Available online at www.sciencedirect.com

ScienceDirect

Computers in Biology and Medicine 00 (2014) 1–34

Computers in Biology and Medicine

Mining Approximate Temporal Functional Dependencies with Pure Temporal Grouping in Clinical Databases

Carlo Combi[a], Matteo Mantovani[a], Alberto Sabaini[a], Pietro Sala[a], Francesco Amaddeo[b], Ugo Moretti[b], Giuseppe Pozzi[c]

[a]*Dipartimento di Informatica, Università degli Studi di Verona, strada le Grazie 15, I-37134 Verona, Italy, name.surname@univr.it*
[b]*Dipartimento di Sanità Pubblica e Medicina di Comunità, Università degli Studi di Verona, p.le L.A. Scuro 10, I-37134 Verona, Italy, name.surname@univr.it*
[c]*DEIB, Politecnico di Milano, p.za L. da Vinci 32, I-20133 Milano, Italy, giuseppe.pozzi@polimi.it*

Abstract

Functional dependencies (FDs) typically represent associations over facts stored by a database, such as "*patients with the same symptom get the same therapy*." In more recent years, some extensions have been introduced to represent both temporal constraints (temporal functional dependencies - TFDs), as "*for any given month, patients with the same symptom must have the same therapy, but their therapy may change from one month to the next one*," and approximate properties (approximate functional dependencies - AFDs), as "*patients with the same symptom* generally *have the same therapy*." An AFD holds most of the facts stored by the database, enabling some data to deviate from the defined property: the percentage of data which violate the given property is user-defined.

According to this scenario, in this paper we introduce approximate temporal functional dependencies (ATFDs) and use them to mine clinical data. Specifically, we considered the need for deriving new knowledge from psychiatric and pharmacovigilance data.

ATFDs may be defined and measured either on temporal granules (e.g., grouping data by day, week, month, year) or on sliding windows (e.g., a fixed-length time interval which moves over the time axis): to this regard, we propose and discuss some specific and efficient data mining techniques for ATFDs. We also developed two running prototypes and showed the feasibility of our proposal by mining two real-world clinical data sets. The clinical interest of the dependencies derived considering the psychiatry and pharmacovigilance domains confirms the soundness and the usefulness of the proposed techniques.

Keywords: Approximate temporal functional dependency, temporal granule, sliding window, grouping, psychiatric patients, pharmacovigilance

1. Introduction

Current clinical database systems enable us to store huger and huger quantities of data, and data mining techniques help in extracting relevant knowledge from these data. Analyzing temporal evolution of data, time series, changes of information over time, may lead to additional *temporal* knowledge. *Temporal* data mining is the research field in this direction, working on structured [1] and, occasionally, on semi-structured data [2].

Knowledge in (clinical) databases may be expressed in two ways: on one hand, it can be represented through suitable constraints on data; on the other hand, it can be derived through the analysis of data, by discovering patterns, regularities, and so on.

According to the first point of view and considering data stored in a plain relational database, we may express constraints by identifying functional dependencies (FD). Let us consider, for example, a simple database table describing the reference areas for emergency admissions in a region. We typically specify that patients have a single reference hospital for emergencies, depending on their address (considering the neighborhood of the admitting reference hospital). We can thus specify a functional data dependency between the home address of the patient and the location of the hospital: all patients with the same address must refer to the same hospital. Leveraging the definition of functional dependencies as a way of expressing constraints on data, the research community focused also on extending FDs to deal with data temporalities [3, 4, 5, 6, 7]: as example, a temporal functional dependency (TFD) may be used to express the constraint that the reference hospital for emergencies depends on the patient home address, but this dependency may change according to the season of the year.

On the other hand, a different approach has to be taken if we consider, for example, a database table collecting data on patients who were admitted for emergencies to hospitals. In this case, we cannot constrain patient addresses to hospitals in a strict way, but we could discover on the collected data that the dependency between patient addresses and hospitals hold on *most* tuples of the database, but not on *all* the tuples of that database. We call this an *approximate* functional dependency (AFD): patients with the same home address usually go the same hospital (not always the reference one) when they are at home, but, as an example, some patients on holiday could have been admitted to an hospital which is not the closest one to their home address. The issue of discovering approximate functional dependencies from data has been largely studied in the literature [8, 9, 10, 11].

As final consideration, we may also experience that over some periods of the year we generally observe an approximate functional dependency, while in some other periods we observe a different approximate dependency: for example, it could occur that patients go to different hospitals for emergencies even according to some specific skills of hospitals in managing seasonal pathologies. In this case, it still holds that we can discover approximate dependencies between patient addresses and hospitals for emergencies, but only if we group data according to the season and the year of the emergency admission. We call this an *approximate, temporal* functional dependency (ATFD). At the best of our knowledge, studies on *approximate temporal* functional dependencies still lack.

According to the depicted scenario, the aim of this paper is to propose a first step, focusing on a specific type of ATFD, of a general framework for temporal data mining of clinical data. In particular, we adopt a framework for temporal functional dependencies recently proposed by Combi et al. in [7]: the framework subsumes all the previous proposals dealing with temporal functional dependencies for relational databases and introduces some new kinds of temporal functional dependencies. According to this framework, we then focus on the issue of mining (approximate) temporal functional dependencies based on a temporal grouping of tuples. We introduce the concept of approximate temporal functional dependency with temporal grouping, and discuss through some examples both the case when grouping is induced by granularities (i.e. time units) and the case when sliding windows are used. Then, we propose efficient algorithms for this kind of temporal data mining. Finally, we discuss the application of our algorithms to real world clinical data from the psychiatric and pharmacovigilance domains, respectively.

Besides the technical performances, we discuss the clinical meaning and the most relevant mined temporal dependencies; to this regard, it worth noting that the mined temporal functional dependencies are a relatively new kind of clinical knowledge on data, which deserves further efforts to become clearly interpretable by physicians in a daily clinical setting. Indeed, while association rules and temporal association rules have been considered in clinical domains for years and their role in the clinical decision-support process has been widely acknowledged [12, 13], approximate temporal functional dependencies represent a new piece of knowledge that has to be properly integrated in clinical decision-support processes. As an example, temporal association rules may allow one to derive knowledge as “most patients presenting a symptom of chest pain overlapping nausea receive, within few days, a therapy with acetylsalicylic acid”. On the other side, approximate temporal functional dependencies provide knowledge at a higher abstraction level, as “in most cases, patients with the same symptoms are given the same drug (i.e. active principle), considering a time window of 10 days”. Such kind of knowledge refers to a general relationship between some features of a patient, in this case symptoms and therapies: the relationship holds for any specific values of such features. Such a valuable kind of knowledge requires physicians to merge it with more specific knowledge, such as that one coming from temporal association rules, in the whole decision making process.

The main novelty aspects of this paper can be summarized as in the following, even with a specific reference to the preliminary work in [14], where the main focus was on the proposal of ATFDs and on some preliminary experiments on a reduced set of psychiatry data with some first prototiypal algorithms.

- we discuss in detail the proposed approach for ATFDs and introduce completely new algorithms both for granularity-based temporal mining and for mining through sliding windows;
- we present and discuss two important clinical domains, i.e. psychiatry and pharmacovigilance, where temporal data mining is highly required. As the mined temporal dependencies are sometime completely new and unexpected even to expert physicians, we discuss here some possible interpretations of the discovered knowledge;
- the new experimental results, with a new and extended setting considering two different data sets from psychia-

try and pharmacovigilance, consist of both a detailed performance analysis and an evaluation and discussion of the mined ATFDs from a clinical point of view.

In the following, we describe the background and the related work (Section 2) and discuss the two clinical domains we considered for temporal data mining, namely psychiatry and pharmacovigilance (Section 3); we introduce the concept of approximate temporal functional dependency (ATFD), providing some examples on the application scenario (Section 4); then we describe how to mine minimal ATFDs (Section 5) and deploy the proposed techniques in clinical domains; we describe the experimental results obtained by considering data in the two mentioned domains (Section 6), and finally (Section 7) we draw some conclusions and sketch out some possible directions for future research.

2. Background and Related Work

We recall here the definition of functional dependency (FD), and then introduce its extensions: approximate functional dependency (AFD) and temporal functional dependency (TFD). Such concepts will lead to the definition of approximate temporal functional dependency (ATFD) of Section 4, where ATFD inherits the properties both from AFD and from TFD. The concept of functional dependency (FD) comes from the database theory and is defined as follows [15]:

Definition 2.1 (*Functional Dependency*). Let r be a relationship over the relational schema R: let $X, Y \subseteq R$ be attributes of R. We assert that r fulfills the functional dependency $X \rightarrow Y$ (written as $r \models X \rightarrow Y$) if the following condition holds: $\forall t, t' \in r(t[X] = t'[X] \Rightarrow t[Y] = t'[Y])$

Informally, for all the couples of tuples t and t' showing the same value(s) on X, the corresponding value(s) on Y for those tuples are identical.

2.1. Temporal Functional Dependencies

Moving closer to the main kind of temporal features we shall consider here, several kinds of temporal functional dependencies (TFDs) have been proposed in the literature, usually as temporal extensions of the widely know (atemporal) functional dependencies [16]. As an example, we may consider that patients affected by a common pathology p_1 may assume a common therapy t_1 during some month M_1, while in other month M_2 the same patients affected by the same pathology p_1 as above, do follow the another common therapy t_2.

Recently, Combi et al. proposed a framework for TFDs that subsumes and extends the considered previous proposals [7]. The proposed framework is based on a simple temporal relational data model based on the notion of *temporal relation*, i.e. a relation extended with a timestamping temporal attribute VT, representing the *valid time* temporal dimension, i.e. the time when the fact is true in the real world [17].

Two temporal views have been introduced: they allow one to join tuples that represent relevant cases of (temporal) evolution. On the base of the introduced data model, and leveraging the introduced temporal views, TFDs may be expressed by the syntax $[\textit{E-Exp}(R), \textit{t-Group}]X \rightarrow Y$ where $\textit{E-Exp}(R)$ is a relational expression on R, called *evolution expression*, *t-Group* is a mapping $\mathbb{N} \rightarrow 2^{\mathbb{N}}$, called *temporal grouping*, and $X \rightarrow Y$ is a functional dependency.

As for the semantics, similarly to the case of standard FDs, a TFD is a statement about admissible temporal relations on a temporal relation schema R with attributes $U \cup \{VT\}$. A temporal relation r on the temporal relation schema R satisfies a TFD $[\textit{E-Exp}(R), \textit{t-Group}]X \rightarrow Y$ if it is not possible that the relation obtained from r by applying the expression $\textit{E-Exp}(R)$ features two tuples t, t' such that (i) $t[X] = t'[X]$, (ii) $t[VT]$ and $t'[VT]$ (and the valid times of their evolutions, if present) belong to the same temporal group, according to the mapping *t-Group*, and (iii) $t[Y] \neq t'[Y]$. In other words, FD $X \rightarrow Y$ must be satisfied by each relation obtained from the evolution relation by selecting those tuples whose valid times belong to the same temporal group.

Temporal grouping enable us to group tuples together over a set of temporal granules, based on one temporal dimension. We focus here on the VT temporal dimension.

Four different classes of TFD have been identified in [7]:

- *Pure temporally grouping TFD*: $\textit{E-Exp}(R)$ returns the original temporal relation r. Rules of this class force the FD $X \rightarrow Y$, where $X, Y \subseteq U$, to hold over all the maximal sets which include all the tuples whose VT belongs to the same temporal grouping;

- *Pure temporally evolving TFD*: $\textit{E-Exp}(R)$ collects all the tuples modelling the evolution of an object. No temporal grouping exists: that is, the temporal grouping collects all the tuples of r in one unique set;

- *Temporally mixed TFD*: the expression $\textit{E-Exp}(R)$ collects all the tuples modelling the evolution of the object. The temporal grouping is applied to the set of tuples generated by $\textit{E-Exp}(R)$;

- *Temporally hybrid TFDs*. First, the evolution expression $\textit{E-Exp}(R)$ selects those tuples of the given temporal relation that contribute to the modeling of the evolution of a real-world object (that is, it removes isolated tuples); then, temporal grouping is applied to the resulting set of tuples.

In the remainder of the paper, we shall focus on pure temporally grouping TFDs, only.

2.2. Approximate Functional Dependencies

The concept of approximate functional dependency (AFD) derives from the concept of plain FD. Given a relation r where a FD holds for *most* of the tuples in r, we may identify *some* tuples, for which that FD does *not* hold. Consequently, we define some measurements over the error we make in considering the FD to hold on r. One measurement [8] is know as G_1 and considers the number of violating couples of tuples. Another measurement [8], known as G_2, considers the number of tuples which violate the functional dependency. The most common measurement [8], known as G_3, considers the minimum number of tuples in r to be *deleted* for the FD to hold. Formally, $G_3(X \to Y, r) = |r| - max\{|s| \mid s \subseteq r \wedge s \models X \to Y\}$

The related *scaled measurement* g_3 is defined as $g_3(X \to Y, r) = G_3(X \to Y, r)/|r|$

We can now introduce here the definition of approximate functional dependency AFD as:

Definition 2.2 (*Approximate Functional Dependency*)**.** Let r be a relation over the relational schema R: let $X, Y \subseteq R$ be attributes of R. Relation r fulfills an approximate functional dependency $X \overset{\varepsilon}{\to} Y$ (written as $r \models X \overset{\varepsilon}{\to} Y$) if $g_3(X \to Y, r) \leq \varepsilon$, where ε is the maximum acceptable error defined by the user.

