SN Computer Science (2020) 1:153
https://doi.org/10.1007/s42979-020-00160-9

ORIGINAL RESEARCH

Discovering Evolving Temporal Information: Theory and Application to Clinical Databases

Pietro Sala[1] · **Carlo Combi**[1] · **Matteo Mantovani**[1] · **Romeo Rizzi**[1]

Received: 10 March 2020 / Accepted: 9 April 2020 / Published online: 8 May 2020

Abstract
Functional dependencies (FDs) allow us to represent database constraints, corresponding to requirements as "*patients having the same symptoms undergo the same medical tests*." Some research efforts have focused on extending such dependencies to consider also temporal constraints such as "*patients having the same symptoms undergo in the next period the same medical tests*." Temporal functional dependencies are able to represent such kind of temporal constraints in relational databases. Another extension for FDs allows one to represent approximate functional dependencies (AFDs), as "*patients with the same symptoms generally undergo the same medical tests*." It enables data to deviate from the defined constraints according to a user-defined percentage. Approximate temporal functional dependencies (ATFDs) merge the concepts of temporal functional dependency and of approximate functional dependency. Among the different kinds of ATFD, the *Approximate Pure Temporally Evolving Functional Dependencies* (*APE*-FDs for short) allow one to detect patterns on the evolution of data in the database and to discover dependencies as "*For most patients with the same initial diagnosis, the same medical test is prescribed after the occurrence of same symptom*." Mining ATFDs from large databases may be computationally expensive. In this paper, we focus on *APE*-FDs and prove that, unfortunately, verifying a single *APE*-FD over a given database instance is in general NP-complete. In order to cope with this problem, we propose a framework for mining complex *APE*-FDs in real-world data collections. In the framework, we designed and applied sound and advanced model-checking techniques. To prove the feasibility of our proposal, we used real-world databases from two medical domains (namely, psychiatry and pharmacovigilance) and tested the running prototype we developed on such databases.

Keywords Temporal data mining · Temporal functional dependencies · Temporal databases · Distributed algorithms · Complexity · Pharmacovigilance · Psychiatric case register

Introduction

Since some decades, in most of the real-world domains, there is the need of storing and analyzing huge and often overwhelming quantities of data, which are required both for decision making and in the wider area of the management of complex organizations [23, 29]. According to this scenario, and without loss of generality, in this paper, we will focus on the healthcare/medical domain, where such need arises, to support clinical decision-making and healthcare policies [3]. Advanced techniques such as data mining and analysis allow medical stakeholders to extract useful knowledge from these data. In particular, it is often the case that such knowledge is inherently *temporal*, as it is discovered when analyzing data evolution, time series, and changes of information over time. *Temporal* data mining is the research area focusing on the analysis and discovery from data having some specific temporal characterization [8, 9, 25].

Considering data stored according to the well-known relational model, functional dependencies (FDs) are usually specified for expressing constraints on data and for improving the quality of database schemata, by deriving normal forms [2]. However, functional dependencies (FDs)

✉ Carlo Combi
carlo.combi@univr.it

Pietro Sala
pietro.sala@univr.it

Matteo Mantovani
matteo.mantovani@univr.it

Romeo Rizzi
romeo.rizzi@univr.it

[1] Department of Computer Science, University of Verona, Verona, Italy

could be used to derive some knowledge about the given database. As an example, let us consider a simple relation describing the adverse drug reactions patients may have during a hospitalization. Such a relation stores patient demographic data, together with drugs taken and the adverse reactions possibly occurring. Moreover, a temporal attribute time-stamps the adverse reactions. Typically, patients taking the same drug may have the same adverse reaction. Thus, we can derive a functional dependency between the patient drug and the adverse reaction. It may also be that such functional dependencies hold on "most tuples," but not on *all* of them. Such dependencies have been named *approximate* functional dependency (AFD) [13, 16]. As an example, if we consider patients affected by some allergies that make unpredictable the reaction to a drug, the corresponding tuples will likely differ from all the other ones as for the dependency between drugs and adverse reactions.

Finer constraints may also be discovered. As for adverse drug reactions, for example, further drug prescriptions may follow, to mitigate these known effects. For example, suppose that some drugs are prescribed just to mitigate some well-known adverse reaction. Thus, the drug prescribed just after a given adverse reaction will be related both to the adverse reaction and to the drug previously taken by patients. In such a case, prescribed drugs and related adverse reactions determine drugs administered next. We call this dependency a *temporal* functional dependency (TFD).

Approximate functional dependencies have been extensively considered, and some tools have been proposed for deriving such dependencies [13, 14, 16, 19]. On the other side, temporal functional dependencies have been proposed according to different perspectives and considering different kinds of temporal features [7, 15, 30–32]. To the best of our knowledge, only some recent studies focused on *approximate temporal* functional dependencies [4, 5, 11, 26].

In this paper, we continue such studies by considering a different kind of approximate TFD and its application to data from clinical domains. More specifically, we will adopt the framework for temporal functional dependencies proposed by Combi et al. in [7], which allows the specification of multiple kinds of temporal functional dependencies. According to this framework, we consider here the issue of mining (approximate) temporal functional dependencies based on tuple temporal evolution. *Temporal evolution* of tuples has been originally proposed by Vianu [30] for the characterization of *dynamic functional dependencies* (DFDs), which allow one to specify constraints on the evolution of tuples in consecutive snapshots of a temporal database. Here, we consider the characterization of DFDs introduced in [7], called Pure Temporally Evolving TFDs (*PE*-FDs). In particular, we consider the problem of extracting all Approximate *PE*-FDs, called *APE*-FDs, from a given temporal medical database.

Before moving to the more experimental side of our work, we provide a "negative," yet interesting, result about the complexity of checking *APE*-FDs. First, we prove that checking a single *APE*-FD against a database instance is *NP*-Complete in the size of the instance (i.e., data complexity). Moreover, we noticed that the *NP*-completeness of this problem heavily relies on instances that are fictitious and imply properties of data that are unreasonable in many contexts such as the clinical one. We thus came out with a series of optimizations and heuristics that improve the performances with respect to the more general problem of checking an *APE*-FD against a database instance.

As we pointed out, mining *APE*-FDs introduces many computational challenges that require techniques inherited from different fields of Computer Science (e.g., model checking and combinatorial optimization). We embedded such techniques in a framework that has been implemented as a running prototype and applied to data from pharmacovigilance and psychiatry domains. With respect to the preliminary results presented in [10, 27], we focus here only on *APE* -FDs and do not consider the related temporal association rules [10, 27]. We propose here a new, stronger and more focused definition of *PE*-FD and of the related *APE*-FD, by introducing also a bounded version of temporal evolution of data. Moreover, we provide a detailed discussion and proof of our theoretical results, by introducing a significantly improved and extended presentation with new and more complete examples. Furthermore, we introduce a completely new section, in which we propose a couple of novel optimization techniques for solving the problem of checking *APE*-FDs.

In the following, "Background and Related Work" section describes the background and the related work. "Discovering Pure Temporally Evolving Functional Dependencies" section formally introduces the concepts of *PE*-FD and *APE*-FD. "Some Motivating Clinical Scenarios" section introduces some motivating clinical scenarios using *PE*-FDs and *APE* -FDs. "The Computational Complexity of Checking *APE*-FD" section proves the *NP*-Hardness of checking an *APE*-FD against a given temporal database. "Algorithms for Checking *APE*-FDs" section provides a description of the algorithm that checks a single *APE*-FD against a given database plus a series of optimizations and heuristics that may be generally implemented in order to speed up such verification process. "Mining *APE*-FDs" section provides a high-level description of the main features of our prototype for mining such dependencies and the main ideas underlying its implementation; then, it provides interesting mined *APE*-FDs from the psychiatry and pharmacovigilance domains; in the last part of this section, we analyze the performances of the implemented prototype. "Conclusions" section draws some conclusions and sketches possible directions for future research.

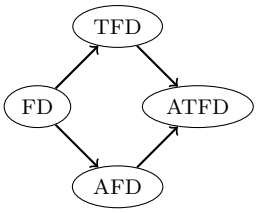


Fig. 1 A graphical account for the IS_A relationships between functional dependency (FD), approximate functional dependency (AFD), temporal functional dependency (TFD) and approximate temporal functional dependency (ATFD)

Background and Related Work

In this section, we introduce and discuss the main definitions and concepts we will use through this paper. We first recall the definition of functional dependency (FD). Then, we introduce some extensions of FDs, i.e., temporal functional dependencies (TFDs) and approximate functional dependencies (AFDs). The definition of approximate temporal functional dependency (ATFD) is grounded on these concepts. Figure 1 depicts the relationships among such kinds of functional dependencies.

