent X and consequent Y **then**
3: $\quad R := Rule\_Mining\_Algorithm(\sigma_c, \gamma_c, X_c, Y_c, m_x, m_y)$
4: **else if** the rule mining algorithm does not accept constraints concerning X and Y **then**
5: $\quad R := Rule\_Mining\_Algorithm(\sigma_c, \gamma_c)$
6: $\quad R := \{r \in R\colon X(r) \subseteq X_c \wedge Y(r) \subseteq Y_c \wedge |X(r)| \leqslant m_x \wedge |Y(r)| \leqslant m_y\}$
7: **end if**
8: **if** $\neg R \equiv \varnothing$ **then**
9: $\quad$ **for all** rules $r_i \in R$, $i \in [1; |R|]$ **do**
10: $\quad\quad b(r_i) := b_c(v_c)$
11: $\quad\quad t_m(r_i) := t_{mc}(v_c)$
12: $\quad$ **end for**
13: **end if**
14: **return** R

Computational complexity of the ARM algorithm is strictly dependent on the encapsulated *Rule_Mining_Algorithm*. Hence, it can be considered beneficial that we do not force any specific algorithm, but allow a developer choose the

best one available. Additional workload of the steps 6 and 9–11 does not seem to be a serious problem, as it gives linear complexity of $O(n)$, where n is the number of rules $r \in R$.

3.2.6 Rule base maintenance

Updating the rule base KB_R, when old rules are combined with new ones R, is truly the crucial phase of the whole APS cycle. This maintenance is to keep the rule base in such a shape, as if all the rules were discovered in a single batch of history facts, accumulated since the very beginning of APS learning, even though actually only a recent portion of observations is processed in every run and analyzed past events are disposed. Formally, given two rule sets KB_R and R found in sets of facts h_1 and h_2, where $h_1 \cap h_2 \equiv \varnothing \wedge \forall s_i \in h_1, \forall s_j \in h_2 . T(s_i) < T(s_j)$, the new, combined rule set KB'_R is to be found, such that $KB'_R \cong KB^S_R$, where KB^S_R is a rule set found in $h_1 \cup h_2$.

As a solution to the stated problem we introduce the RMAIN algorithm[4] (*Rule MAINtenance*). Its key idea is to find pairs of rules – one from KB_R and the other one from R – with similar antecedents and consequents, and combine their support, confidence and other parameters, using appropriate proportion formulae (see the RMAIN algorithm). If a given rule does not exist in one of the combined sets, we use estimators of expected support $\hat{\sigma}$ and confidence $\hat{\gamma}$, which substitute corresponding values of a hypothetical, rejected rule (because of too low support or confidence), in order to decrease computation error [Dudek, 2005]. Moreover, we use the time depreciation function $f_T \colon [0; +\infty) \to [0; 1]$ (see Definition 10 in the section 3.1) in order to promote recent rules and deprecate old ones. Otherwise, in subsequent learning runs the old rules would be more and more resistant to any changes.

In the algorithm three cases of rule comparison are considered: (i) stable rules, which are present both in R and KB_R (steps 2–7); (ii) new rules, which belong to R, but are not found in KB_R (steps 2, 8–12); (iii) old rules, which belong solely to KB_R and are not present in R (steps 15–20). In each case rule's support, confidence, base number and time are updated, using proportion formulae. However, in cases (ii) and (iii) the unknown values of rule parameters are approximated by the estimators. After parameter update, new rules from R can be added to KB_R, if only they satisfy minimum support and confidence thresholds. At the same time old rules are withdrawn from KB_R, if their updated support or confidence do not meet the threshold requirements. In order to simplify the formulae notation inside the RMAIN algorithm, we use the following abbreviations: $f_{p_j} = f_t(t_{now} - t_m(p_j))$, $f_{r_i} = f_t(t_{now} - t_m(r_i))$, $f_g = f_t(t_{now} - t_{mg})$, $f_c = f_t(t_{now} - t_{mc})$.

[4] A preliminary version of the RMAIN algorithm was introduced in the previous work [Dudek and Zgrzywa, 2005].