Among the several AFDs that can be identified over a relation r, the minimal AFD is of particular interest, as many other AFDs can then be derived from the minimal one. We thus define the minimal AFD as follows:

Definition 2.3 (*Minimal AFD*)**.** Given an AFD over r, we define $X \overset{\varepsilon}{\to} Y$ to be minimal for r if $r \models X \overset{\varepsilon}{\to} Y$ and $\forall X' \subset X$ we have that $r \not\models X' \overset{\varepsilon}{\to} Y$.

3. Motivating Medical Domains

In order to motivate and validate our approach, we consider two clinical domains: the first one refers to *psychiatry*, collecting data about contacts between patients and psychiatrists, psychologists, and social workers; the second one refers to *pharmacovigilance*, collecting data about drug administrations and adverse reactions.

3.1. Psychiatric Case Register

The first application domain (see Section 3.3 for further examples) refers to the Verona Psychiatric Case Register (PCR). The Verona Health District serves about 460,000 inhabitants. The National Health Service in trust with the University of Verona offers a public Community-based Psychiatric Service (CPS), providing psychiatric care to mentally ill as well as psychological care and responses to social needs. Data about patients are collected in the information system PCR, which has recorded information about patients' accesses to this service since 1979. At the first contact with the psychiatric service, socio-demographic information, past psychiatric history, and clinical data are routinely collected for patients aged 14 and over. Recorded contacts with psychiatrists, psychologists, social workers and psychiatric nurses include home visits, telephone calls, day cares, and so on. Data on some 28,700 patients and more than 1,500,000 psychiatric contacts have been recorded. Besides patients' personal data (e.g., birth information, health insurance card number, gender, nationality, and previous contacts), patients' medical record, and contact information (contact duration, involved professionals, referrals, contact type, and conclusions), PCR also records education, employment, professional status, type of accommodation, and marital status of patients.

PCR is used as a basis to evaluate the direct management costs for groups of patients, and to monitor the effects coming from changes in resources, organization, and needs. The clinical purposes include monitoring of patients to plan future contacts at regular time intervals, and providing clinicians with reports about admissions and contacts for every patient in a given time period.

PCR stores several temporal data: a patient's contact is temporally qualified by its occurrence timestamp, while other personal information are qualified by their respective valid time. These temporal data can then be used by epidemiologists, e.g. to identify the number of contacts in different time periods with respect to different factors such as age, diagnosis.

3.2. Pharmacovigilance

Pharmacovigilance (PhV) collects, analyzes, and prevents adverse reactions induced by drugs (ADR) [18]. In fact, also because of the limitations of pre-marketing trials (e.g. short duration of the study, highly selected test population), adverse reactions often go undetected, and become evident when the drug is put on the market, only [19]. Therefore a continuous monitoring of the effects is needed.

The spontaneous reporting of ADRs identifies unexpected reactions and informs the regulating authority about them. This practice is valuable, provides early warnings, and requires limited economic and organizational resources [20]. It also has the advantage of covering every drug on the market and every category of patient.

PhV considers possible relationships between one or more adverse reactions and one or more drugs, mainly focusing on unknown or completely undocumented relationships. Reports suggest a cause-effect link among ADRs and drugs: the link can be classified as "suspected" or "concomitant'. Reports are submitted by a physician, a chemists, or a private citizen.

Each report includes patient's information (age, nationality, gender, weight, outcome of reactions, and so on), drug(s) involved in the suspected reaction(s) (identified by their *Anatomical Therapeutic Chemical* - ATC - classification, brand name, dosage), and the description of the occurred adverse reaction(s) encoded by means of the MedDRA classification [21]: MedDRA is a standard medical terminology used to classify adverse event information associated with the use of bio-pharmaceuticals and other medical products (e.g. medical devices and vaccines).

Temporal data refer to entry date, drug name, exposure period, and adverse reaction. These temporal data are used to investigate any cause-effect relationship among drugs and reaction(s) in different time periods, or according to the time frame of the exposure.

3.3. The Motivating Example

Throughout the paper we will refer to examples from the Verona Psychiatric Case Register (PCR). Table 1 graphically depicts a simplified excerpt of database table `Contact`: `VT` (valid time, i.e., the date of the contact), ContactNumber (internal identifier of the contact), Patient (patient's name), Duration (of the contact), Area (location of the patient at contact time), and Professional (name of the operator responsible of the contact).

Contact

VT	Contact Number	Patient	Duration	Area	Professional	Tuple#
2007-05-14	828	Joan	10	North	Mike	1
2007-09-18	840	Joan	10	North	Romina	2
2007-11-05	859	Joan	15	North	Mike	3
2008-03-11	934	Joan	35	South	Mike	4
2008-03-12	935	Joan	35	West	Mike	5
2008-05-13	936	Joan	20	South	Romina	6

Table 1. A (adapted) fragment of database table Contact, collecting data about contacts of patients with the Community-based Psychiatric Service

We can start making some comments. Generally, one Patient refers to the same Professional: but this may be not always true. In the example of Table 1, we have Professional = "Mike" for all the tuples but Tuple#=2 and Tuple#=6. Thus, FD $Patient \rightarrow Professional$ does not hold: that FD holds if we delete those two tuples out of the six tuples we have in Table 1. Consequently, according to the measurement of Definition 2.2, $G_3(Patient \rightarrow Professional, Contact) = 2$.

Moreover, AFD $Patient \xrightarrow{0.5} Professional$ holds because $g_3(Patient \rightarrow Professional, Contact) \leq 0.5$. Thus, accepting an error of 0.5 (50% of error), we can assert the AFD $Patient \rightarrow Professional$. Instead, the AFD $Patient \xrightarrow{0.1} Professional$ does not hold, because $g_3(Patient \rightarrow Professional, Contact) > 0.1$.

Obviously, the plain FD $X \rightarrow Y$ equals the AFD $X \xrightarrow{0.0} Y$, where $g_3(X \rightarrow Y, r) = 0.0$.

Besides plain AFDs, clinicians could be interested in discovering some temporal properties, relevant even from the clinical point of view. For example, according to the content of Table 1, it could be important to discover that some (approximate) dependencies hold month by month. The dependency from *Patient* to *Duration* holds month by month and it could be related to seasonal conditions influencing the overall state of the patient and requiring different durations of the contact. On the other hand, further dependencies could be observed when corresponding tuples are within a fixed time span. For example, an approximate dependency holds from *Patient* to *Duration*, considering a time span of three months (i.e., by deleting Tuple#=2 and Tuple#=6): sch an approximate dependency could be related to the fact that the same patient usually has contacts of the same duration within some given time span, as the possible changes of a psychiatric state are slow with respect to the frequency of the contacts. As we shall see in the following sections, such dependencies require to group data in different ways, either according to non-overlapping time granules or according to (overlapping) time windows. We observe here that discovering this kind of temporal dependencies over clinical data could help physicians to have a better and deeper understanding and management of some temporal behaviors of their patients.

4. Approximate Temporal Functional Dependencies

Moving from the definitions of FD, TFD, and AFD, we now introduce the concept of ATFD. In the following, we consider the basic temporal extension of the relational model proposed in [7]: we consider relations of a generic relational schema R with attributes $U \cup \{VT\}$, where the set U is that of atemporal attributes, while VT represents the valid time. Moreover, according the the taxonomy proposed in [7] and described in Section 2, we consider here pure temporally grouping TFDs of the form $[r, t\text{-}Group]X \rightarrow Y$, where *t-Group* consists of *granularity* (*Gran*) or *sliding window* (*SW*) *grouping*, only:

1. Grouping on granules (granularity grouping, or *Gran* grouping). A temporal *granularity* is a partition of a temporal domain in indivisible non-overlapping groups, i.e., granules, of time points: minutes, hours, days, months, years as well as working days are *granularities* [22].

   Definition 4.1 (*Grouping by Gran(i)*)**.** Two tuples $t_1, t_2 \in r$ belong to the same temporal group $Gran(i)$ iff $t_1[VT], t_2[VT] \in Gran(i)$ where $Gran(i)$ is the i^{th} granule of granularity *Gran*.

2. Grouping on sliding windows (*SW*). A sliding window[1] $SW(i, k)$ includes all the time points in interval $[i \dots i + k - 1]$. Thus, once we fix the length of the *SW* over relation r (i.e. k in the example), every *SW* over r will feature that length, and will - at most - include k elements (if relation r has tuples for all the time points of interval $[i \dots i + k - 1]$).

   Definition 4.2 (*Grouping by SW(i,k)*)**.** Two tuples $t_1, t_2 \in r$ belong to the same sliding window $SW(i, k)$ iff $t_1[VT], t_2[VT] \in [i \dots i + k - 1]$.

Before introducing ATFD, let us consider a new error measure, namely G_4, we shall use for approximate temporal functional dependencies. G_4 considers the minimum number of tuples in r which *must* be *modified* for the plain TFD to hold on all the tuples of r. In the following, if looking for an FD such as $X \rightarrow Y$, we assume to modify values for the Y attributes, only. The ε parameter is user-defined and it states the maximum error acceptable by that user.

$G_4([r, t\text{-}Group]X \rightarrow Y, r) = min\{|s| \mid s \subseteq r, ((r - s) \cup w) \models [r, t\text{-}Group]X \rightarrow Y\}$

where the set w is the minimal one for which the following formula holds:

$\forall t \in s\ (\exists t' \in w(t[U - Y] = t'[U - Y] \wedge t[VT] = t'[VT]))$

The related *scaled measurement* g_4 is defined as $g_4(X \rightarrow Y, r) = G_4(X \rightarrow Y, r)/|r|$

We anticipate here that, if we consider the *Gran* grouping, the two measurements g_3 and g_4 do not differ. However, as we shall describe in the following, g_3 and g_4 may differ when the *SW* grouping is considered.

[1] Actually a sliding window comes with three parameters: granularity, beginning timestamp, and size as the number of time points inside the window.

4.1. *ATFD with Gran Grouping*

We define the ATFD with granularity grouping as:

Definition 4.3 (*ATFD with Gran grouping*)**.** Let r be a relation over the relational schema R with attributes $U \cup \{VT\}$: let $X, Y \subseteq U$ be attributes of R. Let *Gran* be the reference granularity. Relation r fulfills an approximate temporal functional dependency (written as $r \models [r, Gran]X \overset{\varepsilon}{\to} Y$) iff $g_3([r, Gran]X \to Y, r) \leq \varepsilon$.

That is, the percentage of tuples in the entire relation r to be *deleted* for a ATFD to hold on all the tuples of r is less than ε; tuples of r are then grouped according to the granule of *Gran* their VT value belongs to, to evaluate the considered ATFD. We recall that the count of tuples in r to be deleted refers to the entire relation r, and not to the group - and one tuple may belong to one group only, if we use a *Gran* grouping.

As an example, let us consider the fragment of the database table Contact, as depicted by Table 1, and the above Definition 4.3 based on the measurement G_3.

ATFD $[Contact, Year(i)]$ $Patient \xrightarrow{0.4} Duration$ holds, as tuples for which the rule does not hold, i.e. the tuples Tuple#=3 or Tuple#=6 in the specific example and which need to be deleted for the rule to hold on all the tuples, are less then the 40% in the entire table (Table 1).

In fact, if we group the tuples according to granularity *Year* granularity, we can identify groups *Year*(2007), *Year*(2008). For the first group (*Year*(2007)), two tuples (Tuple#=1 and Tuple#=2) out of three in the group confirm the FD $Patient \to Duration$ for a Duration of 10. For the second group (*Year*(2008)), two tuples (Tuple#=4 and Tuple#=5) out of three in the group confirm the FD $Patient \to Duration$ for a Duration of 35. As a consequence, the overall error is 2/6, or 1/3, and it is smaller than 40%, and the required ATFD $[Contact, Year(i)]$ $Patient \xrightarrow{0.40} Duration$ holds on the fragment of Table 1.

If we again consider the fragment of Table 1 and group tuples according to granularity *Year*, as we did before, we can check ATFD $[Contact, Year(i)]$ $Patient \xrightarrow{0.1} Area$, accepting an error of 10%. While FD $Patient \to Area$ holds on all the tuples of group of *Year*(2007) (where Area = “North”), inside the group of *Year*(2008) the FD (where Area = “South”) fails on one tuple (Tuple#=5) out of the three we have. The overall error is 1/6 or 16.66%, which is greater that the allowed 10%. Thus, ATFD $[Contact, Year(i)]$ $Patient \xrightarrow{0.1} Area$ with an error of 0.1 does not hold on the fragment of Table 1.

As for plain AFD, we can introduce the concept of minimality also for ATFD.

Definition 4.4 (*Minimal ATFD with Gran grouping*)**.** An ATFD $[r, Gran]X \overset{\varepsilon}{\to} Y$ is said to be minimal for r iff $r \models [r, Gran]X \overset{\varepsilon}{\to} Y$ and $\forall X' \subset X$ we have that $r \not\models [r, Gran]X' \overset{\varepsilon}{\to} Y$.