Functional Dependencies and Their Temporal Extensions

The concept of functional dependency (FD) comes from the database theory [2].

Definition 1 Let $\mathbf{r}$ be a relation with schema R. Let X, Y be sets of attributes of R. A functional dependency between X and Y

$$\mathbf{r} \vDash X \rightarrow Y$$

represents the constraint that for all couples of tuples t and t' in having the same value(s) on attribute(s) X, the corresponding value(s) on Y for those tuples are identical.

More formally, relation $\mathbf{r}$ satisfies functional dependency $X \rightarrow Y$ if the following condition holds:

$$\forall t, t' \in \mathbf{r}(t[X] = t'[X] \rightarrow t[Y] = t'[Y])$$

Temporal functional dependencies (TFDs) have been proposed as extensions of (atemporal) functional dependencies [33]. As an example, we may represent the constraint that a pathology functionally determines a corresponding drug, but only considers tuples month by month. In other words, the patients with the same pathology are treated with a common drug during some month, while in another month the same patients affected by the same pathology take another (common) drug. Combi et al. proposed a framework for TFDs that subsumes and extends previous proposals [7]. They use a temporal relational data model, allowing one to represent the notion of *temporal relation*. Each relation is equipped with a time-stamping temporal attribute VT, which represents the *valid time*, i.e., the time when the fact is true in the represented real world [6]. VT has values in domain $\mathcal{T}$ isomorphic to $\mathbb{N}$.

Two temporal views allow joining tuples that satisfy specific temporal conditions, which represent relevant cases of (temporal) evolution. On the basis of the introduced data model, and leveraging such temporal views, we may provide a definition for TFDs.

Definition 2 Let $R = U \cup \{VT\}$ be a relational schema where attributes in U are atemporal and VT has domain $\mathcal{T}$. A TFD is expressed as

$$[E\text{- }Exp(R), t\text{- }Group]X \rightarrow Y$$

where E- $Exp(R)$ is a relational expression on schema R, called *evolution expression*, t- $Group$ is a mapping $\mathcal{T} \rightarrow 2^{\mathcal{T}}$, called *temporal grouping*, and $X \rightarrow Y$ is a functional dependency on the (atemporal) attributes U of E- $Exp(R)$.

A TFD is a statement about admissible temporal relations. A temporal relation $\mathbf{r}$ on a temporal schema $R = U \cup \{VT\}$ satisfies a TFD $[E\text{- }Exp(R), t\text{- }Group]X \rightarrow Y$, if it is not possible that the relation obtained from $\mathbf{r}$ by applying the expression E- $Exp(R)$ features two tuples t, t' such that

1. $t[X] = t'[X]$,
2. $t[VT]$ and $t'[VT]$ (the same for $t[\overline{VT}]$ and $t'[\overline{VT}]$, if attribute $\overline{VT}$, obtained by renaming VT, appears in the relation resulting from the evolution expression) belong to the same temporal group, according to the mapping t- $Group$,
3. $t[Y] \neq t'[Y]$.

In other words, FD $X \rightarrow Y$ must be satisfied by each relation obtained from the evolution expression by selecting those tuples whose valid times belong to the same temporal group. Temporal grouping enables us to group tuples together over a set of temporal granules, based on VT. Four different classes of TFD have been proposed in [7]:

- *Pure temporally grouping* TFD: E- $Exp(R)$ returns the original temporal relation $\mathbf{r}$. These TFDs force FD $X \rightarrow Y$, where $X, Y \subseteq U$, to hold over each set of tuples temporally grouped according to their VT;
- *Pure temporally evolving* TFD: E- $Exp(R)$ specifies how to derive the tuples modeling the evolution of objects. No temporal grouping exists, i.e., all the tuples of $\mathbf{r}$ are considered together;

- *Temporally mixed* TFD: in this case, after evaluating the expression *E- Exp*(*R*), a temporal grouping *t- Group* is performed;
- *Temporally hybrid* TFD *s*: first, the evolution expression *E- Exp*(*R*) allows the selection of those tuples that are needed to represent the evolution of real-world objects/concepts; then, temporal grouping is applied to the selected tuples.

In the remainder of the paper, we shall focus on Pure Temporally Evolving TFDs only.

Approximate Functional Dependencies and Their Temporal Extensions

The concept of approximate functional dependency (AFD) is defined moving from the concept of plain FD. In fact, given a relation $\mathbf{r}$ where an FD holds for *most* tuples in $\mathbf{r}$, we may identify *some* tuples, for which that FD does *not* hold. Consequently, we can define some measurements of the error we make in considering the FD to hold on $\mathbf{r}$.

One measurement [16] we can apply is known as G_1, which considers the number of violating pairs of tuples. Formally:

$$G_1(X \to Y, \mathbf{r}) = |\{(t,t') \ : \ t,t' \in \mathbf{r} \wedge t[X] = t'[X] \wedge t[Y] \neq t'[Y]\}|$$

The related *scaled measurement* g_1 is defined as follows:

$$g_1(X \to Y, \mathbf{r}) = G_1(X \to Y, \mathbf{r})/|\mathbf{r}|^2$$

where $|r|$ is the cardinality of relation $\mathbf{r}$, i.e., the number of tuples belonging to relation $\mathbf{r}$.

Another measurement [16] we can apply is known as G_2, which considers the number of tuples that violate the functional dependency. Formally:

$$G_2(X \to Y, \mathbf{r}) = |\{t \ : \ t \in \mathbf{r} \wedge \exists t'(t' \in \mathbf{r} \wedge t[X] = t'[X] \wedge t[Y] \neq t'[Y])\}|$$

The related *scaled measurement* g_2 is defined as follows:

$$g_2(X \to Y, \mathbf{r}) = G_2(X \to Y, \mathbf{r})/|\mathbf{r}|$$

Topics related to approximate functional dependencies have been considered since some years [13, 14, 16, 19]. Instead, to the best of our knowledge, very few studies focused on *approximate temporal* functional dependencies [4, 5, 11, 26]. In [4], Combi et al. consider the problem of mining approximate TFDs with different kinds of temporal grouping on clinical data. In [11, 26], Sala and Combi extend the concept of approximate TFD to deal with *interval-based* TFDs. In this paper, we continue such studies by considering a different kind of approximate TFD and its application to data from clinical domains.

Discovering Pure Temporally Evolving Functional Dependencies

In the following, we focus on *Pure Temporally Evolving Functional Dependencies* (*PE*-FDs for short), as specified in the framework proposed in [7]. Our temporal functional dependencies will be given on a temporal schema $R = U \cup \{VT\}$ where U is a set of *atemporal attributes* and VT is a special attribute denoting the valid time of each tuple. Hereinafter, we assume tuples time-stamped with natural numbers (i.e., $\mathcal{D}om(VT) = \mathbb{N}$). Let $J \subseteq U$ be a nonempty subset of U. We define the set W as $W = U \backslash J$ and set $\overline{W}$, which is basically a renaming of attributes in W. Formally, for each attribute $A \in W$, we have $\overline{A} \in \overline{W}$ (i.e., $\overline{W} = \{\overline{A} : A \in W\}$).

Definition 3 (*Views Evolution and Bounded Evolution*) Given an instance $\mathbf{r}$ of R, an instance $\tau_J^{\mathbf{r}}$ of schema $R_{ev} = JW\overline{W}\{VT, \overline{VT}\}$ is defined as follows:

$$\tau_J^{\mathbf{r}} = \left\{ u \ \middle| \ \exists t,t' \left(\begin{array}{c} r(t) \wedge r(t') \wedge t[J] = t'[J] = u[J] \wedge u[W] = t[W] \wedge \\ u[\overline{W}] = t'[W] \wedge t[VT] = u[VT] \wedge t'[VT] = u[\overline{VT}] \\ \wedge t[VT] < t'[VT] \wedge \\ \forall t''((r(t'') \wedge t[VT] < t''[VT]) \rightarrow t'[VT] \leq t''[VT]) \end{array} \right) \right\}$$

Schema R_{ev} is called the *evolution schema* of R. We will denote as τ_J^R the view *Evolution* on R that is built by expression $\tau_J^{\mathbf{r}}$ for every instance $\mathbf{r}$ of R. View τ_J^R joins two tuples t_1 and t_2 that agree on the values of the attributes in J (i.e. $t_1[J] = t_2[J]$), if t_2 immediately follows t_1. More precisely, such tuples are joined if $t_1[VT] < t_2[VT]$ and there does not exist a tuple $t \in \mathbf{r}$ with $t[J] = t_1[J]$ and $t_1[VT] < t[VT] < t_2[VT]$ (i.e., there does not exist a tuple that holds at some point in between the valid times of such tuples).