Algorithm 6 RMAIN: Rule base maintenance

Input: KB_R, R; vectors $v_c \in KB_T$ and $v_g \in KB_G$; current system time t_{now}

Output: updated KB_R and $v_g \in KB_G$

update v_g: $\sigma(v_g) := \frac{\sigma(v_g)b(v_g)+\sigma(v_c)b(v_c)}{b(v_g)+b(v_c)}$, $\gamma(v_g) := \frac{\gamma(v_g)b(v_g)+\gamma(v_c)b(v_c)}{b(v_g)+b(v_c)}$
for all rules $r_i \in R, i \in [1; |R|]$ **do**
  if $\exists p_j \in KB_R.\, p_j \equiv r_i$ **then**
    update p_j: $sup(p_j) := \frac{b(p_j)sup(p_j)f_{p_j}+b(r_i)sup(r_i)f_{r_i}}{b(p_j)+b(r_i)}$,
    $con(p_j) := \frac{con(p_j)con(r_i)\left(b(p_j)sup(p_j)f_{p_j})+b(r_i)sup(r_i)f_{r_i}\right)}{b(p_j)sup(p_j)f_{p_j}con(r_i)+b(r_i)sup(r_i)f_{r_i}con(p_j)}$,
    $t_m(p_j) := \frac{b(p_j)t_m(p_j)+b(r_i)t_m(r_i)}{b(p_j)+b(r_i)}$, $b(p_j) := b(p_j) + b(r_i)$.
    if after the update $sup(p_j) < \sigma(v_g)$ or $con(p_j) < \gamma(v_g)$ **then**
      $KB_R := KB_R \backslash \{p_j\}$
    end if
  else if $\neg\exists p_j \in KB_R.\, p_j \equiv r_i$ **then**
    update r_i: $sup(r_i) := \frac{b(v_g)\hat{\sigma}(v_g)f_g+b(r_i)sup(r_i)f_{r_i}}{b(v_g)+b(r_i)}$,
    $con(r_i) := \frac{\hat{\gamma}(v_g)con(r_i)\left(b(v_g)\hat{\sigma}(v_g)f_g+b(r_i)sup(r_i)f_{r_i}\right)}{b(v_g)\hat{\sigma}(v_g)f_g con(r_i)+b(r_i)sup(r_i)f_{r_i}\hat{\gamma}(v_g)}$, if $0 < \hat{\gamma}(v_g) \leqslant 1$,
    $con(r_i) := con(r_i)$, if $\hat{\gamma}(v_g) = 0$,
    $t_m(r_i) := \frac{b(v_g)t_m(v_g)+b(r_i)t_m(r_i)}{b(v_g)+b(r_i)}$, $b(r_i) := b(v_g) + b(r_i)$
    if after the update $sup(r_i) \geqslant \sigma(v_g)$ and $con(r_i) \geqslant \gamma(v_g)$ **then**
      $KB_R := KB_R \cup \{r_i\}$
    end if
  end if
end for
for all rules $p_j \in KB_R$, such that $\neg\exists r_i \in R.p_j \equiv r_i$ **do**
  update p_j: $sup(p_j) := \frac{b(p_j)sup(p_j)f_{p_j}+b(v_c)\hat{\sigma}(v_c)f_c}{b(p_j)+b(v_c)}$,
  $con(p_j) := \frac{con(p_j)\hat{\gamma}(v_c)\left(b(p_j)sup(p_j)f_{p_j}+b(v_c)\hat{\sigma}(v_c)f_c\right)}{b(p_j)sup(p_j)f_{p_j}\hat{\gamma}(v_c)+b(v_c)\hat{\sigma}(v_c)f_c con(p_j)}$, if $0 < \hat{\gamma}(v_c) \leqslant 1$,
  $con(p_j) := con(p_j)$, if $\hat{\gamma}(v_c) = 0$,
  $t_m(p_j) := \frac{b(p_j)t_m(p_j)+b(v_c)t_m(v_c)}{b(p_j)+b(v_c)}$, $b(p_j) := b(p_j) + b(v_c)$
  if after the update $sup(p_j) < \sigma(v_g)$ or $con(p_j) < \gamma(v_g)$ **then**
    $KB_R := KB_R \backslash \{p_j\}$
  end if
end for
update the vector v_g: $\hat{\sigma}(v_g) := \frac{\hat{\sigma}(v_g)b(v_g)+\hat{\sigma}(v_c)b(v_c)}{b(v_g)+b(v_c)}$,
$\hat{\gamma}(v_g) := \frac{\hat{\gamma}(v_g)b(v_g)+\hat{\gamma}(v_c)b(v_c)}{b(v_g)+b(v_c)}$, $t_m(v_g) := \frac{b(v_g)t_m(v_g)+b(v_c)t_m(v_c)}{b(v_g)+b(v_c)}$
update the vector v_g: $b(v_g) := b(v_g) + b(v_c)$
return updated KB_R and v_g

HKey	*HTime*	*Term*	*Eval*	*Stored*
1	2008/04/19 16:30	*base, disc, computer, ..., storage*	*relevant*	*yes*
2	2008/04/20 15:35	*architecture, development, ..., schema*	*relevant*	*no*
...	...	...	...	...
k	2008/05/21 17:38	*copy, mechanism, ..., parallel*	*N*	*yes*

Figure 3: History KB_H of an example web browsing assistant.