4.2. *ATFD with SW Grouping*

We define the ATFD with sliding window (*SW*) grouping as follows:

Definition 4.5 (*ATFD with SW grouping*)**.** Let r be a relation over the relational schema R with attributes $U \cup \{VT\}$: let $X, Y \subseteq U$ be attributes of R. Let $\{i \dots i+k-1\}$ be a sliding window (SW) of length k. The relation r fulfills an approximate temporal functional dependency (written as $r \models [r, \{i \dots i+k-1\}]X \xrightarrow{\varepsilon} Y$) iff $g_4([r, \{i \dots i+k-1\}]X \rightarrow Y, r) \leq \varepsilon$.

A similar definition can be derived from Definition 4.5 by replacing g_4 with g_3, as we shall discuss in Section 4.3.

We consider as many SWs as possible, every SW sizing k elements: thus, the first considered sliding window is $i \dots i+k-1$, the second considered sliding window is $i+1 \dots i+k$, the third considered sliding window is $i+2 \dots i+k+1$, and so on. Every SW sets up a group (or *chain*) over which the ATFD is checked. The ATFD must hold, with an acceptable amount of error smaller than ε, over the entire database: we recall that, if we *delete* (as for measurement g_3) or *modify* (as for the measurement g_4) a tuple inside a SW, that tuple will remain *deleted* or *modified* in *all* the SWs (either preceding or following the current SW) which include that tuple.

As an example, let us consider the fragment of the database table `Contact`, as depicted by Table 2, where the attribute `VT` refers to the valid time of the tuple at the *day* granularity. If we fix the length of the SW to 5, i.e. every sliding window includes a group (or *chain*) of five days, the first SW will formally include time points {2009-04-11, 2009-04-12, 2009-04-13, 2009-04-14, 2009-04-15}: since relation r in Table 2 has tuples for `VT`=2009-04-11 or `VT`=2009-04-14 or `VT`=2009-04-15, the first SW includes 3 tuples having `VT` values 2009-04-11, 2009-04-14, 2009-04-15, respectively. Thus, the following 6 SWs consider all the possible `VT` value groups {2009-04-11, 2009-04-14, 2009-04-15}, {2009-04-14, 2009-04-15}, {2009-04-15}, {2009-04-26, 2009-04-27, 2009-04-28}, {2009-04-27, 2009-04-28}, and {2009-04-28}.

ATFD $[Contact, \{i \dots i+4\}]$ $Patient \xrightarrow{0.4} Duration$ holds. Indeed, tuples for which the dependency does not hold, i.e. `Tuple#` 3 and `Tuple#` 6 in the specific example, are those which need to be *modified* according to the measurement g_4 of Definition 4.5. More precisely, the value of attribute `Duration` for `Tuple#` 3 has to be changed to 20; the value of attribute `Duration` for `Tuple#` 6 has to be changed to 40. Should we *modify* these two tuples, we shall obtain a plain TFD, holding on all the six SWs. The tuples we modified are less then the 40% of the entire fragment (Table 2), thus proving that the ATFD holds even with a threshold ε of 2/6 (i.e. 1/3), which is smaller than 0.4.

If we again consider the fragment of Table 2 and group tuples according to the same six SWs as we did before, we can now check the ATFD $[Contact, \{i \dots i+4\}]$ $Patient \xrightarrow{0.1} Area$, accepting an error of 10%. The TFD fails on one tuple (`Tuple#` 5, i.e. 1/6 of the entire relation), which needs to be modified according to measurement g_4 of Definition 4.5: thus, the ATFD does not hold with a ε of 0.1.

Analogously to Definition 4.4, we can introduce the concept of minimality also for ATFD with SW grouping.

Definition 4.6 (*Minimal ATFD with SW grouping*)**.** Given an ATFD over $[r, \{i \dots i+k-1\}]$, we define $X \xrightarrow{\varepsilon} Y$ to be

VT	Patient	Duration	Area	Tuple#
2009-04-11	Jackie	20	North	1
2009-04-14	Jackie	20	North	2
2009-04-15	Jackie	15	North	3
2009-04-26	Jackie	40	South	4
2009-04-27	Jackie	40	West	5
2009-04-28	Jackie	30	South	6

Table 2. A fragment taken from the database table `Contact`

minimal for r iff $r \models [r, \{i \dots i+k-1\}]X \xrightarrow{\varepsilon} Y$ and $\forall\, X' \subset X$ we have that $r \not\models [r, \{i \dots i+k-1\}]X' \xrightarrow{\varepsilon} Y$.

4.3. g_3 and g_4 with SW Grouping

When using a *SW* grouping, and only when using this grouping, measurements g_3 and g_4 (and the related G_3 and G_4) may differ: on the other side, when using granularity grouping, measurements g_3 and g_4 do *not* differ. In fact, in the *SW* grouping, according to the measurement g_3 we *delete* one or more tuples: this may modify the temporal relationships among the tuples, the number of tuples inside every *SW*, and - occasionally - the total number of *SW*s to consider. Instead, according to the measurement g_4, we *modify* one or more tuples, leaving unchanged the temporal relationships among tuples, the number of tuples inside every *SW*, and the total number of *SW*s to consider.

VT	Patient	Professional	Tuple#
2010-06-11	Claudia	Mike	1
2010-06-12	Claudia	Mike	2
2010-06-13	Claudia	Romina	3
2010-06-14	Claudia	Romina	4
2010-06-15	Claudia	Romina	5

Table 3. A fragment taken from the database table `Contact`

As an example, let us consider the fragment of `Contact` in Table 3, assuming a length 2 for the *SW* (i.e., we consider $i \dots i+1$). We shall then have the following *SW*s: {2010-06-11, 2010-06-12}, {2010-06-12, 2010-06-13}, {2010-06-13, 2010-06-14}, {2010-06-14, 2010-06-15}, and {2010-06-15}, for a grand total of 5 *SW*s. We are interested in the ATFD $[Contact, \{i \dots i+1\}]\ Patient \xrightarrow{0.25} Professional.$

For the measurement G_3, the FD $Patient \rightarrow Professional$ holds in the *SW*s {2010-06-11, 2010-06-12}, {2010-06-13, 2010-06-14}, {2010-06-14, 2010-06-15}, {2010-06-15}: the FD does not hold in the *SW* {2010-06-12, 2010-06-13}. Consequently, if we delete from Table 3 `Tuple#` 3, the remaining *SW*s are: {2010-06-11, 2010-06-12}, {2010-06-12}, {2010-06-14, 2010-06-15}, {2010-06-15}. The FD holds on all the *SW*s remaining after the deletion of `Tuple#` 3. Thus, the measurement G_3 is 1, as we only deleted one tuple (`Tuple#` 3), obtaining the ATFD with an error of $1/5$ (i.e. 20.00%), which is smaller than the maximum acceptable error of 0.25. The ATFD holds, and the measurement

G_3 returns $G_3([Contact, \{i \dots i+1\}]Patient \rightarrow Op, Contact) = 1$.

For the measurement G_4, if we update in Table 3 `Tuple#` 1 and `Tuple#` 2, setting the value of the attribute `Professional` to "Romina", the ATFD will hold in the *SW*s {2010-06-11, 2010-06-12}, {2010-06-12, 2010-06-13}, {2010-06-13, 2010-06-14}, {2010-06-14, 2010-06-15], and {2010-06-15}, that is in 5 out of the 5 *SW*s. Thus, the measurement G_4 is 2, as we *modified* two tuples (`Tuple#` 1 and `Tuple#` 2) to obtain the ATFD. Changing one tuple only, whatever the tuple is, does not suffice to obtain the required TFD.

$G_4([Contact, \{i \dots i+1\}]Patient \rightarrow Op, Contact) = 2$

We obtain the ATFD with an error of 2/5 (i.e. 40%), which is greater than the maximum acceptable error of 0.25: as a consequence, according to G_4 the ATFD does not hold with the required threshold of 25%.

5. Mining Minimal ATFDs

We now consider how to mine minimal ATFDs both with granularity and sliding window groupings. While mining *Gran* grouping ATFDs can be mapped to mine corresponding suitable AFDs, mining *SW* grouping ATFDs requires ad hoc algorithms. The approach we propose here differs from the one in [14], where we performed both *Gran* and *SW* grouping ATFDs through an AFD analysis by the TANE [10] tool.

5.1. Mining Granularity-based ATFDs

Let us now consider how we can reduce the evaluation of minimal Granularity-based ATFDs to the evaluation of corresponding minimal AFDs: such approach allows us to use well-know algorithms for AFD and to characterize the complexity of mining minimal ATFD. In general, we proceed in three different steps: in the first one (*PreAFD*), the given relation is pre-processed to represent, by a suitable attribute, the granule/window each tuple belongs to. Next, we have an (atemporal) relation to consider for the usual AFD extraction (*AFD* phase). Finally, a suitable post-processing phase (*PostAFD*) is needed to properly identify and represent the mined ATFDs.

As the temporal grouping of a granularity *Gran* is a bijective function, we may conclude that the set E of tuples not satisfying the considered granularity-based ATFD, may be partitioned in subsets $E_{G(i)}$, where $E_{G(i)}$ is the set of tuples of E having their valid time contained in the granule $G(i)$. Thus, it holds $|E| = G_4([r, Gran]X \rightarrow Y, r)$.

Given an instance r with schema $R = U \cup \{VT\}$, a granularity *Gran* and a threshold ε, the preprocessing phase (see Algorithm 1 PreAFD-G(r,G)) builds up the relation $preAFD$ with schema $R' = U \cup \{codGran\}$: the attribute `VT` is replaced by the attribute *codGran*. The tuples of $preAFD$ have the same value for *codGran* if and only if they belong to the same temporal granule of the given granularity *Gran*. More formally, for each tuple t of r, we apply the function $f(Gran, t) : r \mapsto preAFD$, where $f(t)[U] = t[U]$ and $f(t)[codGran] = i$, with $t[VT] \in Gran(i)$. The Algorithm 1 shows the pseudo-code of the preprocessing phase, having complexity $O(|r|)$.

```
Algorithm: PreAFD-G($r$,$G$)
Input: $r$,$G$
Output: $preAFD$
$preAFD \leftarrow \emptyset$ ;                    /* $preAFD$ has schema $U \cup codGran$ */
forall the $t_i \in r$ do
    $t'[U] \leftarrow t_i[U]$;
    $t'[codGran] \leftarrow f(G, t_i)$;
    $preAFD \leftarrow preAFD \cup t'$
end forall
return $preAFD$
```

Algorithm 1: PreAFD-G(r,G)

On the obtained relation *preAFD*, the next phase derives the minimal AFDs, according to the threshold ε. Let *outAFD* be the set of the found AFDs. As discussed in [10], the complexity of the corresponding algorithm is $O(2^{|R|})$, since the number of attributes of *preAFD* is that of R.

The error tuples of the ATFD $[r, Gran]X \rightarrow Y$ are, through the correspondence function f, the same as those of the AFD $codGran, X \rightarrow Y$ on *preAFD*. As *preAFD* and r have the same cardinality (f is total and injective), we have $g_4([r, Gran]X \rightarrow Y, r) = g_4(codGran, X \rightarrow Y, preAFD)$. Indeed, for each $Gran(i)$ for the set $r_i = \{t[U] \mid t \in r \wedge t[VT] \in Gran(i)\}$ we can compute $G_4(X \rightarrow Y, r_i)$ and, by definition, $\sum_i G_4(X \rightarrow Y, r_i) = G_4([r, Gran]X \rightarrow Y, r)$. According to the definition of $f(t)$, we have that the set $r'_i = \{t'[U] \mid t' \in preAFD \wedge t'[codGran] = i\} = r_i$; thus $G_4(X \rightarrow Y, r'_i) = G_4(X \rightarrow Y, r_i)$ and it holds $G_4(codGran, X \rightarrow Y, preAFD) = \sum_i G_4(X \rightarrow Y, r'_i) = \sum_i G_4(X \rightarrow Y, r_i) = G_4([r, Gran]X \rightarrow Y, r)$.

As $|r| = |preAFD|$, we may conclude that $g_4([r, G]X \rightarrow Y, r) = g_4(codGran, X \rightarrow Y, preAFD)$.

The last phase (post-processing) maps the derived AFDs into the minimal ATFDs holding in r. Algorithm 2 *PostAFD-G* maps the AFDs of the form $codGran, X \overset{\varepsilon}{\rightarrow} Y$ into the ATFDs of the form $[r, Gran]X \overset{\varepsilon}{\rightarrow} Y$. The complexity of the algorithm is $O(|outAFD|)$: as the number ($|outAFD|$) is of order $O(2^{|R|})$, Algorithm 2 has complexity $O(2^{|R|})$.

```
Algorithm: PostAFD-G($outAFD$)
Input: $outAFD$
Output: $postAFD$
$postAFD \leftarrow \emptyset$ ;        /* $postAFD$ is a set of ATFDs of the form $codGran, X \overset{\varepsilon}{\rightarrow} Y$ */
forall the $codGran, X \overset{\varepsilon}{\rightarrow} Y \in outAFD$ do
    $postAFD = postAFD \cup ([r, Gran]X \overset{\varepsilon}{\rightarrow} Y)$
end forall
return $postAFD$
```

Algorithm 2: PostAFD-G($outAFD$)

It is straightforward to observe that these three phases allow us to derive all the minimal ATFDs with grouping based on granularity *Gran*, and holding in r with threshold ε.