For application purposes, it would be important to consider in an evolution schema only those pairs of consecutive tuples whose difference between $\overline{VT}$ and VT is within some given bound. Given a parameter $k \in \mathbb{N} \cup \{+\infty\}$, tuples of $\tau_J^{\mathbf{r}}$ are filtered by means of the selection $\Delta_k(\tau_J^{\mathbf{r}}) = \sigma_{\overline{VT}-VT \leq k}(\tau_J^{\mathbf{r}})$ (notice that $\Delta_{+\infty}(\tau_J^{\mathbf{r}}) = \tau_J^{\mathbf{r}}$).

We will denote as $\Delta_k(\tau_J^{\mathbf{r}})$ the view *Bounded Evolution*. It forces to consider only those tuples belonging to $\tau_J^{\mathbf{r}}$ having a temporal distance within the given threshold k. In the following, given a tuple $t \in \tau_J^{\mathbf{r}}$, we denote its temporal distance $t[\overline{VT}] - t[VT]$ with $\Delta(t)$.

Let us now define, by using the introduced temporal view *Evolution*, a slightly restricted version of *Pure Temporally Evolving Functional Dependency* with respect to that defined in [7]. Without loss of generality, such definition will allow us to simplify the notation and to focus on a general kind of temporal evolution of considered data.

Definition 4 (*Pure Temporally Evolving Functional Dependency*) A *Pure Temporally Evolving Functional Dependency* over the temporal schema $R = U \cup \{VT\}$, *PE*-FD for short, is an expression of the form:

$$[\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}.$$

We have that $X \subseteq W$ and $\overline{Y}, \overline{Z} \subseteq \overline{W}$ with $X \neq \emptyset$ and $|Z| = 1$ (Z contains a single attribute). An instance $\mathbf{r}$ of R fulfills a *PE*-FD $[\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}$, written $\mathbf{r} \vDash [\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}$, if and only if for each pair of tuples $t, t' \in \Delta_k(\tau_J^{\mathbf{r}})$ we have $(t[X] = t'[X] \wedge t[\overline{Y}] = t'[\overline{Y}]) \rightarrow t[\overline{Z}] = t'[\overline{Z}]$.

A *PE*-FD could express dependencies as "A common therapy follows the same symptom for all patients."

Now, we introduce two specializations of *PE*-FDs. Given a *PE*-FD $[\Delta_k(\tau_J^R)]X\ \overline{Y} \rightarrow \overline{Z}$, if set $\overline{Y} = \emptyset$ (i.e., the dependency is $[\Delta_k(\tau_J^R)]X \rightarrow \overline{Z}$), we will say that the *PE*-FD is *simple*. Moreover, if the considered *PE*-FD is of the type $[\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{X}$, we will say that the *PE*-FD is an *update*. *PE*-FDs featuring both the properties (i.e., *PE*-FDs of type $[\Delta_k(\tau_J^R)]X \rightarrow \overline{X}$) are called *simple updates*. A graphical account of such classes is given in Fig. 2.

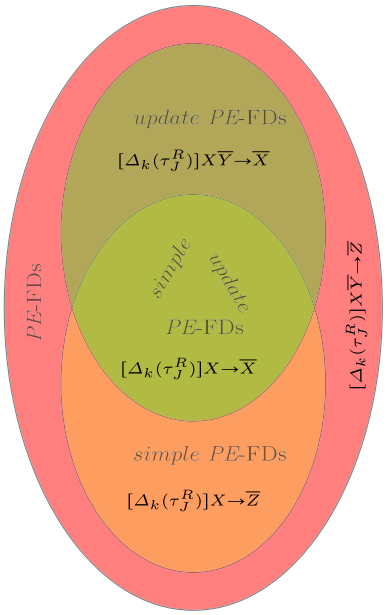


Fig. 2 A graphical account of how different classes of *PE*-FDs are related

Approximate Pure Temporally Evolving Functional Dependencies

We add approximation to *PE*-FDs in a very similar way we did for FDs in "Background and Related Work" section. First, we specialize the measurement G_3, which considers the minimum number of tuples in $\mathbf{r}$ to be deleted for the FD to hold, to deal with *PE*-FD as follows:

$$G_3([\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}, \mathbf{r}) = |\mathbf{r}| - \max\{|\mathbf{s}| : \mathbf{s} \subseteq \mathbf{r}, \mathbf{s} \vDash [\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}\}$$

By means of G_3, we can define the relative *scaled measurement* g_3 as follows:

$$g_3([\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}, \mathbf{r}) = \frac{G_3([\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}, \mathbf{r})}{|\mathbf{r}|}.$$

Now we are ready to define the *Approximate Pure Temporally Evolving Functional Dependency*.

Definition 5 (*Approximate Pure Temporally Evolving Functional Dependency*) An *Approximate Pure Temporally Evolving Functional Dependency* over the temporal schema $R = U \cup \{VT\}$, *APE*-FD for short, is an expression of the form:

$$[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$$

with $0 \leq \epsilon \leq 1$, $X \subseteq W$ and $\overline{Y}, \overline{Z} \subseteq \overline{W}$ with $X \neq \emptyset$ and $|Z| = 1$.

An instance $\mathbf{r}$ of R satisfies the *APE*-FD $[\Delta_k(\tau_J^R)]X\ \overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, written $\mathbf{r} \vDash [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, if and only if $g_3([\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}, \mathbf{r}) \leq \epsilon$.

Some Motivating Clinical Scenarios

In this section, we describe and discuss two scenarios, borrowed from the clinical domain, in order to provide examples of how *PE*-FDs and *APE*-FDs work. The first scenario is taken from psychiatric case register. Let us consider the temporal schema $Contact = \{Name, Phys, CT, Dur\} \cup \{VT\}$. Such a schema stores values about a phone-call service provided to psychiatric patients. This service is intended for monitoring and helping psychiatric patients, who are not hospitalized. Whenever a patient feels the need to talk to a physician, she can call the service. Data about calls are collected according to schema *Contact*. For the sake of simplicity, temporal attribute *VT* identifies the day when the call has been received. In addition, the service may be used by people somehow related to patients, as, for instance, relatives worried about the current condition of a patient.

More precisely, attribute *Name* identifies patients, *Phys* identifies physicians, *CT* (*Contact Type*) specifies the person who is doing the call (e.g., value "self" stands for the patient himself, "family" for a relative) and *Dur* stores information about total duration of calls (value $\sim n$ means approximately n minutes).

Fig. 3 An instance **r** of schema *Contact* that stores the phone contacts about two psychiatric cases. Attribute # represents the tuple number, and it is used only for referencing tuples in the text (i.e., # does not belong to the schema *Contact*)

$\mathbf{r} =$

#	*Name*	*Phys*	*CT*	*Dur(minutes)*	*VT*
1	*McMurphy*	*Sayer*	*self*	∼15	1 Jan 2016
2	*McMurphy*	*Sayer*	*family*	∼5	5 Jan 2016
3	*McMurphy*	*Maguire*	*family*	∼15	10 Jan 2016
4	*McMurphy*	*Maguire*	*self*	∼15	15 Jan 2016
5	*McMurphy*	*Maguire*	*self*	∼5	20 Jan 2016
6	*McMurphy*	*Maguire*	*self*	∼5	25 Jan 2016
7	*Lowe*	*Sayer*	*family*	∼15	7 Jan 2016
8	*Lowe*	*Sayer*	*family*	∼15	15 Jan 2016
9	*Lowe*	*Maguire*	*self*	∼15	22 Jan 2016
10	*Lowe*	*Sayer*	*self*	∼5	28 Jan 2016
11	*Lowe*	*Maguire*	*self*	∼15	3 Feb 2016
12	*Lowe*	*Sayer*	*self*	∼5	6 Feb 2016