The complexity of RMAIN is polynomial $O(mn)$, where n is the number of new rules in R and m is the number of old rules in KB_R, $m, n \in \mathbb{N}$. Other properties of the RMAIN algorithm were analyzed in [Dudek, 2005].

4 Example: web browsing assistant

In order to illustrate how the APS learning cycle works, we present the beneath example, referring to the personal web browsing assistant agent scenario (described in the Introduction). The agent history KB_H is as table containing the following columns (see Figure 3): *HKey*, *HTime*, *Term*, *Eval*, and *Stored*, representing: unique key, observation moment, collection of frequent index terms in the page (multivalued column), user's evaluation (*relevant* or *irrelevant*), and information about saving a page (*yes* or *no*), respectively. N values are placed into cells with missing information (e.g a page stored without any evaluation).

The first step of the APS learning run is the FSEL algorithm, which selects a block of recent observations, stored in the history since the previous processing run. Then these facts are transformed by the HTRANS algorithm from the initial schema: $S_H = \{HKey, HTime, Term, Eval, Stored\}$ into the schema: $S^{\#} = \{HKey, HTime, Term^{(architecture)}, Term^{(base)}, \ldots, Term^{(server)}, Eval^{(relevant)}, Eval^{(irrelevant)}, Stored^{(yes)}, Stored^{(no)}\}$.

All the attributes of the new schema can accept only two values: 1 (positive assertion) and 0 (negative assertion), except for the first two special columns *HKey* and *HTime*, which are left unchanged. In the next step of the APS run, the new $S^{\#}$ schema is filled with history facts by the HFILL algorithm (see Figure 4). Following the APS cycle, N values are eliminated using the ENV algorithm. Each row with unknown values is removed and instead new facts are inserted into the history, which reflect all the possible worlds (Figure 4). However, a row containing more N values than a presumed threshold η_c, is just dropped without any replacement (this case is not shown in Figure 4).

The transformed and preprocessed history is now ready for association rule mining, managed by the ARM algorithm. Lets assume that after analyzing 500 recent KB_H facts, two new rules were found (the R set): $r_1 : term\_architecture \wedge$

HKey	*HTime*	*Term* (*architecture*)	*Term* (*base*)	...	*Eval* (*relevant*)	*Eval* (*irrelevant*)	*Stored* (*yes*)	*Stored* (*no*)
1	2008/04/19 16:30	0	1	...	1	0	1	0
2	2008/04/20 15:35	1	0	...	1	0	0	1
...	...	...	...	...	...	...	...	...
k	2008/05/21 17:38	0	1	...	N	N	1	0
k_1	2008/05/21 17:38	0	1	...	**1**	**0**	1	0
k_2	2008/05/21 17:38	0	1	...	**0**	**1**	1	0

Figure 4: The transformed history while removing N values (ENV algorithm).

$term\_schema \Rightarrow eval\_relevant$ ($sup = 0.15$; $con = 0.90$; $b = 500$; $t_m =$ 2008/05/05 17:04) and r_2: $term\_base \wedge term\_matrix \Rightarrow eval\_relevant$ ($sup = 0.28$; $con =$ 0.85; $b = 500$; $t_m =$ 2008/05/05 17:04).

The newly discovered rules r_1 and r_2 need to be added to the KB_R rule base, which has contained two rules since the previous APS learning run (both based on 3000 processed facts): p_1: $term\_architecture \wedge term\_schema \Rightarrow eval\_relevant$ ($sup = 0.12$; $con = 0.73$; $b = 3000$; $t_m =$ 2008/01/16 10:45) and p_2: $term\_copy \wedge term\_mechanism \Rightarrow stored\_yes$ ($sup = 0.21$; $con = 0.69$; $b = 3000$; $t_m =$ 2008/01/16 10:45). It appears that the rules r_1 and p_1 are syntactically equal (see Definition 7), i.e. they have the same antecedents and consequents and only values of their support and confidence are different. In the R rule set r_2 is a new rule, which has been just found in the recent learning run. On the other hand the rule p_2 in KB_R has not been discovered lately, so we can expect its support and confidence will be decreased during KB_R update.