Finally, a strategy (see Algorithm 3) is needed to evaluate how to mine a relation r according to a set of temporal granularities. Let us focus on ($GranSet$, $\prec$), a set of continuous and total granularities with a total order according to relation $FinerThan$ [22]. Informally, relation $FinerThan(Gran_1, Gran_2)$ holds if each granule of granularity $Gran_1$ is contained in a granule of $Gran_2$. If $FinerThan(Gran_1, Gran_2)$, then:

1. $r \models [r, Gran_2]X \xrightarrow{\varepsilon} Y \Rightarrow r \models [r, Gran_1]X \xrightarrow{\varepsilon} Y$
2. (*pruning condition*) $r \not\models [r, Gran_1]X \xrightarrow{\varepsilon} Y \Rightarrow r \not\models [r, Gran_2]X \xrightarrow{\varepsilon} Y$

Algorithm 3 *StrategyG*(r,*GranSet*) starts to mine, according to order $\prec$, the ATFDs from the minimal (finest) granularity to the coarsest one in (*GranSet*, $\prec$). In mining the ATFDs with coarser granularity, we shall find those ATFDs with antecedent $X' \supseteq X$, where X is the antecedent of a ATFD at a finer granularity. If there is no ATFD at granularity $Gran_i$, for the *pruning condition* no ATFD will be mined at any granularity $Gran(j)$ such that $Gran(i) \prec Gran(j)$. Algorithm 3 *StrategyG*, has complexity $O(2^{|R|})$, since the three phases *PreAFD-AFD-PostAFD* are executed at most |*GranSet*| times.

```
Algorithm: StrategyG($r$,$GranSet$,$\varepsilon$)
Input: $r$,$GranSet$,$\varepsilon$
Output: ATFD$s$
currGran ← $inf\{Gran_i \in GranSet\}$;
ATFD$s$ ← $\emptyset$;
while $currGran \neq \emptyset$ do
    $ATFD \leftarrow \{[r, currGran]X \xrightarrow{\varepsilon} Y$ found in $r$ executing PreAFD-G, AFD, and PostAFD-G$\}$;
    if $ATFD = \emptyset$ then
        return ATFD$s$
    else
        ATFD$s$ ← ATFD$s$ ∪ $ATFD$;
        GranSet ← GranSet - currGran ;            /* ATFD$s$ is the set of all minimal ATFDs */
        ;                    /* $[r, Gran_i]X \xrightarrow{\varepsilon} Y$ valid in $r$ for granularities $\in GranSet$ */
        currGran ← $inf\{Gran_i \in GranSet\}$
    end if
end while
return ATFD$s$
```

Algorithm 3: StrategyG(r,*GranSet*,ε)

5.2. Mining SW-based ATFDs

For *SW* grouping, in [14] we adopted an approach in 4 phases: *PreAFD-G*, *AFD*, *PostAFD-G*, and *StrategyG*. We introduce here a novel approach for *SW*-based analysis, which does not need any *PreAFD* and *PostAFD* step. In the following we will focus on G_3 and g_3 error measures.

We aim at verifying whether the ATFD $[R, t\text{-}Group]X \xrightarrow{\varepsilon} Y$ holds over an instance r of a temporal relational schema $R(U, VT)$. First, we define the relation $ValueCount(r, v) = \{(y, vt, c) \mid c = |\{t \mid t \in r \wedge t[VT] = vt \wedge t[X] = v \wedge t[Y] = y\}|\}$, which, given the instance r of the schema R and a tuple v of values for attributes X, returns triples (y, vt, c). A triple (y, vt, c) belongs to $ValueCount(r, v)$ iff there exists exactly $c > 0$ distinct tuples $t \in r$ where $t[X] = v \wedge t[Y] = y \wedge t[VT] = vt$.

Given r as above, a set $r' \subseteq r$ is *minimal* for $[R, \mathit{SlidingWindow}(k)]X \rightarrow Y$ iff r - r' fulfills $[R, \mathit{SlidingWindow}(k)]X \rightarrow Y$, and for every $r'' \subset r'$ we have that r - r'' does not satisfy $[R, \mathit{SlidingWindow}(k)]X \rightarrow Y$.

In order to identify the ATFD $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ we can restrict our analysis to the minimal sets r': it can be easily observed that $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ holds over a given instance r iff there exists a minimal set r' for $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ with $|r'| \leq \epsilon \cdot |r|$. By restricting our analysis to minimal sets, only, we have to find *one* minimal set with minimum cardinality. This allows us to check immediately if an ATFD $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ holds over a given instance of R. Let us consider now the following result over minimal sets.

Lemma 5.1. Given an instance r of a temporal relational schema $R(U, VT)$, for every $k \in \mathbb{N}$, for every minimal set $r' \subseteq r$ for $[R, \mathit{SlidingWindow}(k)]X \rightarrow Y$, and for every value v, it holds $\mathit{ValueCount}(r', v) \subseteq \mathit{ValueCount}(r, v)$

```
Algorithm: VerifySW($[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$, $r$)
Input: $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y, r$
Output: TRUE or FALSE
$\sigma \leftarrow 0$;
foreach $v \in \{x \mid \exists t \in r(t[X] = x)\}$ do
    $\sigma \leftarrow \sigma + \mathit{MinDelete}(\mathit{ValueCount}(r, v), k)$;
    if $\sigma > \epsilon \cdot |r|$ then
        return FALSE
    end if
end foreach
return TRUE ;                /* verify if $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ holds over $r$ */
```

Algorithm 4: VerifySW(. . .)

Lemma 5.1 provides us with the following property: the tuples of r which have the same attribute values for (X, VT) are either totally deleted or totally kept in a minimal set $r' \subseteq r$ for $[R, \mathit{SlidingWindow}(k)]X \rightarrow Y$.

By means of this property, a procedure that verifies $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ over r is described in Algorithm 4. By definition, we have that r does not satisfy $[R, \mathit{SlidingWindow}(k)]X \rightarrow Y$ iff there exists at least one pair of tuples t, t' with $t[X] = t'[X] \wedge t[Y] \neq t'[Y] \wedge t'[VT] - t[VT] \leq k$.

Thus, we can partition r into $r = r_{v_1}, \ldots, r_{v_m}$, where $r_{v_i} = \{t \mid t \in r \wedge t[X] = v_i\}$ (i.e., tuples having the tuple v_i of values for attributes X).

For each i, $1 \leq i \leq m$, we compute the minimal set r'_{v_i} for $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ restricted to the set r_{v_i}. It follows immediately that $\bigcup\limits_{1 \leq i \leq m} r'_{v_i}$ represents a minimal set r' for $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ over the whole relation r.

In the procedure, every set r_{v_i} of the partition is represented by its $\mathit{ValueCount}(r, v_i)$ relation, which contains the minimum amount of information needed to determine the cardinality of r'_{v_i}.

The whole collection of relations $ValueCount(r, v_i)$, for $1 \leq i \leq m$, can be computed in $O(|r| \cdot \log(|r|))$, by means of a simple counting aggregation function over r lexicographically ordered and grouped by attributes X, Y, VT.

As shown in Algorithm 4, the auxiliary function *MinDelete* is applied to each relation $ValueCount(r, v_i)$ and it returns the minimum amount of tuples that have to be removed from r_{v_i} to satisfy $[R, SlidingWindow(k)]X \xrightarrow{\varepsilon} Y$ on r_{v_i}. Since $|ValueCount(r, v_i)| \leq |r_{v_i}|$, the overall worst-case complexity is reached when the partition is formed by one set only, namely r, i.e. all the tuples in r assume the same (tuple of) values for attributes X.

Being $O(f(n))$ the worst case complexity of *MinDelete*, where n is the size of the input relation, the overall worst case complexity is $O(|r| \cdot \log(|r|) + f(|r|))$ (as we shall see, k does not affect the complexity of *MinDelete*).

We now have to identify an efficient algorithm to compute the function *MinDelete*, as described in Appendix A. First, Appendix A.1 describes a quadratic time algorithm version of *MinDelete*, that allows us to introduce the main ideas behind our solution, such as the representation of temporal relations by Directed Acyclic Graphs (DAG): this naive algorithm shows that $[R, SlidingWindow(k)]X \xrightarrow{\varepsilon} Y$ can be computed in polynomial time for every instance r of R. Next, Appendix A.2 improves the asymptotic complexity of *MinDelete*, by providing an $O(|r| \cdot \log |r|)$ version of its.

6. Mining Clinical Data

We developed two running prototypes for off-line analysis: G-ATFDminer (*Granularity Approximate Temporal Functional Dependency Miner*); and SW-ATFDminer (*Sliding Windows Approximate Temporal Functional Dependency Miner*).

6.1. Results for mined granularity-related ATFDs

G-ATFDminer is a Java based system aimed at extracting rules of approximate temporal functional dependency for granularity (*Gran*) grouping. We test G-ATFDminer on the psychiatric data set of Section 3.1, and on the phamarcovigilance data set of Section 3.2.

We start by considering the scalability of the implemented software. The parameters of the algorithm are set as follows: $\epsilon = 0.1$; time granularity set to *MONTH*.

6.1.1. Performance Analysis for GATFDminer

The first analysis refers to tests with a fixed number of rows, but a varying number of attributes. G-ATFDminer was tested on a machine equipped with a 6 core AMD Opteron^{TM} 4284, and 8 GB of RAM. We use the Ubuntu 12.04 64-bit (kernel 3.2.0-23-generic) operating system, Java version 1.7.0, and Postgresql 9.1 as DBMS.

Table 4 depicts the processing time in seconds over the psychiatric data set, consisting of 10,000 rows. Likewise, Table 5 depicts the processing time in seconds, according to the number of attributes (the number of rows is kept fixed to 10,000) over the pharmacovigilance data set. Figure 1 depicts the processing time compared to the theoretical overall complexity of the algorithm.

Attributes	**2**	**3**	**4**	**6**	**8**	**10**	**13**	**17**
Time (seconds)	3	6	16	47	177	468	1896	48900

Table 4. Processing time in seconds for the test over 10,000 rows from the psychiatric data set.

Attributes	**2**	**3**	**4**	**6**	**8**	**10**	**13**	**17**
Time (seconds)	2	4	16	108	623	2889	12150	107100

Table 5. Processing time in seconds for the test over 10,000 rows from the pharmacovigilance data set.

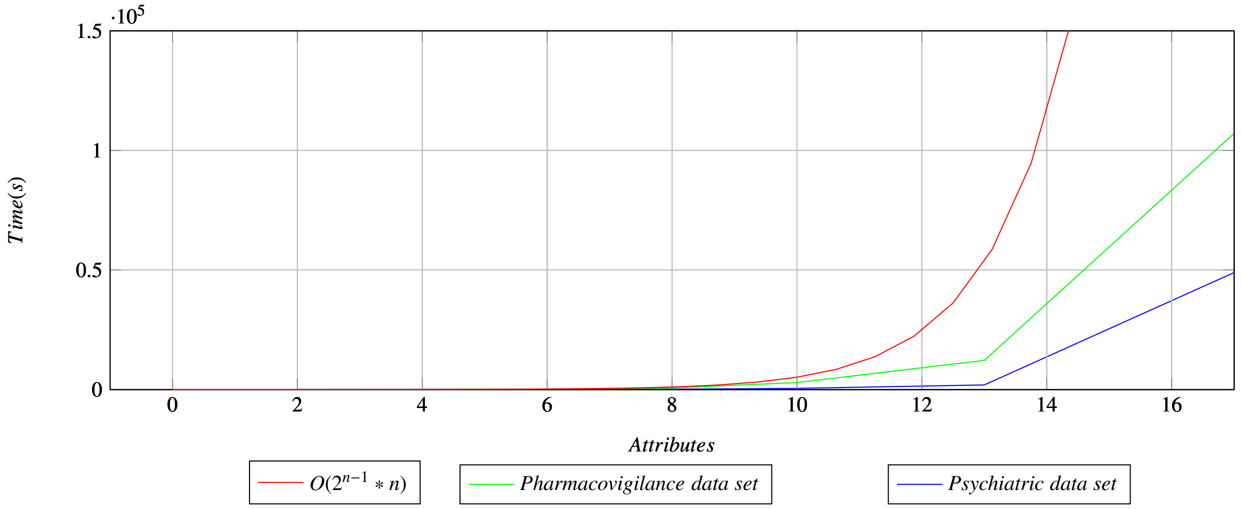

Figure 1. Graphical plot of data from Tables 4 and 5. The red line refers to the theoretical overall complexity of the algorithm; the blue line refers to experimentally detected values over the psychiatric data set; the green line refers to experimentally detected values over the pharmacovigilance data set.

The second analysis refers to tests with a fixed number of attributes (5 in the example) over the psychiatric data set and over the pharmacovigilance data set, but with a varying number of rows. Table 6 depicts the processing time in seconds for the psychiatric data set (5 attributes). Likewise, Table 7 depicts the processing time in seconds for the pharmacovigilance data set (5 attributes): due to the smaller number of rows in this data set, experiments were performed up to $2^{12} \cdot 100$ rows. Figure 2 compares the processing times with the theoretical overall complexity of the algorithm. The processing time is directly proportional (linearly) to the number of rows in input. The estimated complexity is $n \cdot log(n)$, but by using the pruning strategies, we tear down the processing time close to linear time.