$\tau^{\mathbf{r}}_{Name}$

#	*Name*	*Phys*	*CT*	*Dur*	*VT*	$\overline{Phys}$	$\overline{CT}$	$\overline{Dur}$	$\overline{VT}$
1, 2	*McMurphy*	*Sayer*	*self*	∼15	1 Jan 2016	*Sayer*	*family*	∼5	5 Jan 2016
2, 3	*McMurphy*	*Sayer*	*family*	∼5	5 Jan 2016	*Maguire*	*family*	∼15	10 Jan 2016
3, 4	*McMurphy*	*Maguire*	*family*	∼15	10 Jan 2016	*Maguire*	*self*	∼15	15 Jan 2016
4, 5	*McMurphy*	*Maguire*	*self*	∼15	15 Jan 2016	*Maguire*	*self*	∼5	20 Jan 2016
5, 6	*McMurphy*	*Maguire*	*self*	∼5	20 Jan 2016	*Maguire*	*self*	∼5	25 Jan 2016
7, 8	*Lowe*	*Sayer*	*family*	∼15	7 Jan 2016	*Sayer*	*family*	∼15	15 Jan 2016
8, 9	*Lowe*	*Sayer*	*family*	∼15	15 Jan 2016	*Maguire*	*self*	∼15	22 Jan 2016
9, 10	*Lowe*	*Maguire*	*self*	∼15	22 Jan 2016	*Sayer*	*self*	∼5	28 Jan 2016
10, 11	*Lowe*	*Sayer*	*self*	∼5	28 Jan 2016	*Maguire*	*self*	∼15	3 Feb 2016
11, 12	*Lowe*	*Maguire*	*self*	∼15	3 Feb 2016	*Sayer*	*self*	∼5	6 Feb 2016

Fig. 4 The evolution expression $\tau^{\mathbf{r}}_{Name}$

An instance **r** of R is provided in Fig. 3. Instance $\tau^{\mathbf{r}}_{Name}$, and $\tau^{\mathbf{r}}_{J}$ in general, may be seen as the output of a two-phase procedure. First, table *Contact* is partitioned into subsets of tuples, one for each value of *Name*. Then, each tuple is joined with its immediate successor in its partition, w.r.t. *VT* values. The whole relation $\tau^{\mathbf{r}}_{Name}$ is provided in Fig. 4. In the following, we will use t for referencing tuples of **r** and u for referencing tuples of $\tau^{\mathbf{r}}_{J}$. Moreover, in the following, each tuple u in $\tau^{\mathbf{r}}_{J}$ will be identified by the pair of indexes of the tuples in **r** that generate u. For instance, the first tuple of $\tau^{\mathbf{r}}_{J}$ in Fig. 4 will be denoted by $u_{1,2}$ since it is generated by the join of tuples t_1 and t_2 in **r**.

Going back to our example, it is worth noting that tuples t_2 and t_7 are not joined in $\tau^{\mathbf{r}}_{Name}$, even if $t_7[VT] = t_2[VT] + 2$ and there is no tuple t with $t[VT] = t_7[VT] + 1$. This is due to the fact that $t_7[Name] \neq t_2[Name]$ forbids the join in $\tau^{\mathbf{r}}_{Name}$. Moreover, t_1 and t_3 are not joined in $\tau^{\mathbf{r}}_{Name}$. Indeed, the presence of tuple t_2 with $t_1[Name] = t_2[Name] = t_3[Name]$ and $t_1[VT] < t_2[VT] < t_3[VT]$ forbids the join in $\tau^{\mathbf{r}}_{Name}$. Figure 5 graphically depicts how pairs of tuples (t_1, t_2), (t_2, t_3), (t_3, t_4), (t_4, t_5), (t_5, t_6) and (t_7, t_8), (t_8, t_9), (t_9, t_{10}), (t_{10}, t_{11}), (t_{11}, t_{12}) are joined in $\tau^{\mathbf{r}}_{Name}$ for the two patients, respectively. Basically, each tuple $u \in \tau^{\mathbf{r}}_{Name}$ corresponds to an edge in Fig. 5, while we have a node for each tuple in **r**.

Let us now discuss some temporal dependencies we can derive from such data. We could be interested in verifying whether there is some relationship between some previous features of patient's call and the fact that the considered call was either with him or with a relative. In our example, we have that $\mathbf{r} \vDash [\Delta_5(\tau^{Contact}_{Name})]Phys, \overline{Phys} \rightarrow \overline{CT}$.

In other words, given consecutive calls related to the same patient within 5 days, the couple composed by the physician of the first call and by the physician of the next one determines the type of contact of the next call. And it holds for all patients. However, if we consider a wider time window of 6 days, we have that $\mathbf{r} \nvDash [\Delta_6(\tau^{Contact}_{Name})]Phys, \overline{Phys} \rightarrow \overline{CT}$, because of pairs (t_2, t_3) and (t_{10}, t_{11}). More precisely, we have that $t_2[Name] = t_3[Name] =$ "*McMurphy*", $t_{10}[Name] = t_{11}[Name] =$ "*Lowe*", $t_2[Phys] = t_{10}[Phys] =$ "*Sleepy*", $t_3[Phys] = t_{11}[Phys] =$ "*Patel*", but $t_3[CT] \neq t_{11}[CT]$ (i.e., $t_3[CT] =$ "*family*", and $t_{11}[CT] =$ "*self*"). In other words, we have that the set of tuples $\{u_{2,3}, u_{10,11}\}$ does not satisfy the FD $Phys, \overline{Phys} \rightarrow \overline{CT}$.

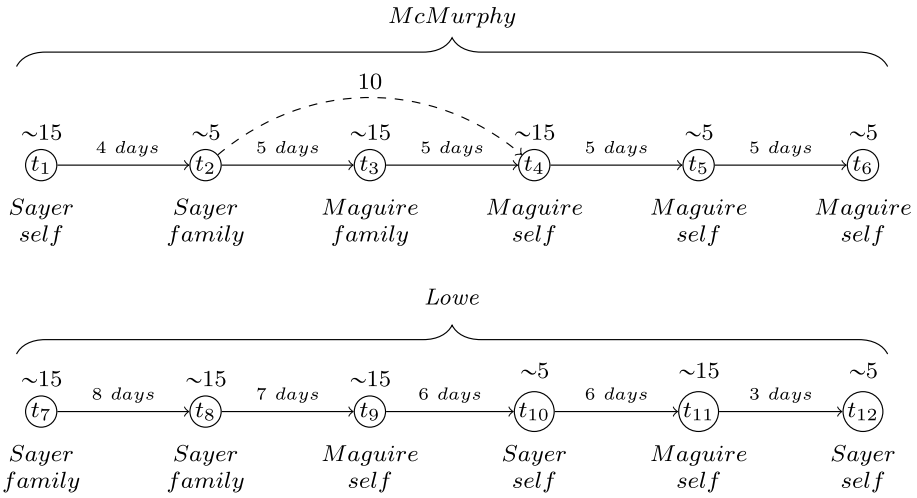


Fig. 5 A graph-based representation of $\tau^{\mathbf{r}}_{Name}$. Nodes represent tuples and are labeled by the corresponding tuple number. Values for attribute *Dur* are reported above each node. Values of *Phys* and *CT* attributes are reported below every node, respectively. Every edge (t_i, t_j) is labeled by value $\Delta(u_{i,j}) = t_j[VT] - t_i[VT]$ (i.e., the temporal distance between two tuples). The dashed edge represents a different scenario where t_2 and t_4 are joined, if t_3 would be deleted, as explained below for *APE*-FDs

The two proposed *PE*-FDs differ only for the maximum temporal distance allowed. In particular, tuple $u_{10,11}$ is one of the responsible ones for $\mathbf{r} \not\models [\Delta_6(\tau^{Contact}_{Name})]\ Phys, \overline{Phys} \rightarrow \overline{CT}$, but it does not belong to $\Delta_5(\tau^{Contact}_{Name})$ because $\Delta(u_{10,11}) > 5$. This allows us to point out a general property of *PE*-FDs and of *APE*-FDs too. Given a *PE*-FD $[\Delta_k(\tau^R_J)]X\overline{Y} \rightarrow \overline{Z}$, if we have that for every instance $\mathbf{r}$ of R it holds $\mathbf{r} \models [\Delta_k(\tau^R_J)]X\overline{Y} \rightarrow \overline{Z}$, then for every $h \leq k$ it holds $\mathbf{r} \models [\Delta_h(\tau^R_J)]X\overline{Y} \rightarrow \overline{Z}$.