In the next stage, both rule sets $R = \{r_1, r_2\}$ and $KB_R = \{p_1, p_2\}$ are combined with each other by the RMAIN rule maintenance algorithm. Three cases are considered while updating KB_R: (i) the rules in both sets, which are syntactically equal (r_1 and p_1 in our example) – steps 2–7 of the algorithm; (ii) the rules, which exist only in R (rule r_2) – steps 2, 8–12, and (iii) the rules, which exist only in KB_R and were not discovered during the recent mining run (rule p_2) – steps 15–20. For RMAIN calculations we assume the following input parameters (see Definitions 9 and 11). The vector of current run parameters v_c: $X_c = \{Term\}$; $Y_c = \{Eval, Stored\}$; $b_{max} = 500$; $b_c = 500$; $\eta_c = 0$; $\sigma_c = 0.06$; $\gamma_c = 0.70$; $\hat{\sigma}_c = 0.03$; $\hat{\gamma}_c = 0.35$; $m_x = 4$; $m_y = 1$; $t_{rc} = t_{now} =$ 2008/05/21 18:05; $t_{mc} =$ 2008/05/05 17:04; $t_{sc} =$ 2008/04/19 16:30; $t_{ec} =$ 2008/05/21 17:38. The vector of global parameters v_g: $b_g = 3000$; $\sigma_g = 0.10$; $\gamma_g = 0.60$; $\hat{\sigma}_g = 0.05$; $\hat{\gamma}_g = 0.30$; $t_{mg} =$ 2008/01/16 10:45; $t_{sg} =$ 2007/10/15 7:24; $t_{eg} =$ 2008/04/19 14:07; $f_T = 1$. Then, after maintenance the KB_R will contain the following three rules: p_1: $term\_architecture \wedge term\_schema \Rightarrow eval\_relevant$ ($sup = 0.12$; con $= 0.75$; $b = 3500$; $t_m =$ 2008/02/01 01:22), p_2: $term\_copy \wedge term\_mechanism \Rightarrow$

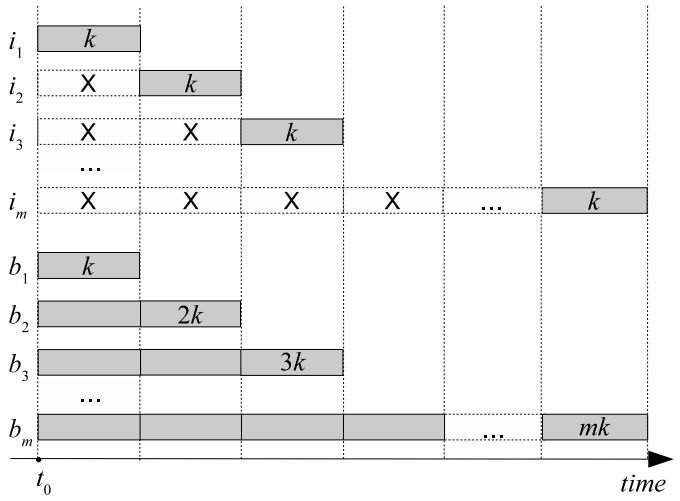

Figure 5: APS processing (blocks $i_1, \ldots, i_m$) and batch mining ($b_1, \ldots, b_m$).

stored_yes (*sup* = 0.18; *con* = 0.67; b = 3500; t_m = 2008/02/01 01:22), and r_2: *term_base* $\wedge$ *term_matrix* $\Rightarrow$ *eval_relevant* (*sup* = 0.08; *con* = 0.44; b = 3500; t_m = 2008/02/01 01:22). The new values of the v_g vector will be: b_g = 3500; $\sigma_g = 0.09$; $\gamma_g = 0.61$; $\hat{\sigma}_g = 0.05$; $\hat{\gamma}_g = 0.31$; t_{mg} = 2008/02/01 01:22. At the end of the learning run all the analyzed KB_H facts are permanently removed from the history and the agent switches over from learning to performance mode.

5 Experimental results

The APS method was evaluated experimentally with respect to both qualitative and performance measures. For the experiments we used a synthetic T10.I5.D20K dataset, corresponding to the WWW browsing agent scenario, which was generated with the *DataGen* application [Dudek, 2005]. The dataset has the following parameters [Agrawal et al. 1994]: number of transactions $|D|$ = 20000; mean transaction size $|T|$ = 10; average size of maximal, potentially frequent itemsets $|I|$ = 5; number of binary attributes 307. No facts inside the test dataset contain unknown values. While creating the data, the facts supporting rules (e.g. *adaptive* $\wedge$ *autonomous* $\wedge$ *learning* $\Rightarrow$ *relevant* $\wedge$ *stored_yes*) were distributed uniformly within a period from 1 January 2004, 00:27:45 until 10 April 2005, 23:32:57 as an activity log of a hypothetical Internet user, who visited 20,000 pages in 465 days (about 15 months), which makes 43 pages daily on average. Experiments were conducted in two series in order to compare incremental and batch processing (see Figure 5). During every test run processing

time of particular APS steps (algorithms) was registered and after the run had been completed, the resulting rule base KB_R was stored. Then we compared the rules discovered in the incremental and batch series, using the proposed qualitative measures: $rule_{overlap}$, sup_{diff} and con_{diff}, given below. The beneath *rule overlapping ratio* is the percent share of rules with the same antecedents and consequents in the compared rule sets (ignoring possible differences in rule support or confidence).