6.1.2. Psychiatric Case Register

By running the G-ATFDminer, we identify some meaningful dependencies on the psychiatric data set we discuss here.

Rows $(2^n \cdot 100)$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Time (seconds)	14	17	20	23	22	25	28	31	37	66	231	292	535

Table 6. Processing time in seconds for the test over 5 attributes from the psychiatric data set.

Rows $(2^n \cdot 100)$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Time (seconds)	24	25	25	24	22	26	28	48	48	78	79	132

Table 7. Processing time in seconds for the test over 5 attributes from the pharmacovigilance data set.

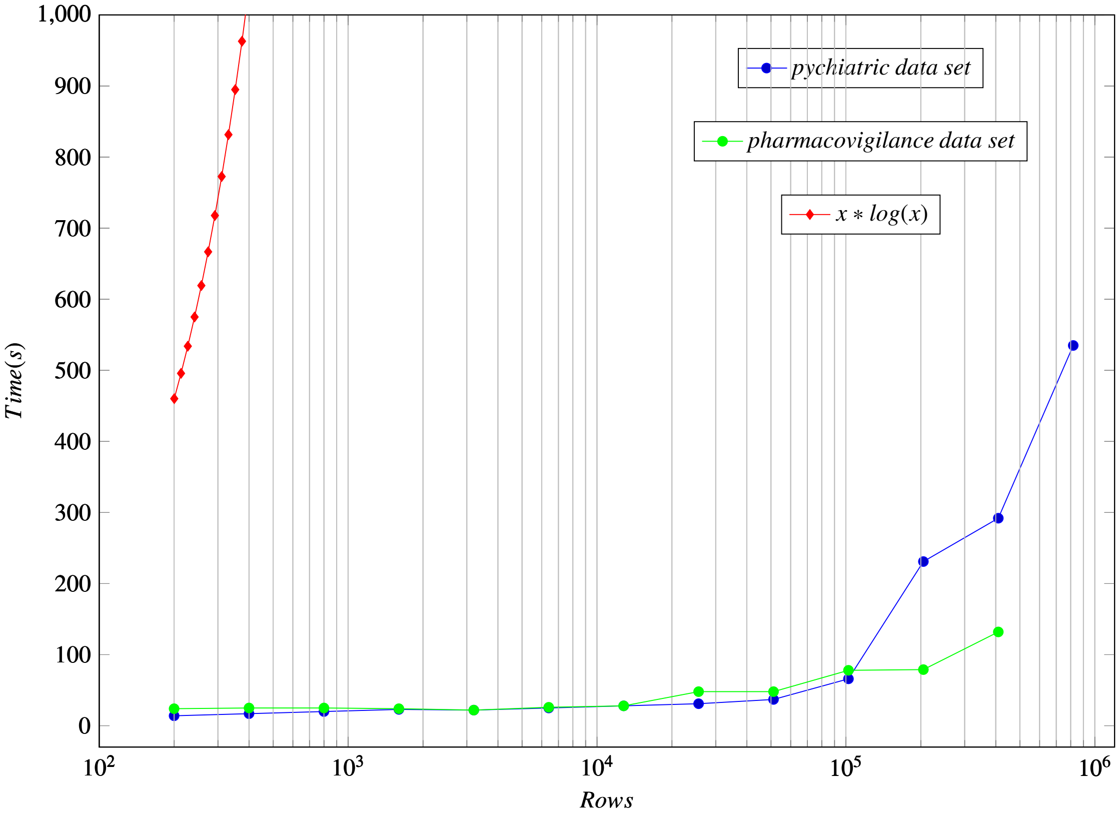

Figure 2. Graphical plot of data from Tables 6 and 7. The red line refers to the theoretical complexity of the algorithm; the blue line refers to experimentally detected values over the psychiatric data set; the green line refers to experimentally detected values over the pharmacovigilance data set. The experimental values are much lower than the theoretical estimation due to suitable pruning strategies.

- $[MONTH]GAF\ Scale \rightarrow Area$: this dependency points out that the state of the patient, expressed by means of the GAF scale, is directly connected to the geographical area the patient lives in. An urban environment, due to its chaotic nature, could negatively influence the state of the patient. Since the dependency refers to the month granularity, one may infer seasonal changes of state in patients, too;

- $[MONTH]Patient \rightarrow GAF\ Scale$: this dependency points out that patients are relatively stable: their state, expressed by the GAF Scale, does not change within a month;

- $[MONTH]Patient \rightarrow Operator_1$: this dependency links the patient to a particular operator ($Operator_1$ is the first person that talks to the patient during a call or a visit). Patients are more used to talk to the same operator, instead of talking to different ones ("strangers");

- $[MONTH]Area \rightarrow$ *Number of Psychiatrists*: this dependency links the geographical area to the number of psychiatrists deployed during one month: shifts or teams of psychiatrists last for at least one month.

6.1.3. Pharmacovigilance

The most relevant dependency we obtained by running the G-ATFDminer on the pharmacovigilance data set is: $[MONTH](Severity, Duration, Gender, Drug) \rightarrow Dosage$. It points out that drug dosage during a therapy, characterized by drug name, patient gender, duration and the severity (expressed a boolean flag), are likely to be adjusted due to reports about occurrences of adverse reaction. This means that the occurrences of adverse reaction may be linked to the dosage of a drug during therapies.

6.2. Results for Mined SW-related ATFDs

SW-ATFDminer (Sliding Window Approximate Temporal Functional Dependencies Miner) is a Java based software extracting rules of approximate temporal functional dependencies for sliding window (*SW*) grouping.

We start by considering the scalability of the implemented software. The parameters of the algorithm are set as follows: $\epsilon = 0.1$; minimum window size is 1 day; maximum window size is 100 years. This means that the mining algorithm returns the maximum window size for ATFDs to hold within this specified interval.

We now describe some of the mined functional dependencies over the two data sets of Section 3.

6.2.1. Performance Analysis for SWATFDminer

The first analysis refers to tests with a fixed number of rows, but different number of attributes. As in Section 6.1, we tested SW-ATFDminer on a machine equipped with a 6 cores AMD Opteron^{TM} 4284, and 8 GB of RAM. We use the Ubuntu 12.04 64-bit (kernel 3.2.0-23-generic) operating system, Java version 1.7.0, and Postgresql 9.1 as DBMS.

Table 8 depicts the processing time in seconds over the psychiatric data set, consisting of 10,000 rows. Likewise, Table 9 depicts the processing time in seconds, according to the number of attributes (the number of rows is kept fixed to 10,000) over the pharmacovigilance data set: due to the smaller number of attributes in this data set, experiments were performed up to 16 attributes. Figure 3 depicts the processing time compared to the theoretical overall complexity of the algorithm.

One may argue that the experimental results (the blue line in Figure 3) show that by our approach, due to suitable pruning strategies, it is possible to mine using 25 attributes before having a decay in the performances. Theoretically, this performance decay should occur as the number of attributes exceeds 10, as depicted by the red line of Figure 3.

Attributes	**2**	**3**	**4**	**6**	**8**	**10**	**13**	**16**	**26**
Time (seconds)	3	6	8	24	118	224	625	5084	113820

Table 8. Processing time in seconds for the test over 10,000 rows from the psychiatric data set.

Attributes	**2**	**3**	**4**	**6**	**8**	**10**	**13**	**16**
Time (seconds)	2	8	15	100	238	425	2031	7069

Table 9. Processing time in seconds for the test over 10,000 rows from the pharmacovigilance data set.

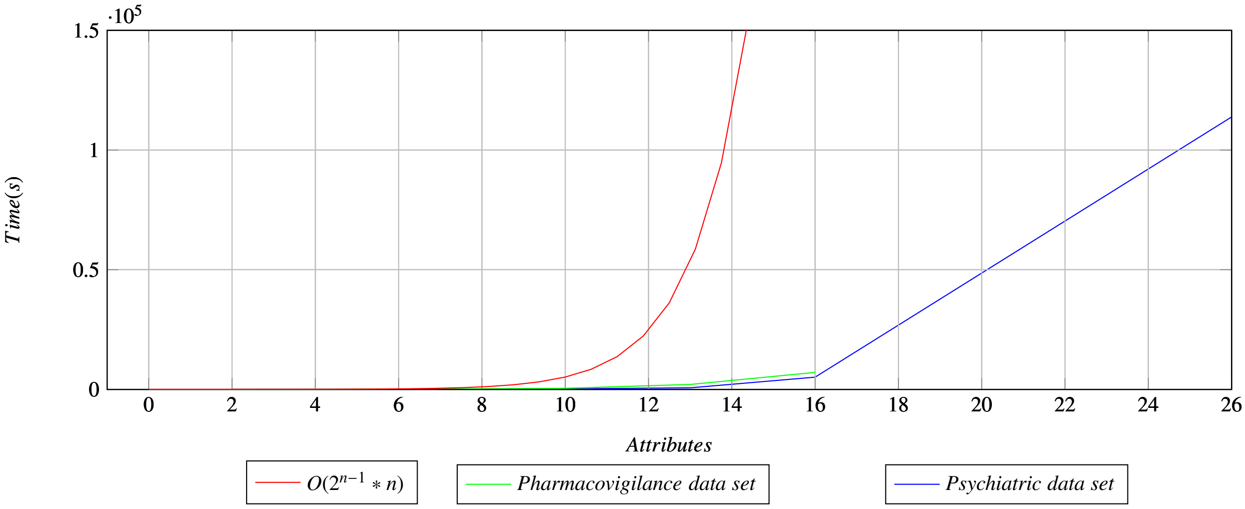

Figure 3. Graphical plot of data from Tables 8 and 9. The red line refers to the theoretical overall complexity of the algorithm; the blue line refers to experimentally detected values over the psychiatric data set; the green line refers to experimentally detected values over the pharmacovigilance data set.

The second analysis refers to tests with a fixed number of attributes (5 in the example) over the psychiatric data set and over the pharmacovigilance data set, but with a varying number of rows. Table 10 depicts the processing time in seconds for the psychiatric data set (5 attributes). Likewise, Table 11 depicts the processing time in seconds for the pharmacovigilance data set (5 attributes): as before, due to the smaller number of rows in this data set, experiments were performed up to $2^{12} \cdot 100$ rows. Figure 4 compares the processing times with the theoretical overall complexity of the algorithm. The processing time is directly proportional (linearly) to the number of rows in input. The estimated complexity is $n \cdot log(n)$, but by using the pruning strategies, we tear down the processing time close to linear time.

Rows $2^n \cdot 100$	**2**	**4**	**6**	**8**	**10**	**11**	**12**	**13**
Time (seconds)	11	12	17	28	73	93	193	243

Table 10. Processing times in seconds for the test over 5 attributes from the psychiatric data set.

Rows $2^n \cdot 100$	**2**	**4**	**6**	**8**	**10**	**11**	**12**
Time (seconds)	20	19	22	30	51	67	89

Table 11. Processing times in seconds for the test over 5 attributes from the pharmacovigilance data set.

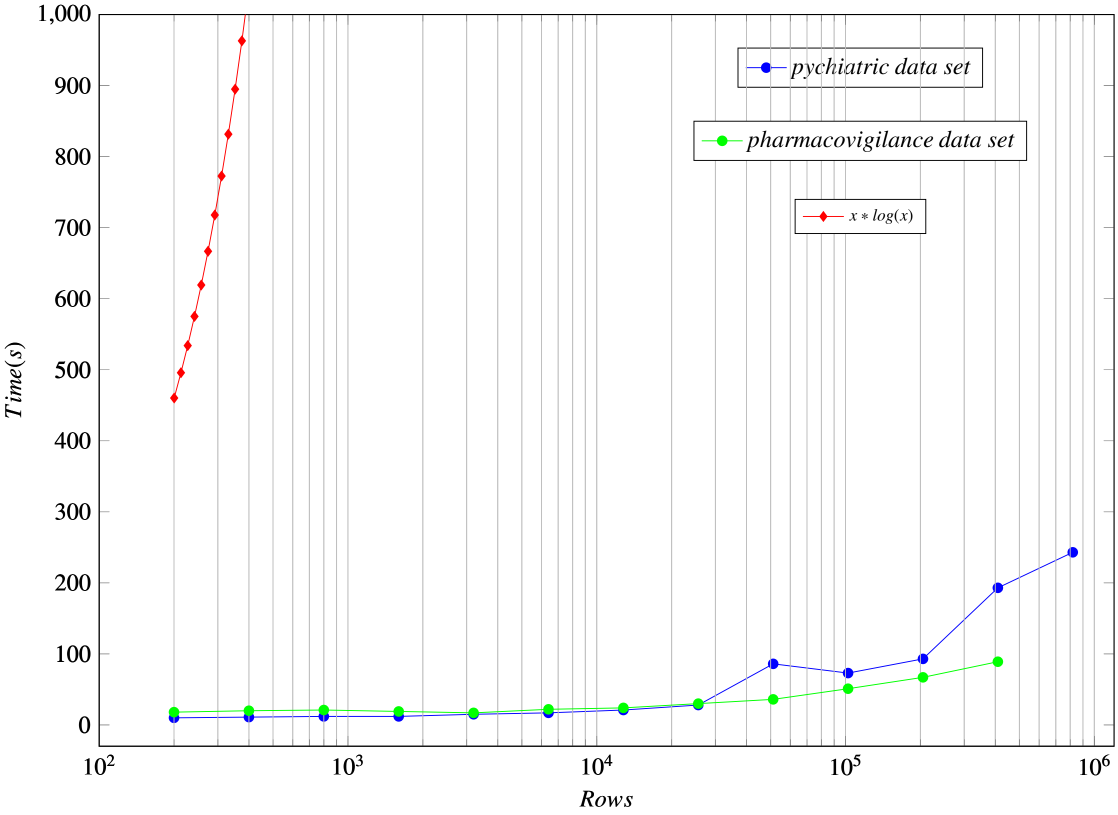

Figure 4. Graphical plot of data from Tables 10 and 11. The red line refers to the theoretical complexity of the algorithm; the blue line refers to experimentally detected values over the psychiatric data set; the green line refers to experimentally detected values over the pharmacovigilance data set. The experimental values are much lower than the theoretical estimation due to suitable pruning strategies.