Moving to the problem of mining approximate dependencies, if we consider *APE*-FD $[\Delta_6(\tau^R_{Name})]Phys, \overline{Phys} \xrightarrow{\epsilon} \overline{CT}$ with $\epsilon = \frac{1}{12}$, we have that $\mathbf{r} \models [\Delta_6(\tau^R_{Name})]\ Phys, \overline{Phys} \xrightarrow{\epsilon} \overline{CT}$. Indeed, by considering relation $\mathbf{r}' = \mathbf{r} \setminus \{t_3\}$, this dependency would hold without the need of approximation (i.e., if tuple t_3 is deleted from relation $\mathbf{r}$). More precisely, we have $\tau^{\mathbf{r}'}_{Name} = \tau^{\mathbf{r}}_{Name} \setminus \{u_{2,3}, u_{3,4}\} \cup \{u_{2,4}\}$. Tuple $u_{2,4}$ was not originally in $\tau^{\mathbf{r}}_{Name}$ because of tuple t_3. Figure 5 depicts this new scenario, by replacing edges (t_2, t_3) and (t_3, t_4) with the dashed edge (t_2, t_4). Moreover, we have $\mathbf{r}' \models [\Delta_{+\infty}(\tau^R_{Name})]Phys, \overline{Phys} \xrightarrow{\epsilon} \overline{CT}$. Thus, it holds $\mathbf{r} \models [\Delta_{+\infty}(\tau^R_{Name})]Phys, \overline{Phys} \xrightarrow{\epsilon} \overline{CT}$ with $\epsilon = \frac{1}{12}$.

The second example we propose is borrowed from the internal medicine domain. As another simple example of how view τ^R_J works, let us consider the temporal schema $ThCy = \{PatId, Phys, Dos\} \cup \{VT\}$. Such a schema allows one to store the values about cycles of therapies in which a specific, fixed, drug is administered to a patient by a given physician. Figure 6a depicts an instance $\mathbf{r}$ of R. Figure 6b shows the result of view τ^{ThCy}_{PatId} to $\mathbf{r}$. It is easy to see that tuples t_1 and t_5 are not joined in $\tau^{\mathbf{r}}_{PatId}$. Even if $t_5[VT] = t_1[VT] + 1$ the fact that $t_1[PatId] \neq t_5[PatId]$ forbids the join in $\tau^{\mathbf{r}}_{PatId}$. Moreover t_1 and t_3 are not joined in $\tau^{\mathbf{r}}_{PatId}$. Even if $t_1[PatId] = t_3[PatId]$, we have that the presence of tuple t_2 with $t_1[PatId] = t_2[PatId] = t_3[PatId]$ and $t_1[VT] < t_2[VT] < t_3[VT]$ forbids the join in $\tau^{\mathbf{r}}_{PatId}$. In Fig. 7a, we have a graphical account of how the pairs of tuples $(t_1, t_2), (t_2, t_3), (t_3, t_4), (t_5, t_6), (t_6, t_7)$ and (t_7, t_8) are joined in $\tau^{\mathbf{r}}_{PatId}$. In both the graphs depicted in Fig. 7a, nodes are labeled with the tuple number and the value for *Dos* attribute is reported above each node. Moreover, we recall from the previous example that each edge (t_i, t_j) is labeled with the value $t_j[VT] - t_i[VT]$ (i.e., the temporal distance between two tuples). In the scenario for $\tau^{\mathbf{r}}_{PatId}$ the value for the attribute *Phys* is reported below each node, while for $\tau^{\mathbf{r}}_{Phys}$ the value of *PatId* is reported below each node.

We would like to point out that it may be the case that a tuple $t \in \mathbf{r}$ has a more than one immediate successor in $\tau^{\mathbf{r}}_J$ (i.e. is joined with more than one tuple). It is the case of view $\tau^{\mathbf{r}}_{Phys}$ shown in Fig. 7b, where tuples are joined with respect to the values of attribute *Phys*. We have that, since Dr. *Shepherd* makes two drug administrations at $VT = 20$, tuple t_1 has both tuples t_3 and t_7 as its immediate successors. We will see that the number of immediate successors of a tuple in $\tau^{\mathbf{r}}_J$ will play a major role in some of the following complexity results.

In this domain, we could be interested in understanding whether there are dependencies among previous and current drug dosages for a given patient, possibly considering the physicians administering the drug.

In the example depicted in Figs. 6 and 7, we have that $\mathbf{r} \models [\Delta_{+\infty}(\tau^R_{PatId})]\ Dos, Phys, \overline{Phys} \rightarrow \overline{Dos}$. It means that the dosage and the couple of physicians related to a drug administration and to the next one, respectively, determine the next drug dosage. However, $\mathbf{r} \not\models [\Delta_{\infty}(\tau^R_{PatId})]Phys, \overline{Phys} \rightarrow \overline{Dos}$ because both tuples $u_{1,2}$ and $u_{7,8}$ belong to $\Delta_{\infty}(\tau^R_{PatId})$. More precisely, we have that $Phys, \overline{Phys} \rightarrow \overline{Dos}$ does not hold on any instance of $\Delta_k(\tau^R_{PatId})$ that contains both $u_{1,2}$ and $u_{7,8}$, since $u_{1,2}[Phys] = u_{7,8}[Phys] =$ "*Shepherd*" and $u_{1,2}[\overline{Phys}] = u_{7,8}[\overline{Phys}] =$ "*Stevens*" but $u_{1,2}[\overline{Dos}] =$ "60 *mg*" is not equal to $u_{7,8}[\overline{Dos}] =$ "20 *mg*".

Fig. 6 The instances $\tau^{\mathbf{r}}_{PatId}$ (**b**) and $\tau^{\mathbf{r}}_{Phys}$ (**c**) obtained from applying views τ^{ThCy}_{PatId} and τ^{ThCy}_{Phys} to the instance **r** (**a**), respectively

(a) $\mathbf{r} =$

#	$PatId$	$Phys$	Dos	VT
1	1	*Shepherd*	30 *mg*	1 May 2016
2	1	*Stevens*	60 *mg*	5 May 2016
3	1	*Shepherd*	40 *mg*	20 May 2016
4	1	*Shepherd*	40 *mg*	27 May 2016
5	2	*Stevens*	30 *mg*	2 May 2016
6	2	*Stevens*	40 *mg*	9 May 2016
7	2	*Shepherd*	60 *mg*	20 May 2016
8	2	*Stevens*	20 *mg*	25 May 2016

(b) $\tau^{\mathbf{r}}_{PatId} =$

#	$PatId$	$Phys$	Dos	VT	$\overline{Phys}$	$\overline{Dos}$	$\overline{VT}$
1, 2	1	*Shepherd*	30 *mg*	1 May 2016	*Stevens*	60 *mg*	5 May 2016
2, 3	1	*Stevens*	60 *mg*	5 May 2016	*Shepherd*	40 *mg*	20 May 2016
3, 4	1	*Shepherd*	40 *mg*	20 May 2016	*Shepherd*	40 *mg*	27 May 2016
5, 6	2	*Stevens*	30 *mg*	2 May 2016	*Stevens*	40 *mg*	9 May 2016
6, 7	2	*Stevens*	40 *mg*	9 May 2016	*Shepherd*	60 *mg*	20 May 2016
7, 8	2	*Shepherd*	60 *mg*	20 May 2016	*Stevens*	20 *mg*	25 May 2016

(c) $\tau^{\mathbf{r}}_{Phys} =$

#	$Phys$	$PatId$	Dos	VT	$\overline{PatId}$	$\overline{Dos}$	$\overline{VT}$
1, 3	*Shepherd*	1	30 *mg*	1 May 2016	1	40 *mg*	20 May 2016
1, 7	*Shepherd*	1	30 *mg*	1 May 2016	2	60 *mg*	20 May 2016
3, 4	*Shepherd*	1	40 *mg*	20 May 2016	1	40 *mg*	27 May 2016
7, 4	*Shepherd*	2	60 *mg*	20 May 2016	1	40 *mg*	27 May 2016
5, 2	*Stevens*	2	30 *mg*	2 May 2016	1	60 *mg*	5 May 2016
2, 6	*Stevens*	1	60 *mg*	5 May 2016	2	40 *mg*	9 May 2016
6, 8	*Stevens*	2	40 *mg*	9 May 2016	2	20 *mg*	25 May 2016