$$rule_{overlap}\left(R_1, R_2\right) = \frac{\left|KB_R^O\left(R_1, R_2\right)\right|}{|R_1| + |R_2| - \left|KB_R^O\left(R_1, R_2\right)\right|},$$

where $KB_R^O(R_1, R_2)$ is an intersection of two compared rule sets R_1 and R_2 with respect to syntactical equity of rules (see Definition 7). While $rule_{overlap}$ is used to evaluate general accuracy of incremental mining, deeper insight is provided by the next two measures sup_{diff} and con_{diff}, which show how the rules mined incrementally differ from rules in the reference batch-mined set, regarding support and confidence.

$$sup_{diff}\left(R_1, R_2\right) = \frac{1}{n}\left(\sum_{i=1}^{c} |sup(p_i) - sup(r_i)| + d\right),$$

$$con_{diff}\left(R_1, R_2\right) = \frac{1}{n}\left(\sum_{i=1}^{c} |con(p_i) - con(r_i)| + d\right),$$

where n is the number of all rules in R_1 and R_2 with different antecedents and consequents, $c = KB_R^O(R_1, R_2)$, $d = n - c$, and $p_i \in R_1$ has the same antecedent and consequent as $r_i \in R_2$ for $i \in \{1, \ldots, c\}$.

As the experimental testbed we used the *APS Incremental Learning* application [Dudek et al. 2005, Dudek, 2005] running on an x86 machine with 892.50 MHz CPU, 752 MB SDRAM, 40 GB HDD, NTFS, under OS MS Windows 2000 Server. The program was implemented using MS Visual C++ .NET language and the database management system MS SQL Server 2000 Enterprise Edition. Inside the testbed environment we used the University of Helsinki implementation of the Apriori algorithm [Agrawal et al. 1994], developed by [Goethals, 2003], for mining association rules. The experiments were conducted with the following settings: $k = 1000$; $\sigma = 0.08$; $\gamma = 0.30$; $\hat{\sigma} = 0.04$ and $\hat{\gamma} = 0.15$.

The qualitative results proved perfect accuracy of the APS incremental rule discovery as compared to batch mining. The $rule_{overlap}$ measure attained 100%, whereas sup_{diff} and con_{diff} were 0% for all examined runs. While this is what we can expect from incremental processing of the synthetic, uniform dataset T10.I5.D20K, less regular data (in real applications) is potentially a challenge for the APS method and it is likely to yield less accurate results. Yet, stable

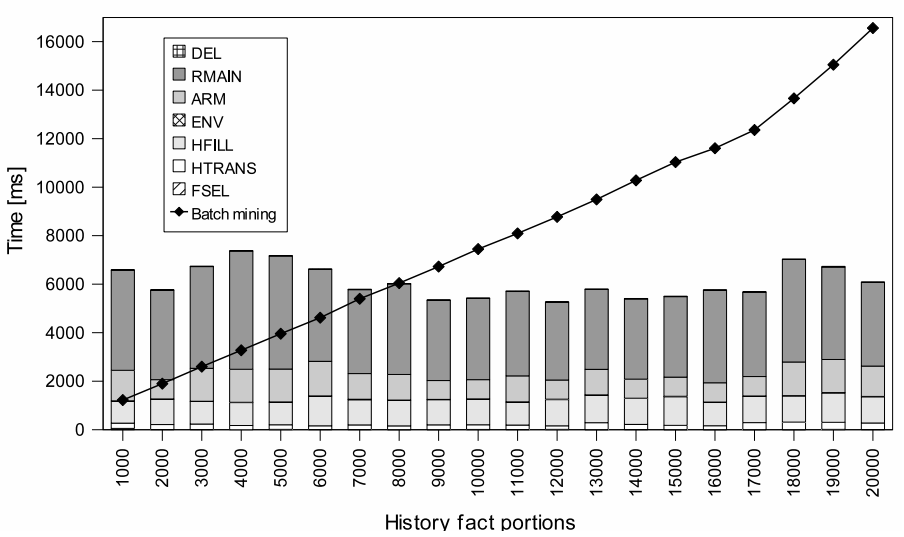

Figure 6: Processing time: particular APS algorithms vs. Apriori batch mining.

rules (i.e. frequent, confident and appearing regularly) should be mined and maintained correctly, regardless the overall irregularity of a data set.