6.2.2. Psychiatric Case Register

The psychiatric data set consists of 26 attributes. Theoretically, a mining algorithm would need to validate 872,415,232 (i.e. $26 \cdot 2^{25}$) functional dependencies: however, after running the pruning operation, only 125,919 have been tested by SW-ATFDminer to obtain 3042 valid rules. By testing only the $0,014\%$ ($125,919/872,415,232 = 0,014\%$) of all the possible ATFDs (that is 1 functional dependency every 6945), SW-ATFDminer allows one to treat an otherwise intractable problem.

By running the SW-ATFDminer, we identify some meaningful dependencies we discuss here.

- $[133days]HealthStructure \rightarrow ContactType$: due to the large window of 133 days, this dependency points out that healthcare structures are specialized in providing particular contact types (that is either scheduled, urgent, or not classified). For example, urgent contacts are less likely to be registered in community-based structures vs. hospital-based structures.

- $[112 days] Patient \rightarrow GAFScale$: even in this case the window size is very large (112 days). This dependency states that the patient's GAF score (Global Assessment of Functioning is a numeric scale used by mental health professionals to rate subjectively the psychopathology severity, the social and occupational functioning of patients) basically is assessed every 3 months, when changes in the patient's mental condition are observed.

- $[13 days] FirstContact \rightarrow ContactType$: this dependency states that, in a window of 13 days, whenever a patient contacts the service for the first time (*FirstContact* = TRUE), the type of that contact is the same for all the patients. This occurs also when the contact is not the first one (*FirstContact* = FALSE). This could be of interest: in fact, both the antecedent and the consequent of the dependency may assume 2 or 3 values, so one could guess, for instance, that the first contact of a patient is usually urgent, and the other contacts are routine/scheduled ones. This could be considered as an indicator of a good quality of care, as after an urgent contact (usually in an emergency room), the next contact is scheduled with a short waiting list.

- $[12 days](GAFScale, Patient) \rightarrow Duration$: over a window of 12 days, this dependency links the patient and the current condition (measured by the GAF score) to the duration of the contact. It means that, if a patient is scored with a higher functioning (high GAF score), the duration of the outpatient contact is shorter, since the need for psychological, psycho-pharmacological and social support is reduced. If a patient has a lower GAF score, the duration of the outpatient contact is longer.

- $[5 days] Patient \rightarrow Referral$: this dependency states that, in a window of 5 days, the *referral* of the service depends on the patient (typically, the referral is a family member, a neighbor, the police, or the physician). The small window size indicates that the referral of a contact is strictly linked to the condition of the patient at that particular stage of the disease.

- $[5 days](Duration, GAFScale) \rightarrow Professional$: this dependency links the duration of the contact and the GAF score of the patient to the professional involved in the contact. The duration of the contact is longer if the patient is talking to his/her usual professional, while if talking to an unusual professional, the contact is shorter.

6.2.3. Pharmacovigilance

We selected 19 attributes from the pharmacovigilance (PhV) data set, and processed them by the SW-ATFDminer. Theoretically, a mining algorithm would need to validate 4,980,736 functional dependencies (i.e. $19 \cdot 2^{18}$): however, after running the pruning operation, SW-ATFDminer tested only $49,904$ of them.

By running the SW-ATFDminer, we identify some meaningful dependencies we discuss here.

- $[30days](Drug, AdverseReaction) \rightarrow Outcome$: due to the large size of the window, this dependency points out that - given an adverse drug reaction - the outcome does not change. This is of interest for the analyst (i.e., asserting that the suspected drug is indeed the one that caused the reaction). In this case, it is not completely understood by pharmacologists why this dependency does not hold even for bigger window sizes.

- $[23days](Drug, AdverseReaction) \rightarrow HealthcareRegion$: this dependency links a drug and its adverse reaction to the geographical region where that event has been detected. This dependency is explained as follows. Many reports are related to specific active pharmacovigilance projects. For example, many reports from the Lombardy region come from a project focusing on the monitoring of emergency departments: many drug-reaction couples (e.g., bleeding and aspirin) occur several times in the considered data sets due to this project. Even in this case, the length of the sliding window needs further investigations. Moreover, it is of interest to analyze this kind of dependency only for those reports which are not related to specific pharmacovigilance projects.

- $[11days](Drug, TreatmentDuration) \rightarrow Outcome$: this dependency states that the outcome of any adverse reaction induced by a drug can be linked to the drug itself and to the duration of the treatment. This dependency is similar to the first one. In this case, it is interesting to observe that, given a drug, both the induced adverse reaction and the duration of the treatment induce a dependency with respect to the outcome. This could confirm that the same drug may produce different outcomes, according to the time span of the treatment.

- $[6days]AdverseReaction \rightarrow Severity$: this dependency links the reaction to its severity. It is acknowledged by pharmacologists that a reaction is usually severe/not severe regardless of the associated drug. This feature holds even in this case, where severity is associated to the overall report, possibly containing several reactions for the same drug.

- $[5days](ATC, AdverseReaction) \rightarrow DrugRole$: The ATC (*Anatomical Therapeutic Chemical*) classification system is widely used to classify drugs. ATC classifies drugs into different groups, according to the organ or system on which they act, and/or their therapeutic and chemical characteristics. This dependency links the higher level of this classification (e.g. *cardiovascular system*, *dermatology*, *central nervous system*), and the observed adverse reaction to the drug role (e.g. *suspected* of being the cause of the reaction, or just *contemporary*). This dependency may be explained by the “notoriety” of a reaction with respect to a group of drugs. For example, an hypertensive drug is expected to possibly induce hypotension, an anti-arrhythmic one may cause bradycardia. The short width of the sliding window could be explained by the irregular flow of reports and by the fact that different types of reports come in different time periods. Even in this case, the found window size has to be studied and deeply considered by pharmacologists.

7. Conclusions

In this paper, we introduced and discussed approximate temporal functional dependencies, with their related algorithms and clinical data mining issues. More precisely, we discussed how to mine pure temporally grouping temporal functional dependencies. We considered both granularity-based and sliding window temporal groupings. We applied ATFD mining to two different clinical data sets, related to psychiatric patient management and to pharmacovigilance, respectively. ATFDs proved to be an interesting tool for mining clinical data and the derived dependencies have been discussed as for their clinical relevance.

As a future work, we plan to extend mining techniques to other kinds of temporal functional dependencies, according to the framework proposed in [7]. Moreover, a tuning of these techniques for specific temporal clinical data will be considered. In particular ATFDs could be the result of specific tools within an integrated suite of (temporal) data mining tools for clinical data, comprising both temporal dependencies and temporal association rules.

References

[1] C. Combi, G. Pozzi, R. Rossato, Querying temporal clinical databases on granular trends, Journal of Biomedical Informatics 45 (2) (2012) 273–291.

[2] C. Combi, B. Oliboni, G. Pozzi, Modeling and Querying Temporal Semistructured Data, in: S. Kozielski, R. Wrembel (Eds.), Annals of Information Systems - New Trends in Data Warehousing and Data Analysis, Vol. 3, Springer Science+Business Media, New York, NY, USA, 2009, pp. 299–323.

[3] J. Wijsen, Temporal FDs on complex objects, ACM Trans. Database Syst. 24 (1) (1999) 127–176.

[4] V. Vianu, Dynamic functional dependencies and database aging, J. ACM 34 (1) (1987) 28–59.

[5] C. S. Jensen, R. T. Snodgrass, M. D. Soo, Extending existing dependency theory to temporal databases, IEEE Trans. Knowl. Data Eng. 8 (4) (1996) 563–582.

[6] X. S. Wang, C. Bettini, A. Brodsky, S. Jajodia, Logical design for temporal databases with multiple granularities, ACM Trans. Database Syst. 22 (2) (1997) 115–170.

[7] C. Combi, A. Montanari, P. Sala, A uniform framework for temporal functional dependencies with multiple granularities, in: D. Pfoser, Y. Tao, K. Mouratidis, M. A. Nascimento, M. F. Mokbel, S. Shekhar, Y. Huang (Eds.), SSTD, Vol. 6849 of Lecture Notes in Computer Science, Springer, 2011, pp. 404–421.

[8] J. Kivinen, H. Mannila, Approximate inference of functional dependencies from relations, Theor. Comput. Sci. 149 (1) (1995) 129–149.

[9] Y. Huhtala, J. Kärkkäinen, P. Porkka, H. Toivonen, Efficient discovery of functional and approximate dependencies using partitions, in: S. D. Urban, E. Bertino (Eds.), ICDE, IEEE Computer Society, 1998, pp. 392–401.

[10] Y. Huhtala, J. Kärkkäinen, P. Porkka, H. Toivonen, TANE: An efficient algorithm for discovering functional and approximate dependencies, Comput. J. 42 (2) (1999) 100–111.

[11] S. Lopes, J.-M. Petit, L. Lakhal, Functional and approximate dependency mining: database and fca points of view, J. Exp. Theor. Artif. Intell. 14 (2-3) (2002) 93–114.

[12] R. Bellazzi, C. Larizza, P. Magni, R. Bellazzi, Temporal data mining for the quality assessment of hemodialysis services, Artificial Intelligence in Medicine 34 (1) (2005) 25–39.

[13] L. Sacchi, C. Larizza, C. Combi, R. Bellazzi, Data mining with temporal abstractions: learning rules from time series, Data Min. Knowl. Discov. 15 (2) (2007) 217–247.

[14] C. Combi, P. Parise, P. Sala, G. Pozzi, Mining approximate temporal functional dependencies on pure temporal grouping, in: DMBIH, IEEE Workshop Proceedings, IEEE, 2013, pp. 258–265.

[15] E. F. Codd, Normalized data structure: A brief tutorial, in: E. F. Codd, A. L. Dean (Eds.), SIGFIDET Workshop, ACM, 1971, pp. 1–17.

[16] J. Wijsen, Temporal dependencies, in: L. Liu, M. T. Özsu (Eds.), Encyclopedia of Database Systems, Springer US, 2009, pp. 2960–2966.

[17] C. Combi, A. Montanari, G. Pozzi, The T4SQL Temporal Query Language, in: M. J. Silva, A. H. F. Laender, R. A. Baeza-Yates, D. L. McGuinness, B. Olstad, Ø. H. Olsen, A. O. Falcão (Eds.), CIKM, ACM, 2007, pp. 193–202.

[18] World Health Organization and WHO Collaborating Centre for International Drug Monitoring, The Importance of Pharmacovigilance, Safety monitoring of medicinal products, World Health Organization, 2002.

[19] M. Sordo, G. Ochoa, S. N. Murphy, A pso/aco approach to knowledge discovery in a pharmacovigilance context, in: GECCO (Companion), 2009, pp. 2679–2684.

[20] R. Meyboom, M. Lindquist, A. Egberts, I. Edwards, Signal selection and follow-up in pharmacovigilance, Drug Safety 25 (6) (2002) 459–465.

[21] MedDRA MSSO, About MedDRA, in: About MedDRA, 2010, pp. –.
URL `http://www.meddramsso.com/public_about_meddra.asp`

[22] C. Combi, M. Franceschet, A. Peron, Representing and reasoning about temporal granularities, J. Log. Comput. 14 (1) (2004) 51–77.

[23] H. Mannila, K.-J. Räihä, On the complexity of inferring functional dependencies, Discrete Applied Mathematics 40 (2) (1992) 237–243.

Appendix A. The Function MinDelete

This appendix describes two implementations for the *MinDelete* function: Appendix A.1 describes a naive implementation, while Appendix A.2 describes a smarter implementation.

Appendix A.1. A Naive MinDelete Function

Function *MinDelete* of Algorithm 5 receives as input a relation $\overline{r}$ with schema $\overline{R}(Y, VT, C)$; this relation stores, for a given tuple of values for attributes X, the number C of tuples, which share the same value for the attribute Y at the same valid time VT in the original instance r.