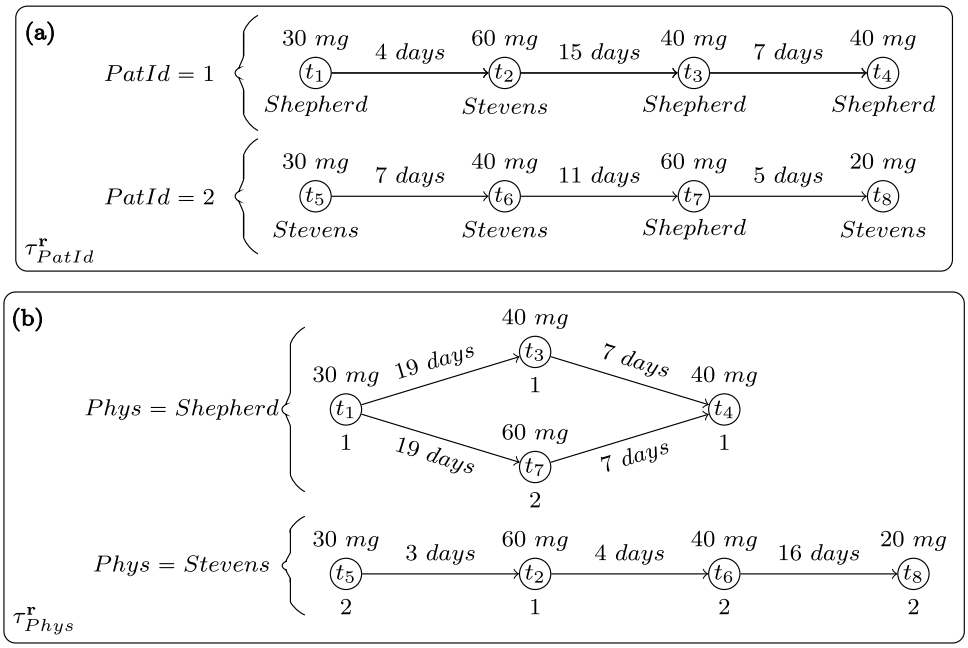


Fig. 7 A graphical account for the instances $\tau^{\mathbf{r}}_{PatId}$ (**a**) and $\tau^{\mathbf{r}}_{Phys}$ (**b**) related to the instance **r** shown in Fig. 6a

On the other hand, $\mathbf{r} \models [\Delta_4(\tau^R_{PatId})]Phys, \overline{Phys} \rightarrow \overline{Dos}$ because having 4 as maximum allowed time distance implies $u_{7,8} \notin \Delta_4(\tau^{\mathbf{r}}_{PatId})$ (i.e., $t_8[VT] - t_7[VT] > 4$) and thus the conflict between $u_{1,2}$ and $u_{7,8}$ no longer exists. Furthermore, it is easy to see by considering the pairs $u_{1,2}$ and $u_{5,6}$ that $\mathbf{r} \setminus \models [\Delta_{+\infty}(\tau^R_{PatId})]Dos, \overline{Phys} \rightarrow \overline{Dos}$. However, by shrinking the maximum allowed time distance to 6, we obtain $u_{5,6} \notin \Delta_6(\tau^{\mathbf{r}}_{PatId})$ and thus we have that $\mathbf{r} \models [\Delta_6(\tau^R_{PatId})]Dos, Phys \rightarrow Dos$.

Obviously, in an instance $\tau^{\mathbf{r}}_j$, there may be more than one pair (u, u') which generates a conflict. Let us consider the simple update *PE*-FD $[\Delta_{+\infty}(\tau^R_{PatId})]\ Dos \rightarrow \overline{Dos}$. Such *PE*-FD does not hold on **r** (i.e., $\mathbf{r} \setminus \models [\Delta_{+\infty}(\tau^R_{PatId})]Dos \rightarrow \overline{Dos}$). This can be shown using the pair $u_{1,2}$ and $u_{5,6}$ as a witness for a conflict since $u_{1,2}[Dos] = u_{5,6}[Dos] =$ "30 *mg*" and

$u_{1,2}[\overline{Dos}] \neq u_{5,6}[\overline{Dos}]$. However, in this case, we have that the conflicting pairs are more than one. As a matter of fact, all pairs $(u_{1,2}, u_{5,6})$, $(u_{2,3}, u_{7,8})$ and $(u_{3,4}, u_{4,6})$ are conflict-generating. If we want to rule all the conflicts out by playing on the maximum allowed distance, we have to set it to 6 and then we have $\mathbf{r} \vDash [\Delta_6(\tau^R_{PatId})]Dos \rightarrow \overline{Dos}$.

The Computational Complexity of Checking *APE*-FD

In this section, we address the complexity of checking an *APE*-FD against an instance $\mathbf{r}$. We call this problem *Check-APE*-FD:

Problem 1 (Check-*APE*-FD). Given a temporal schema R, a *PE*-FD $[\Delta_k(\tau^R_J)]\, X\overline{Y} \rightarrow \overline{Z}$ on R, an instance $\mathbf{r}$ of R, and a real number $0 \leq \epsilon \leq 1$, determine whether $\mathbf{r} \vDash [\Delta_k(\tau^R_J)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ or not.

Let us consider, for example, the *PE*-FD $[\Delta_{+\infty}(\tau^{ThCy}_{PatId})]Phys, \overline{Phys} \rightarrow \overline{Dos}$. We have proved above that $\mathbf{r} \setminus \nvDash fd$. Figure 8 graphically reports all the possible $\tau^{\mathbf{r}'}_{PatId}$ where $\mathbf{r}'$ is obtained from $\mathbf{r}$ by deleting exactly one tuple. For example, if $\mathbf{r}' = \mathbf{r}\setminus\{t_1\}$, it means that the dotted edge (t_1, t_2) has been removed. This means that t_1 and t_2 are not joined in $\tau^{\mathbf{r}'}_{PatId}$. Moreover, if we take $\mathbf{r}' = \mathbf{r}\setminus\{t_2\}$ we have that both the edges (t_1, t_2) and (t_2, t_3) are removed and the dashed edge (t_1, t_3) turns out to be "active." This means that t_1 and t_2 are not joined in $\tau^{\mathbf{r}'}_{PatId}$ as well as t_2 and t_3, but t_1 and t_3 turn out to be joined in $\tau^{\mathbf{r}'}_{PatId}$ due to the absence of t_2. Let us observe that in this case, the join operation involving t_1 and t_3 belongs to $\tau^{\mathbf{r}'}_{PatId}$ and not to $\tau^{\mathbf{r}}_{PatId}$. This specific behavior, in which the deletion of a tuple introduces additional, possibly different, constraints as a side effect, instead of just removing existing ones, gives us a hint on the problem *Check- APE*-FD. Such problem is not so easy to solve. Notice that $\mathbf{r}\setminus\{t_1\} \setminus \nvDash [\Delta_{+\infty}(\tau^{ThCy}_{PatId})]Phys, \overline{Phys} \rightarrow \overline{Dos}$ as well, because of the pairs (t_2, t_3) and (t_6, t_7). However, $\mathbf{r}\setminus\{t_2\} \vDash [\Delta_{+\infty}(\tau^{ThCy}_{PatId})]Phys, \overline{Phys} \rightarrow \overline{Dos}$ and thus we have that $\mathbf{r} \vDash [\Delta_{+\infty}(\tau^{ThCy}_{PatId})]Phys, \overline{Phys} \overset{\epsilon}{\rightarrow} \overline{Dos}$ with $\epsilon = \frac{1}{8}$.

Therefore, problem *Check- APE* -FD belongs to the complexity class *NP*. In order to prove that, it suffices to apply a guess-and-check algorithm. First, this algorithm guesses a set $\mathbf{r}'$ with $|\mathbf{r}'| \leq \epsilon \cdot |\mathbf{r}|$. Then, if $\mathbf{r}\setminus\mathbf{r}' \vDash [\Delta_k(\tau^R_J)]X\overline{Y} \rightarrow \overline{Z}$, the algorithm returns *YES*, otherwise *NO*. In the procedure above, we implicitly make use of a function that verifies, given an instance $\mathbf{r}$ of R and a *PE*-FD $[\Delta_k(\tau^R_J)]X\overline{Y} \rightarrow \overline{Z}$, whether $\mathbf{r} \vDash [\Delta_k(\tau^R_J)]X\overline{Y} \rightarrow \overline{Z}$ holds or not. We can call this problem *Check- PE*-FD. Since there is no approximation, checking if $\mathbf{r} \vDash [\Delta_k(\tau^R_J)]X\overline{Y} \rightarrow \overline{Z}$ may be performed in polynomial time [7]. For this reason, we can conclude that *Check-APE*-FD belongs to the complexity class NP. In the following, we will prove that Check-*APE*-FD is NP-hard even in the case of the most constrained kind of *APE*-FDs, which is represented by the class of simple update *APE*-FDs. From now on, we will consider Problem 1 only for simple update *APE*-FDs. Considering the inclusions shown in Fig. 2, we can immediately conclude that our hardness result directly propagates to the other classes of *APE*-FDs.