In the performance evaluation we compared processing time of the APS incremental learning (separating values for particular APS algorithms) to the time of batch rule mining. Analyzing the results we can see the high 60% workload share of rule base maintenance (the RMAIN algorithm), while sole association mining makes only 17%, similarly to filling the history schema (the HFILL algorithm). Other APS processing stages, such as selecting (FSEL), transforming the history schema (HTRANS) and fact deletion together scarcely take about 6% of time. While fact selection and deletion are indeed straightforward database operations, which can be very efficiently performed by contemporary database management systems, the HTRANS and ENV algorithms are more complex. Hence, the very small processing time share of HTRANS and ENV, which was observed, can be misleading in general. The former algorithm would certainly need more time, if the initial history schema contained more attributes (in our experiments there were only about 300). The ENV algorithm is even more complex $O(n2^m)$, so it would work much longer for higher density of missing attribute values.

The overall APS processing time for subsequent fact blocks is quite stable, oscillating around the average value, while batch mining time grows roughly proportionally to the increasing number of analyzed facts (see Figure 6). The moment, when the APS outperforms batch mining (about 8000 for the T10.I5.D20K dataset), depends on application domain and analyzed data. But in general the

more APS learning runs are performed, the bigger profit in saving system resources (CPU, memory and disc space) is plausible.

6 Related work

There is hardly any research directly covering the contribution of this paper, but there are many works related to some parts and aspects of APS.

Firstly, there is a line of research exploring usage of data mining methods inside agent systems, including the *Agent Academy* (AA) by Symeonidis and his coworkers [Symeonidis et al. 2002]. AA uses data mining algorithms (such as ID3, C4.5 and Apriori) for training agents, based on a common knowledge repository. As compared to their system, the APS framework is an internal learning procedure of an autonomous agent and not an external training process, outside an agent.

Secondly, there are many works on web recommendation systems using various data mining techniques, such as: clustering, association rule mining, classifiers and decision trees. Yao, Hamilton and Wang [Yao et al. 2002] proposed the *PagePrompter* system, supporting web site administering by means of association rules and clusters, which are discovered through web log mining. Wang and Shao worked out a personalized recommendation method for e-learning students, based on analysis of past navigation sessions using clustering and association rules [Wang and Shao, 2004]. Within the same application domain, Chen, Hsieh and Hsu [Chen et al. 2007] used the Apriori algorithm inside the agent-based, personalized e-learning system (PELS) for mining association rules, which build the human learner's profile and help to identify common misconceptions during the learning process. An interesting comparative study of sequential association rules, based on web log mining and used for building web document prediction models was done by Yang, Li and Wang [Yang et al. 2004]. In contrast to the above works, the proposed APS method is not restricted to the web recommendation domain (actually, it is only an example application), but it is a general learning framework for software agents.

Finally, many alternative methods of incremental association mining have been developed to date: FUP [Cheung et al. 1996], FUP_2 [Cheung et al. 1997], Borders [Aumann et al. 1999], MAAP [Zhou and Ezeife, 2001], SWF [Lee et al. 2005] and EDUA [Zhang et al. 2007]. However, they differ from the APS approach: (i) they use intermediate representation – frequent itemsets, while the APS operates on final rules, which are much more suitable for agent reasoning, (ii) they usually reduce, but do not eliminate reruns through analyzed data, while our method provides approximate results without any database rescanning (which allows reducing disc space usage); (iii) they mostly do not use any time decaying heuristics, whereas we proposed the f_T time depreciation function

in the APS. Data stream mining research uses concepts, which are very closely related to the idea of partial memory learning [Maloof and Michalski, 2004], developed in the APS. A good example is the *estDec* method of stream mining [Chang and Lee, 2003]. However, as compared to *estDec*, the APS method allows using a broader range of time decaying functions (not restricted to exponential decay) and it covers management of the whole data mining process, not only rule discovery and maintenance.