Given such relation $\overline{r}$, the algorithm computes the *minimum* number of tuples t, with $t[X] = x$, to be deleted from r in order to have the TFD $[R, SlidingWindow(k)]X \rightarrow Y$ to hold over the remaining tuples of r that share the same tuple x of values for $t[X]$. That is, Algorithm 5 looks for a minimal set $r'_x \subseteq r_x$ for $[R, SlidingWindow(k)]X \rightarrow Y$.

The procedure builds a DAG with positive weights on the edges. The nodes of G represent tuples of the input relation, and have two auxiliary nodes s and e. The set E of edges and their respective weights W are defined as follows:

- there is one edge from s to every $t \in r$, and one edge from every $t \in r$ to e;

```
Algorithm: MinDeleteNaive($\overline{r}, k$)
Input: $\overline{r}, k$
Output: $DAG\_Shortest\_Path(V, E, W, S, f)$
; /* $\overline{r}$ is supposed to contain only tuples $(y, vt, c)$ where $y$ is a value in the domain of
  $Y$, and both $vt$ and $c$ are natural numbers */
assign to $y$ a value $\in Dom(Y)$;
$s \leftarrow (y, \min_{t \in r} t[VT] - k - 1, 0)$;
$e \leftarrow (y, \max_{t \in r} t[VT] + k + 1, 0)$;
$V \leftarrow r \cup \{s, e\}$;
$E \leftarrow \{(s, t) \mid t \in r\} \cup \{(t, e) \mid t \in r\} \cup \{(t, t') \mid t, t' \in r \wedge t[VT] < t'[VT] \wedge (t'[VT] - t[VT] > k \vee t[Y] = t'[Y])\}$;
foreach $(t, t') \in E$ do
    $W(t, t') \leftarrow \sum_{t'' \in r \wedge t'' \neq t' \wedge t[VT] < t''[VT] \leq t'[VT]} t''[C]$
end foreach
return $DAG\_Shortest\_Path(V, E, W, S, f)$
```

Algorithm 5: MinDeleteNaive($\overline{r}, k$)

- one edge connects two tuples $t, t' \in r$, iff the two tuples with subsequent VTs ($t[VT] < t'[VT]$) do not create a conflict with respect to $[R, SlidingWindow(k)]X \to Y$. This occurs when either $t[Y] = t'[Y]$ or $t'[VT] - t[VT] > k$;
- the weight of every edge (t, t') is the number of tuples that must be deleted if all the tuples represented by t together with all the tuples represented by t' will be kept in the final solution and no other tuples are in between, with respect to their valid time. In particular, when considering tuples within the sliding window, we cannot have tuples between $t[VT]$ and $t'[VT]$ with the same corresponding values for attributes X and different values for Y (i.e. violating the dependency).

Every path from s to e in the DAG describes one possible "deletion-strategy". Every edge indicates that the two tuples may coexist without violating the dependency. If an edge $e = (t, t')$ is chosen in the shortest path between s and e, it means that t and t' are kept in r, and thus all the tuples of the DAG in-between them are deleted. The number of such deleted tuples is represented by the weight of the edge e.

As an example, let us consider in Figure A.5 the edge between nodes t_{i+1} and t_{i+6}. Tuples $t_{i+2}, t_{i+3}, \ldots t_{i+5}$ will be deleted. Every path from s to e guarantees that $[R, SlidingWindow(k)]X \to Y$ is satisfied by all the remaining tuples sharing the same tuple x of values for attributes X. Indeed, all the edges on a path guarantee that the two connected nodes represent tuples that can be in a relation satisfying the given temporal functional dependency. Moreover the sum of the weights on every path from s to e is exactly the number of the tuples to be deleted, if the corresponding strategy would be adopted. Finding the weighted shortest path from s to e in the DAG corresponds to identifying the

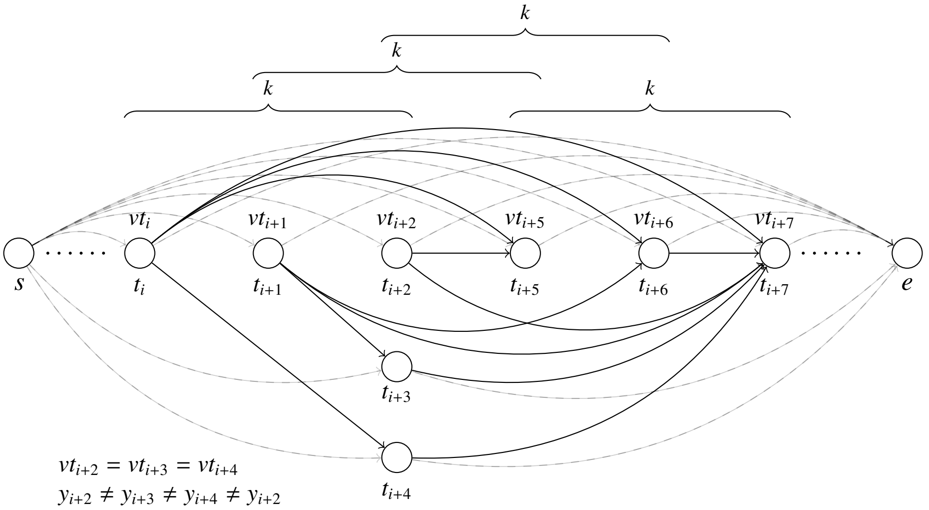

Figure A.5. A fragment of the DAG created by procedure *MinDelete*.

minimum number of tuples to be deleted to make r_x consistent with $[R, \mathit{SlidingWindow}(k)]X \rightarrow Y$, i.e. considering only those tuples with the given (tuple of) values for attributes X.

The complexity of such a procedure is $O(|r|^2)$, and it is determined by the worst case complexity to compute the weights for the edges. Such a computation requires, in the worst case, a quadratic parse of the original relation r: this dominates the overall complexity for every single source shortest path procedure used to compute the output value. We shall see in the next section how such weights can be incrementally computed just in time, tearing down the complexity of the whole procedure by exploiting the particular structure of the generated DAG.

Appendix A.2. A Smart MinDelete Function

The procedure *MinDelete* can be improved with respect to the asymptotic complexity analysis, obtaining the procedure described in Algorithm 6. The soundness and completeness of such a procedure is given by Lemma Appendix A.1, which can be proved by contradiction.

Lemma Appendix A.1. Given a shortest path $P = s, t_1, \ldots, t_m, e$ in the DAG built up according to Algorithm 5, for every pair of consecutive nodes t_i, t_{i+1} in P there does not exist a path P' in the DAG with $|P'| > 1$ and $P' = t_i, t'_1, \ldots, t'_{m'}, t_{i+1}$.

This result strongly depends on how the weights are computed in the DAG, and informally it enables us to disregard all the edges that can lead to a longer path when computing the shortest path through the function *SHP*. *SHP* computes the cost of the shortest path between two given nodes. For example, in the DAG of Figure A.5, the edge (t_i, t_{i+7}) can be deleted without affecting any value for every shortest path. Indeed, looking at the edge weights we observe that:

$$W(t_i, t_{i+7}) = \sum_{j=i+1}^{i+6} t_j[C] > W(t_i, t_{i+4}) + W(t_{i+4}, t_{i+7}) = \sum_{j=i+1}^{i+3} t_j[C] + \sum_{j=i+5}^{i+6} t_j[C]$$

Thus, every path featuring (t_i, t_{i+7}) is not the shortest one, since such an edge may be replaced by the edge pairs $(t_i, t_{i+4}), (t_{i+4}, t_{i+7})$, to obtain a path with a lower weight. Corollary 1 follows from this result.

Corollary 1. Given a shortest path $P = s, t_1, \dots t_m$ in the DAG from s to any node $t_m \in V$, then either $t_m[VT] - t_{m-1}[VT] > k$, or for every t' with $t_{m-1}[VT] < t'[VT] < t_m[VT]$, we have $t'[Y] \neq t_m[Y]$.

This means that every node $t_m \in V$ has either:

- a predecessor in its shortest path outside the sliding window or;
- an immediate predecessor among the tuples with the same value for attribute Y in the sliding window.

We call such a tuple, if it exists, the *minimal window predecessor* of t_m. Given a node t, we say that t', with $t[VT] - t'[VT] > k$, is its *minimal-external predecessor* if and only if for every t'' with $t[VT] - t''[VT] > k$, we have $SHP(s, t'') + W(t'', t) \geq SHP(s, t') + W(t', t)$.

Corollary 2 completes the needed properties for our procedure. It can be proved by contradiction by looking at the properties of the weights in the DAG.

Corollary 2. For every pair of tuples t, t' with $t[VT] \leq t'[VT]$, let $\bar{t}$ and $\bar{t}'$ be the minimal external predecessors of t and t' respectively: then, it holds $\bar{t}[VT] \leq \bar{t}'[VT]$.

Suppose that we are looking for a given node t. Properties highlighted by the two corollaries 1 and 2 restrict the value of $SHP(s, t)$ to $min((SHP(s, t') + W(t', t)), (SHP(s, t'') + W(t'', t))$. Thus, t' is the minimal-external predecessor of t, and t'' is the minimal-window predecessor of t (if any).

The procedure described in Algorithm 6 makes use of these properties to tear down the complexity of procedure *MinDelete* to $O(n \cdot \log(n))$. The procedure requires that the tuples are ordered lexicographically on VT, Y (given an arbitrary order on Y) in the input relation r. In the following, we assume that retrieving a tuple t_j given its index j has a computational cost $\log(|\overline{r}|)$, by supposing that there is some sort of an indexing structure (e.g. a B-tree) built up on the indexes of the tuples, and that the computational cost of computing such a structure is $O(|\overline{r}| \cdot \log |\overline{r}|)$.

Moreover, the procedure assumes that in the input relation there are five additional auxiliary not-initialized attributes LC, PW, PWC, FW and SHP, respectively. These integer attributes have the following meaning:

- LC means "level count", and it represents the sum for the C attribute for all the tuples that share the same VT with the current one (including t itself).

  Formally:

$$t[LC] = \sum_{t' \in r, t[VT] = t'[VT]} t[C]$$

```
Algorithm: MinDelete($\overline{r}$,$k$)
Input: $\overline{r}$,$k$
Output: $Min$
optimized version for MinDelete ;   /* $\overline{r}$ contains only tuples $(y, vt, c, lc, pw, pwc, fw, shp)$ where $y$, $vt$ and $c$ are defined as in Algorithm 5.  Attributes $lc$,$pw$, $pwc$, $fw$ and $shp$ are natural numbers.  */
for $i = 1$ to $|\overline{r}|$ do
    if $i = 1 \vee t_{i-1}[VT] < t_i[VT]$ then
        $Count \leftarrow 0; j \leftarrow i;$
        while $t_j[VT] = t_i[VT] \wedge j \leq |r|$ do
            $Count = Count + t_j[C]$
        end while
    end if
    $t_i[LC] \leftarrow Count$
end for
;  /* $T$ is assumed to be a balanced binary search tree.  Each node of $T$ is ordered on the field $key$ which assumes values in the domain of $y$ and contains attributes $idx$ and $bonus$ which are natural numbers.  */
$T \leftarrow \emptyset; Count \leftarrow 0;$
for $i = |\overline{r}|$ down to $1$ do
    $t_i[PW] \leftarrow$ NIL; $n \leftarrow T.search(t_i[Y]);$
    if $n \neq$ NIL then
        if $t_{n[idx]}[VT] - t_i[VT] \leq k$ then
            $t_{n[idx]}[PW] \leftarrow i; t_{n[idx]}[PWC] \leftarrow Count - n[bonus]$
        end if
        $T.delete(t_i[Y])$
    end if
    $n \leftarrow NewNode(); n[key] \leftarrow t_i[Y]; n[idx] \leftarrow i; n[bonus] \leftarrow Count + t_i[C]; T.insert(n);$
    if $i > 1 \vee t_i[VT] > t_{i-1}[VT]$ then
        $Count \leftarrow Count + t_i[LC]$
    end if
end for
foreach $t \in \overline{r}$ do
    $t[FW] = \min_{t_j \in r \wedge t[VT] \geq t_j[VT] \wedge t[VT] - t_j[VT] \leq k} j$
end foreach
$Ext \leftarrow Win \leftarrow 0;$
for $i = 1$ to $|\overline{r}|$ do
    $OutValue \leftarrow Ext + t_i[LC] - t_i[C];$
    if $t_i[PW] \neq$ NIL then
        $InValue \leftarrow t_{t[PW]}[SHP] + t_i[PWC]; t_i[SHP] \leftarrow \min(InValue, OutValue)$
    else
        $t_i[SHP] \leftarrow OutValue$
    end if
    if $i < |r| \wedge t_i[VT] < t_{i+1}[VT]$ then
        $Win \leftarrow Win + t_i[LC]; Ext \leftarrow Ext + t_i[LC];$
        if $t_{i+1}[FW] > t_i[FW]$ then
            for $j = t_i[FW]$ to $t_{i+1}[FW] - 1$ do
                $Ext \leftarrow \min(Ext, Win + t_j[SHP] - t_j[LC]);$
                if $t_j[VT] < t_{j+1}[VT]$ then
                    $Win \leftarrow Win - t_j[LC]$
                end if
            end for
        end if
    end if
end for
$Min \leftarrow t_{|r|}[SHP]; Count \leftarrow 0;$
for $i = |\overline{r}| - 1$ down to $1$ do
    if $t_{i+1}[VT] > t_i[VT]$ then
        $Count \leftarrow Count + t_{i+1}[LC]$
    end if
    $Min \leftarrow \min(Min, t_i[SHP] + Count);$
end for
return $Min$
```

Algorithm 6: MinDelete($\overline{r}$,k)

Such an attribute is introduced in order to improve the readability of the code: the attribute is computed at the very beginning of the procedure and for every $t \in \overline{r}$ with a simple single scan of $\overline{r}$ (lines from 2 to 10 in Algorithm 6).