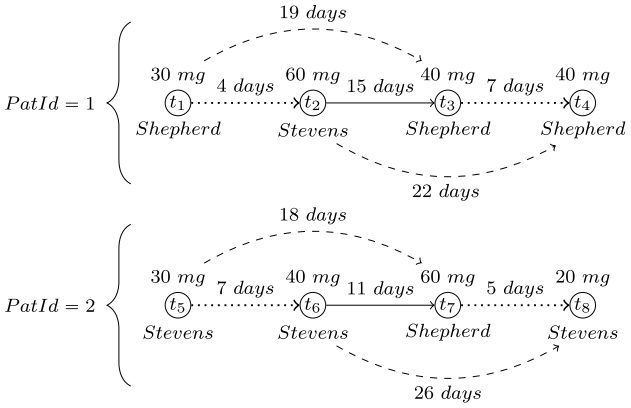


Fig. 8 A graphical account for the possible changes on the view $\tau^{\mathbf{r}}_{PatId}$ considering the possible deletions of at most one tuple

In this section, we will make use of finite words w on a finite nonempty alphabet Σ (i.e., $w \in \Sigma^*$). We will use the standard notation $w[i]$ for denoting the ith symbol of word w. Given a word w, we denote with $first(w)$ and $last(w)$ its first and its last element, respectively (i.e., $first(w) = w[1]$ and $last(w) = w[|w|]$). Moreover, a *finite increasing sequence of* $\mathbb{N}$ ($\mathbb{N}^>$-sequence) is a finite word s on $(\mathbb{N}\setminus\{0\})^*$, where for every i, i', with $1 \leq i < i' \leq |s|$, we have $s[i] < s[i']$.[1] Given a $\mathbb{N}^>$-sequence s we denote with $first(s)$ and $last(s)$ its first and its last element, respectively (i.e., $first(s) = s[1]$ and $last(s) = s[|s|]$). A $\mathbb{N}^>$-sequence s, for which for every i, with $1 \leq i < |s|$, we have $w[i+1] = w[i] + 1$, is called *strict* and we denote it with $[b, e]$, where $b = first(s)$ and $e = last(s)$. Given a word w and a $\mathbb{N}^>$-sequence s, we denote with $w\|_s$ the word $w\|_s = w[first(s)] \ldots w[last(s)]$ (for a graphical account of how a word is filtered by a sequence please refer to Fig. 9). Given a word w and a pair (b, e), with $1 \leq b \leq e \leq |w|$, we call the word $w\|_{[b,e]}$ a *slice* of w. Given two words $w_1, w_2 \in \Sigma^*$ we say that

[1] a $\mathbb{N}^>$-sequence is nothing more than a different representation for a finite set of positive naturals, but it turns out for our purposes to see it as a particular kind of word over positive naturals.

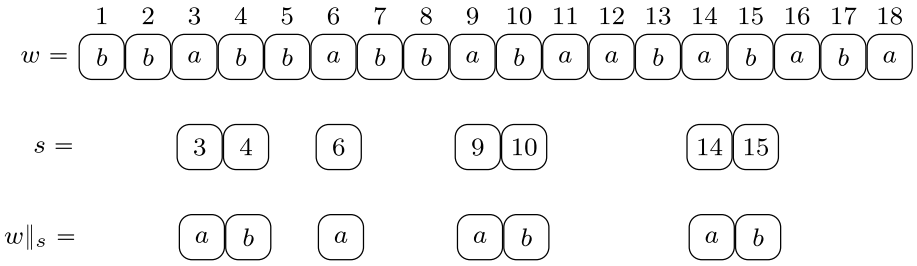


Fig. 9 An example of a word $w\|_s$ obtained by applying a sequence s to a word w

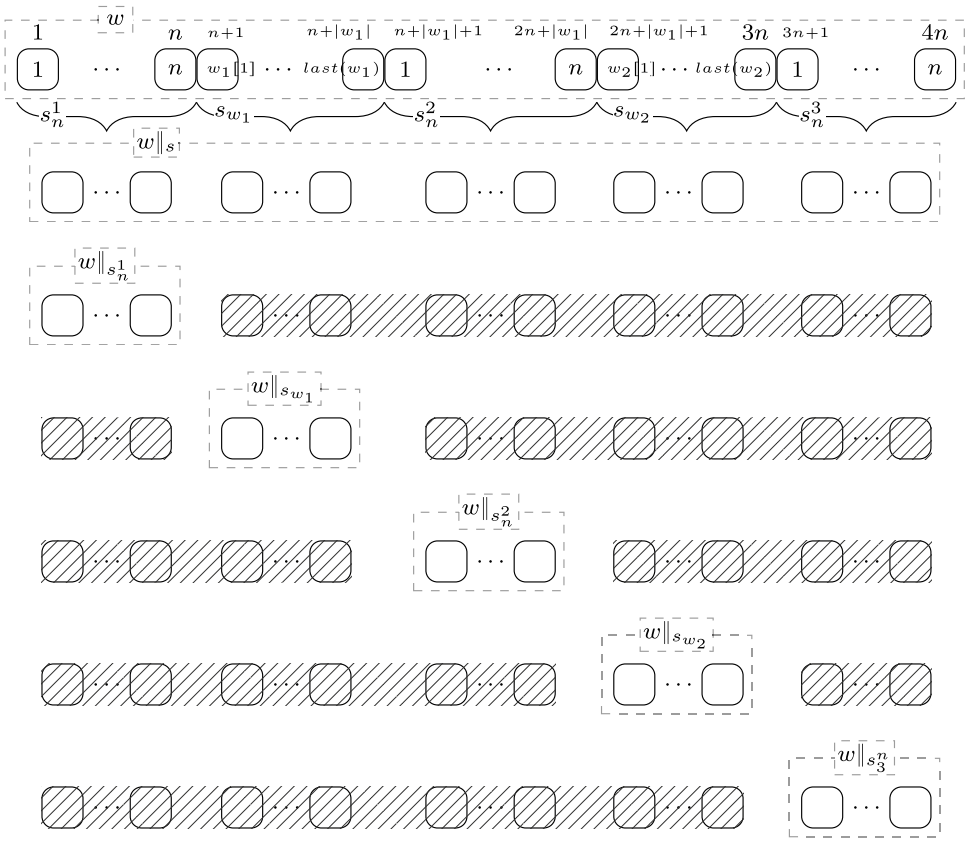


Fig. 10 A graphical account of how $s_n^1, s_{w_1}, s_n^2, s_{w_2}$, and s_n^3 filter blocks of w

w_1 is a subsequence of w_2 (written $w_1 \sqsubseteq w_2$) if and only if there exists an $\mathbb{N}^>$-sequence s for which $w_1 = w_2\|_s$. For instance, $w_1 = abaabba$ is a subsequence of $w_2 = bbabbabbabaababababa$ with $s = 3\ 4\ 6\ 9\ 10\ 14\ 15$. A word w is *repetition free* if and only if for every $a \in \Sigma$ we have $|\{i : w[i] = a\}| \leq 1$. A word w is a *permutation* of Σ if and only if w is repetition free and $|w| = |\Sigma|$.

The proof that *Check- APE*-FD is NP-hard is done in two steps. First, we describe a known NP-Complete problem called *Common Permutation Problem* (*CP*-P for short). Then, we introduce a problem called *Periodic Repair Problem* (*PR*-P) and we prove that *CP*-P may be reduced to it using logarithmic space. Finally, we reduce the *PR*-P to Check-*APE* -FD using logarithmic space.

Let us begin with the Common Permutation Problem which has been proved to be NP-Complete in [12].

Problem 2 (*CP*-P). Given a finite alphabet Σ and two words w_1, w_2 over it, is there a permutation w_p of Σ for which $w_p \sqsubseteq w_1$ and $w_p \sqsubseteq w_2$?

Consider $\Sigma = \{a, b, c\}$ we have that the pair $w_1 = bcbab$ and $w_2 = accaacb$ is a positive instance of Problem 2 because $cab \sqsubseteq w_1$ and $cab \sqsubseteq w_2$. On the other hand, the pair $w_1 = bcbac$ and $w_2 = acab$ is a negative instance of Problem 2 since both words do not share any permutation of Σ as their subsequence. More precisely, w_1 contains the permutations *bca*, *bac* and *cba* while w_2 contains the permutations *acb* and *cab*.