7 Conclusions

In this paper a novel APS method was proposed for incremental, statistical learning of software agents. The original contribution of this work is to incorporate into an agent architecture a complete data mining process, including data accumulation, selection and preprocessing, association rule mining, rule base maintenance and history flushing. The history transaction schema is transformed in every learning run, making the proposed method highly flexible, because different history attributes are allowed to appear in subsequent runs. The crucial part of the APS method is the RMAIN rule maintenance algorithm, which incrementally updates the agent's rule base. RMAIN uses a time decaying function and it operates on rules, unlike most other incremental mining methods, which process only intermediate representation – frequent itemsets. Operating on rule level extends potential applications of APS beyond single-agent learning and enables fusing knowledge coming from the agent's internal learning process with rules shared by other agents. As the analyzed history facts are disposed in every run of the learning cycle, in long-term perspective disk space and CPU time usage is reduced as compared to batch mining approach. This makes APS suitable for resource-bounded agents, which are supposed to work as tiny background processes, not disturbing user's normal activity. The proposed learning method can be also beneficial, if an agent interlaces standard performance with stand-by periods, when it does not work toward any goal, but awaits new tasks from a user. Personal agents in the domain of web browsing, information retrieval and filtering are good examples of possible applications.

We verified experimentally the APS method using a uniform, synthetic web browsing scenario dataset. The tests proved perfect accuracy of the incrementally mined rules as compared to the reference rule sets, which were discovered in a batch mode. Based on these results we can conclude that the APS framework ensures good quality of incremental association rule mining and maintenance, provided that the fact data source is stable and does not change rapidly between subsequent learning runs. On the other hand, dynamic environments are expected to challenge the rule maintenance procedure, which can deteriorate qualitative results.

The APS framework is intended to be implemented as a learning module and integrated with existing agent architectures, which is an interesting issue for further research.

Acknowledgments

The author would like to thank: Bart Goethals for sharing his implementation of the Apriori algorithm, Michal Kubisz for help with development of the testbed program, prof. Aleksander Zgrzywa for guidance and valuable suggestions, Christian Schenk, Henrik Just, Adam Skorczynski and Sebastian Deorowicz for contributing great LaTeX software, which was used for editing the final version of the paper (*MiKTeX*, *writer2LaTeX*, and *LEd*, respectively). The author also wishes to kindly appreciate the support of this research provided by the Wroclaw University of Technology.

References

[Agrawal et al. 1993] Agrawal, R., Imielinski, T., Swami, A.: "Mining association rules between sets of items in large databases"; In Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'93) (1993) 207–216

[Agrawal et al. 1994] Agrawal, R., Srikant, R.: "Fast Algorithms for Mining Association Rules"; In Proceedings of the Twentieth International Conference on Very Large Databases, Santiago, Chile (1994)

[Aumann et al. 1999] Aumann, Y., Feldman, R., Lipshtat, O., Manilla, H.: "Borders: An Efficient Algorithm for Association Generation in Dynamic Databases"; Journal of Intelligent Information Systems, 21 (1999) 61–73

[Bacchus et al. 1996] Bacchus, F., Grove, A., Halpern, J., Koller, D.: "From statistical knowledge bases to degrees of belief"; Artificial Intelligence, 87, 1–2 (1996) 75–143

[Chang and Lee, 2003] Chang, J.H., Lee, W.S.: "Finding Recent Frequent Itemsets Adaptively over Online Data Streams"; In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (2003) 487–492

[Chen et al. 2007] Chen, C.-M., Hsieh, Y.-L., Hsu, S.-H.: "Mining learner profile utilizing association rule for web-based learning diagnosis"; Expert Systems with Applications, 33 (2007) 6–22

[Cheung et al. 1996] Cheung, D.W., Ng, V.T., Tam, B.W.: "Maintenance of Discovered Knowledge: A Case in Multi-level Association Rules"; In Proc. of the Second International Conference on Knowledge Discovery and Data Mining, (1996) 307–310

[Cheung et al. 1997] Cheung, D.W., Lee, S.D., Kao, B.: "A general incremental technique for maintaining discovered association rules"; In Proceedings of the Fifth International Conference on Database Systems for Advanced Applications, Melbourne, Australia (1997) 185–194

[Dudek and Zgrzywa, 2005] Dudek, D., Zgrzywa, A.: "The Incremental Method for Discovery of Association Rules"; In Kurzynski, M., Puchala, E., Wozniak, M., Zolnierek, A. (eds.): Proceedings of the Fourth International Conference on Computer Recognition Systems (CORES'05), Advances in Soft Computing, Springer-Verlag, Berlin Heidelberg (2005) 153–160

[Dudek et al. 2005] Dudek, D., Kubisz, M., Zgrzywa, A.: "APS: Agent's Learning With Imperfect Recall"; In Kwasnicka, H., Paprzycki, M. (eds.): Proceedings of the Fifth International Conference on Intelligent Systems Design and Applications, IEEE Computer Society Press, Washington, Brussels, Tokyo (2005) 172–177