- PW stands for "predecessor in window": it is the index for which $t_{t[PW]}$ is the minimal-window predecessor of t in $\overline{r}$.

  Formally (recall that tuples in r are lexicographically ordered on VT, Y):

$$t[PW] = \begin{cases} \max i \left(\begin{array}{c} t_i \in \overline{r} \wedge t[Y] = t_i[Y] \\ \wedge t[VT] - t_i[VT] \leq k \end{array} \right) & \textbf{if} \begin{array}{l} \exists t \in \overline{r}(t[Y] = t_i[Y] \\ \wedge t[VT] - t_i[VT] \leq k) \end{array} \\ \textbf{NIL} & \text{otherwise} \end{cases}$$

- PWC represents the weight of the edge between a node t and its minimal window predecessor (if any):

$$t[PWC] = \begin{cases} W(t_{t[PW]}, t) & \text{if } t[PW] \neq \textbf{NIL} \\ \textbf{NIL} & \text{otherwise} \end{cases}$$

Both PW and PWC are computed in the second for-loop of the algorithm (lines from 13 to 25 of Algorithm 6), but their computation is much more complex with respect to the one for LC values.

In fact, a naive way to compute them may lead to a quadratic complexity, making all our efforts unfruitful. On the contrary, the values PW and PWC can be computed for all the tuples $t \in \overline{r}$ in $O(|\overline{r}| \cdot \log |\overline{r}|)$ by using a balanced tree T, say a B-tree, as an auxiliary structure (any other data structure where the computational cost to search /insert /delete is logarithmic will perform the same way).

A node n of such a tree consists of a key value $n[key]$, which represents a value in the domain of Y, given an arbitrary order of the values for Y. $n[idx]$ represents the index of the last tuple with attribute Y equal to the key encountered in the current window. $n[bonus]$ is a value to easily compute the value of PWC. During the procedure, for every node $n \in T$ we guarantee that for every $n' \in T$, with $n \neq n'$, $n[key] \neq n'[key]$. This second for-loop that computes the values for attributes PW and PWC for all the tuples $t \in r$, works backward from the tuple with the maximum value for the attributes VT, Y. At every step, PW is assigned to NIL for the current tuple t_i, and the key $t_i[Y]$ is searched into T. If there exists a node $n \in T$ with $n[key] = t_i[Y]$ and $t_{n[idx]}[VT] - t_i[VT] \leq k$, then t_i is the *minimal-window predecessor* of $t_{n[idx]}$, and $t_{n[idx]}[PW] = i$.

The value $t_{n[idx]}[PWC]$ is computed as follows: at every step $Count$ is increased by the value $t_i[LC]$ of the current tuple if it represents the first tuple of all the tuples with the same VT. When we insert a node n, we store into $n[bonus]$ the current value of *Count*, which is the sum of all the C attributes for the tuples already encountered. The weight $t_{n[idx]}[PWC] = W(t_i, t_{n[idx]})$ is simply computed as $Count - n[bonus]$.

At the end of the iteration, n is removed from the tree and a new node for the current tuple is inserted into T. For every iteration we have a constant number of retrieve /search /delete /insert operations, each one costing $O(\log |\overline{r}|)$. The loop completes after $O(|\overline{r}| \cdot \log |\overline{r}|)$ operations.

- FW stands for "first in window", and for a tuple t it represents the minimum index j such that a tuple t_j exists for which $0 \leq t[VT] - t_j[VT] \leq k$.

The third for-loop of the algorithm (lines from 26 to 28 of Algorithm 6) computes j for every tuple $t \in \overline{r}$. Algorithm 6 performs $|\overline{r}|$ *min* operations on the indexes, by using the value $t[VT]$ for the current tuple t. We recall that the index i of every tuple is given according to the lexicographical order on VT, Y. Then, for every $i \leq j$ we have $t_i[VT] \leq t_j[VT]$. The complexity of the third loop is $O(|\overline{r}| \cdot \log |r|)$.

The fourth for-loop of the algorithm (from line 30 to 48 of Algorithm 6) for every tuple $t \in \overline{r}$ computes *SHP*. At the end of the fourth loop, we require that $t[SHP] = SHP(s, t)$ for every $t \in \overline{r}$.

We use two additional variables, namely *Ext* and *Win*, belonging to the set of natural numbers. *Ext* represents the *SHP* value for the minimal-external predecessor for the tuple t. *Win* represents the sum of all the attributes C for all the tuples t' of the current tuple t with $t[VT] - t'[VT] \leq k$. For every tuple, the value of the attribute *SHP* is the minimum between the *minimal-external predecessor* and the *SHP* of the minimal-windows predecessor (if any) plus their respective weights. At the end of every iteration, we update - if needed - the value *Ext* (lines from 37 to 47). Then we collect all the tuples that are in the current window, and that will not be included into the next one. Such a set of tuples is non-empty iff $t_{i+1}[FW] > t_i[FW]$ where t_i is the current tuple.

According to the property expressed by Corollary 2, the set containing the candidates for minimal-external predecessor is restricted to the current tuple, and the t_j ones, with $j = t_i[FW], \ldots, t_{i+1}[FW] - 1$. Values for $t_j[SHP] = SHP(s_j)$ are already defined, and the value for $W(t_j, t_{i+1})$ is computed by the most internal for-loop (lines from 40 to 45) by means of *Win*. A graphical representation of this for-loop is depicted in Figure A.6.

Having *SHP*(s, t) for every $t \in \overline{r}$, we still have to determine the cost of the shortest path of the entire graph, i.e. *SHP*(s, f). This is done by the fifth and last for-loop (lines from 50 to 55 of Algorithm 6). Basically, this loop moves backward from the tuple $t \in \overline{r}$ with maximum value for the attributes VT and Y, computing the distance between the current tuple and f: at the end, the minimum value is returned. The complexity of the fifth loop is $O(|\overline{r}|)$ and the overall complexity of the procedure *MinDelete* shown in Algorithm 6 is $O(|\overline{r}| \cdot \log |\overline{r}|)$.

In conclusion, the complexity of procedure *VerifySW* is $O(|\overline{r}| \cdot \log |\overline{r}|)$: this complexity does not depend on the size of the sliding window k, which is provided as input. However, the sliding window size may assume any positive value: we are interested in finding all the independent ATFDs $[R, SlidingWindow(k)]X \xrightarrow{\varepsilon} Y$ with the maximum sliding window size, where only ϵ and r are provided by the user. Indeed, k is the maximum value, for which the approximate temporal dependency $[R, SlidingWindow(k)]X \xrightarrow{\varepsilon} Y$ holds over r.

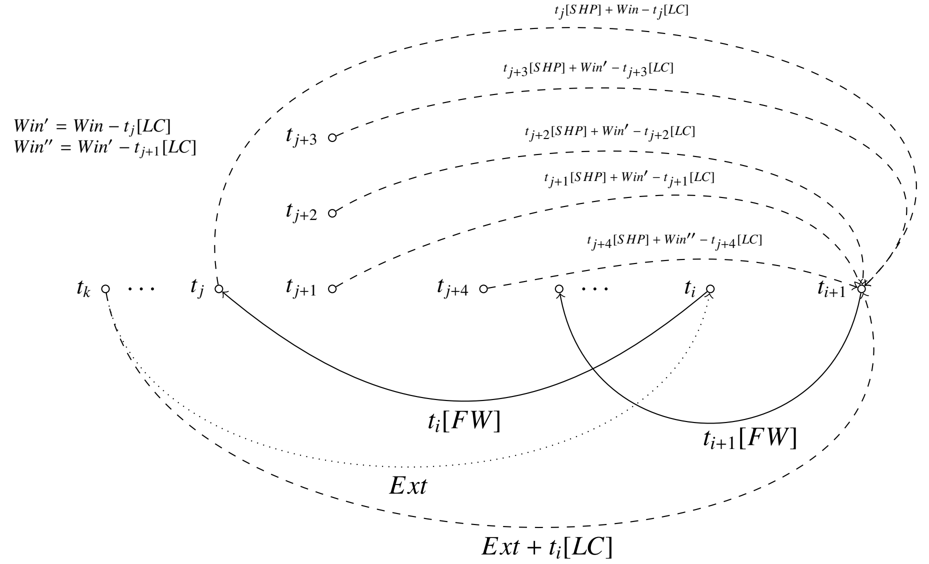
$t_j[SHP] + Win - t_j[LC]$
$t_{j+3}[SHP] + Win' - t_{j+3}[LC]$
$Win' = Win - t_j[LC]$
$Win'' = Win' - t_{j+1}[LC]$
t_{j+3}
$t_{j+2}[SHP] + Win' - t_{j+2}[LC]$
$t_{j+1}[SHP] + Win' - t_{j+1}[LC]$
t_{j+2}
$t_{j+4}[SHP] + Win'' - t_{j+4}[LC]$
t_k
$\cdots$
t_j
t_{j+1}
t_{j+4}
$\cdots$
t_i
t_{i+1}
$t_i[FW]$
$t_{i+1}[FW]$
Ext
$Ext + t_i[LC]$

Figure A.6. A graphical representation of how the minimal external predecessor value Ext is updated when $t_{i+1}[FW] > t_i[FW]$. The new value of Ext is given by the minimum among the values associated to dashed edges (such operation is performed iteratively by the most internal for-loop of lines from 40 to 45 in Algorithm 6).

At the beginning of this section we observed that, in the worst case, the number of couples (X, Y) to be tested is exponential [23]. The size k of the sliding window is provided as a parameter, which does not affect the complexity of the procedure. One may ask if testing all the possible sliding windows in the interval $[0, \max_{t \in r} t[VT]]$ is reasonable: in fact, testing all the possible sliding windows would increase the overall complexity, depending in this case on the value $\max_{t \in r} t[VT]$. However, such a test is not necessary in this case.

Lemma Appendix A.2. (Downward closure property) For every temporal relation R, for every couple of attributes X, Y, for every instance r of R, for every $0 \leq \epsilon \leq 1$, and for every k, if $[R, SlidingWindow(k)]X \xrightarrow{\varepsilon} Y$ holds over r, then for every $k' \leq k$ we have that $[R, SlidingWindow(k)]X \xrightarrow{\varepsilon} Y$ holds over r.

Lemma Appendix A.2 asserts that finding the maximum k, if it exists, for which $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ holds over r, may suffice. Then, by having X, Y and ϵ fixed, we perform a dichotomic search by the procedure *VerifySW*, starting from $k = \max_{t \in r} t[VT]$, till we either terminate unsuccessfully or find the maximum sliding window k for which $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ holds over r. The complete procedure is given in Algorithm 7.

VerifySW can be applied at most $\log(\max_{t \in r} t[VT])$ times. Finally, let $k_{max} = \max_{t \in r} t[VT]$ be the maximum value for the attribute VT in the instance r; the encountered complexity in finding the maximum size k (if any) for which $[R, \mathit{SlidingWindow}(k)]X \overset{\varepsilon}{\rightarrow} Y$ holds over r is $|r| \cdot \log(|r|) \cdot \log(k_{max})$, by assuming that the VT is non-negative, and k_{max} is expressed in the lower time-granule of its temporal domain (e.g. if VT is a year-month-day date then k_{max} is expressed in days).

```
Algorithm: MaxSlidingWindow($R, X, \epsilon, Y, r$)
Input: $R, X, \epsilon, Y, r$
Output: $k$
; /* A procedure for the dichotomic search of the maximum size $k$, if it exists, such
  that $[R, SlidingWindow(k)]X \overset{\varepsilon}{\rightarrow} Y$ holds over $r$.  */
$k_{max} \leftarrow \max_{t \in r} t[VT]]; k_{min} \leftarrow 0;$
while $k_{max} \neq k_{min}$ do
    if $VerifySW([R, SlidingWindow(k_{max})]X \overset{\varepsilon}{\rightarrow} Y, r)$ then
        return $k_{max}$
    end if
    if NOT $VerifySW([R, SlidingWindow(k_{min})]X \overset{\varepsilon}{\rightarrow} Y, r)$ then
        return NIL
    end if
    $k \leftarrow \left\lceil \frac{k_{max}+k_{min}}{2} \right\rceil;$
    if $VerifySW([R, SlidingWindow(k)]X \overset{\varepsilon}{\rightarrow} Y, r)$ then
        $k_{min} \leftarrow k$
    else
        $k_{max} \leftarrow k$
    end if
end while
if $VerifySW([R, SlidingWindow(k_{max})]X \overset{\varepsilon}{\rightarrow} Y, r)$ then
    return $k_{max}$
else
    return NIL
end if
```

Algorithm 7: MaxSlidingWindow(R, X, ϵ, Y, r)