[Dudek, 2005] Dudek, D.: "Knowledge Acquisition Within an Agent System Using Data Mining Methods"; PhD Thesis, Technical Report No. 2, Institute of Applied Informatics, Wroclaw University of Technology, Wroclaw (2005) (in Polish)

[Dudek, 2007] Dudek, D.: "Using Data Mining Algorithms for Statistical Learning of a Software Agent"; In Nguyen, N.T., Grzech, A., Howlett R.J., Jain L.C. (eds.): Proceedings of the First KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications (KES-AMSTA 2007), Lecture Notes in Artificial Intelligence (LNAI) 4496, Springer-Verlag, Berlin Heidelberg (2007) 111–120

[Fagin et al. 1995] Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: "Reasoning About Knowledge"; MIT Press, Cambridge, MA, USA (1995)

[Goethals, 2002] Goethals, B.: "Efficient Frequent Pattern Mining"; PhD thesis, Transnational University of Limburg, Diepenbeek, Belgium (2002)

[Goethals, 2003] Goethals, B.: "Implementation of the Apriori Algorithm", `http://www.adrem.ua.ac.be/~goethals/`, University of Helsinki (2003)

[Laird et al. 1987] Laird, J., Newell, A., Rosenbloom, P.: "Soar: An Architecture for General Intelligence"; Artificial Intelligence, 33 (1987) 1–64

[Lee et al. 2005] Lee, C.-H., Lin, C.-R., Chen, M.-S.: "Sliding window filtering: an efficient method for incremental mining on a time-variant database"; Information systems, 30 (2005) 227–244

[Maes, 1994] Maes, P.: "Modeling Adaptive Autonomous Agents"; In Artificial Life: An Overview. MIT Press, Cambridge, MA, USA (1994) 135–162

[Maloof and Michalski, 2004] Maloof, M.A., Michalski, R.S.: "Incremental learning with partial instance memory"; Artificial Intelligence, 154 (2004) 95–126

[Mannila, 1997] Mannila, H.: "Methods and problems in data mining. A tutorial"; In Afrati, F., Kolaitis, P. (eds.): Proceedings of the International Conference on Database Theory (ICDT'97), Delphi, Greece (1997) 41–55

[Newell, 1990] Newell, A.: "Unified Theories of Cognition"; Harvard University Press, Cambridge (1990)

[Pankowski, 1992] Pankowski, T.: "Foundations of Databases"; Polish Scientific Publishers PWN, Warsaw (1992) (in Polish)

[Ribeiro, 2002] Ribeiro, C.: "Reinforcement Learning Agents"; Artificial Intelligence Review, 17 (2002) 223–250

[Sen and Weiss, 1999] Sen, S., Weiss, G.: "Learning in Multiagent Systems"; In: Weiss, G. (ed.), Multiagent systems, The MIT Press (1999) Chapter 6, 259–298

[Symeonidis et al. 2002] Symeonidis, A.L., Mitkas, P.A., Kechagias, D.D.: "Mining Patterns And Rules For Improving Agent Intelligence Through An Integrated Multi-Agent Platform"; In Proc. of the Sixth IASTED International Conference on Artificial Intelligence and Soft Computing (ASC 2002), Banff, Alberta, Canada (2002)

[Wang and Shao, 2004] Wang, F.-H., Shao, H.-M.: "Effective personalized recommendation based on time-framed navigation clustering and association mining"; Expert Systems with Applications, 27 (2004) 365–377

[Yang et al. 2004] Yang, Q., Li, T., Wang, K.: "Building Association-Rule Based Sequential Classifiers for Web-Document Prediction"; Data Mining and Knowledge Discovery, 8 (2004) 253–273

[Yao et al. 2002] Yao, Y.Y., Hamilton, H.J., Wang, X.: "PagePrompter: An Intelligent Web Agent Created Using Data Mining Techniques"; In Alpigini, J.J. et al. (eds.), RSCTC 2002, Lecture Notes in Artificial Intelligence, 2475, Springer-Verlag, Berlin Heidelberg (2002) 506–513

[Zhang et al. 2007] Zhang, S., Zhang, J., Zhang, C.: "EDUA: An efficient algorithm for dynamic database mining"; Information Sciences, 177 (2007) 2756–2767

[Zhou and Ezeife, 2001] Zhou, Z., Ezeife, C.I.: "A Low-Scan Incremental Association Rule Maintenance Method Based on the Apriori Property"; In Stroulia, E., Matwin, S. (eds.): AI 2001, Lecture Notes in Artificial Intelligence, 2056. Springer-Verlag, Berlin Heidelberg (2001) 26–35