A word w is periodic if and only if for every pair of indexes (i, i'), with $1 \leq i, i' < |w|$, we have that $w[i] = w[i']$ implies $w[i + 1] = w[i' + 1]$. Let us observe that if w is repetition-free, then it is periodic. Moreover, if w is periodic for

every pair (b, e), with $1 \leq b \leq e \leq |w|$, we have that $w\|_{[b,e]}$ is periodic (i.e., every slice of a periodic word is itself periodic). The following lemma turns out to be useful for our reduction.

Lemma 1 *Given a periodic word w, if w is not repetition-free, then there exists an index $i < |w|$ such that $last(w) = w[i]$.*

Proof Since w is not repetition free, there exists two indexes i, i', with $1 \leq i < i' \leq |w|$, such that $w[i] = w[i']$. We prove the claim by induction on $\Delta = |w| - i'$. For the base of the induction, we have $\Delta = 0$ and thus the claim trivially holds since i is the index we were looking for. Let us consider $\Delta = n + 1$. Since w is periodic and $w[i] = w[i']$, we have that $w[i+1] = w[i'+1]$. Thus, positions $i+1$ and $i'+1$ witness a repetition and since $|w| - (i'+1) < |w| - i' = \Delta$ we can apply the inductive hypothesis and prove our claim. □

In order to prove that *Check- APE* -FD is NP-Complete even for simple update *PE*-FD $[\Delta_k(\tau_J^R)]X \rightarrow \overline{X}$, we introduce the following intermediate problem called *Periodic Repair Problem* (*PR*-P for short):

Problem 3 (*PR*-P) Given a word $w = a_1 \dots a_n$, a finite alphabet Σ and a natural number k, determine whether a periodic word $w' \sqsubseteq w$ exists such that $|w'| \geq k$.

Problem 3 belongs to the complexity class NP. A simple nondeterministic algorithm for *PR*-P guesses an $\mathbb{N}^>$ -sequence s such that $|s| \geq k$ and $last(s) \leq |w|$ (i.e., s "chooses" only positions in $1 \dots |w|$). Then, it suffices to check whether or not $w\|_s$ is periodic (periodicity checking may be performed in logarithmic space).

In the following, we describe how to reduce *CP*-P to *PR* -P. Let us consider two words w_1 and w_2 on an alphabet Σ with length n_1 and n_2, respectively. We assume without loss of generality that Σ is a finite subset of the negative integers (i.e., $\Sigma \subseteq \mathbb{Z}^-$). Let $n = \max(n_1, n_2)$ and $\sigma = |\Sigma|$. Let us consider the following word w over the alphabet $\Sigma \cup \{1, \dots, n\}$ ($\cdot$ is the classical word concatenation operator):

$$w = 1 \cdot \dots \cdot n \cdot w_1 \cdot 1 \cdot \dots \cdot n \cdot w_2 \cdot 1 \cdot \dots \cdot n.$$

Finally, we put $k = 3n + 2\sigma$. Such reduction operates in logarithmic space. The following two lemmas prove the soundness and completeness of the above reduction.

Lemma 2 *If there exists a permutation w_p of Σ which is a common subsequence of w_1, w_2, then there exists a $\mathbb{N}^>$-sequence s, with $|s| \geq 3n + 2\sigma$ and $last(s) \leq 3n + |w_1| + |w_2|$, such that $w\|_s$ is periodic.*

Proof First, let us recall that w_p is a repetition-free sequence of symbols in $\mathbb{Z}^-$. By hypothesis, we have $w_p \sqsubseteq w_1$ and $w_p \sqsubseteq w_2$ and thus there exists a pair of $\mathbb{N}^>$-sequence s_1 and s_2 with $|s_1| = \sigma, |s_2| = \sigma$ and $w_1\|_{s_1} = w_2\|_{s_2} = w_p$.

Let $\overline{s}_j$ with $j \in \{1, 2\}$ be the $\mathbb{N}^>$-sequence such that $|\overline{s}_j| = \sigma$ and for every $1 \leq i \leq \sigma$, we have $\overline{s}_j[i] = s_j[i] + nj + \sigma(j-1)$. Let us observe that $\overline{s}_j$ is a simple shift of the indexes in the sequence s_j with $j \in \{1, 2\}$. Then, we may define s as follows:

$$s = \begin{array}{l}[1, n] \cdot \overline{s}_1 \cdot [n + |w_1| + 1, 2n + |w_1|] \cdot \overline{s}_2 \\ \cdot [2n + |w_1| + |w_2| + 1, 3n + |w_2| + |w_1|]\end{array}$$

By construction, we have $w\|_s = 1 \dots n \cdot w_p \cdot 1 \dots n \cdot w_p \cdot 1 \dots n$. Since $\Sigma \cap \{1, \dots n\} = \emptyset$ there are not "conflicts" between the blocks $1 \dots n$ and w_p, we can conclude that $w\|_s$ is periodic. □

Lemma 3 *If there exists a sequence s with $3n + 2\sigma \leq |s| \leq 3n + |w_1| + |w_2|$ for which $w\|_s$ is periodic, then there exists a permutation w_p of Σ which is a common subsequence of w_1, w_2.*

Proof First, we define $s_n^1, s_{w_1}, s_n^2, s_{w_2}, s_n^3$ such that $s = s_n^1 \cdot s_{w_1} \cdot s_n^2 \cdot s_{w_2} \cdot s_n^3$ and $last(s_n^1) \leq n$, $n + 1 \leq s_{w_1}[1] \leq last(s_{w_1}) \leq n + |w_1|$, $n + |w_1| + 1 \leq s_n^2[1] \leq last(s_n^2) \leq 2n + |w_1|$, $2n + |w_1| + 1 \leq s_{w_2}[1] \leq last(s_{w_2}) \leq 2n + |w_1| + |w_2|$ and $n + |w_1| + |w_2| + 1 \leq s_n^3[1]$. Informally $s_n^1, s_{w_1}, s_n^2, s_{w_2}, s_n^3$ are the indexes in s that concern the subwords $1 \dots n$ (first block), $w_1, 1 \dots n$ (second block), w_2 and $1 \dots n$ (third block) respectively. A graphical account of this decomposition of $w\|_s$ is given in Fig. 10. This means that we may retrieve the subsequence selected by s on w restricted to the first block by means of the operation $w\|_{s_n^1}$. If we want to retrieve the subsequence selected by s on w restricted to the w_1 block, we write $w\|_{s_{w_1}}$, and so on, for the second block $1 \dots n$ (i.e., $w\|_{s_n^2}$), the block w_2 (i.e., $w\|_{s_{w_2}}$) and the third block $1 \dots n$ (i.e., $w\|_{s_n^3}$). Let us notice that $w\|_s$ is equal to $w\|_{s_n^1} \cdot w\|_{s_{w_1}} \cdot w\|_{s_n^2} \cdot w\|_{s_{w_2}} \cdot w\|_{s_n^3}$.

Suppose by contradiction that $|s_n^2| = 0$. Then, we have that $w' = w\|_{s_{w_1}} \cdot w\|_{s_{w_2}}$ is a slice of $w\|_s$ and thus w' is periodic. Moreover, we have that $w'' = w' \cdot (w\|_{s_n^3})$ is a slice of $w\|_s$ and thus w'' is periodic. Two cases may arise, i.e., either $|s_n^3| = 0$ or not.

If $|s_n^3| = 0$ we have that $w\|_s \sqsubseteq 1 \dots n \cdot w'$ and by a counting argument we have that $|w\|_s| \leq 3n$ which is a contradiction since $3n + 2\sigma \leq |w\|_s|$ and $\sigma > 0$ by definition.

If $|s_n^3| > 0$, we prove that w'' is repetition free. Again by contradiction, from Lemma 1, we will have that, since w'' is periodic, there must exist an index $i < |w''|$ for which $w''[i] = w''[n]$. However, $|s_n^3| > 0$ implies $w''[n] \in \{1, \dots, n\}$ and thus, since $\Sigma \cap \{1, \dots, n\} = \emptyset$, such i' cannot exist

(contradiction). If w'' is repetition free, we have that $|w''| \leq \sigma + n$, and since $w\|_s \sqsubseteq 1 \dots n \cdot w''$, we have that $|w\|_s| \leq 2n + \sigma$, which contradicts $|w\|_s| \geq 3n + 2\sigma$.

We have now that $|s_n^2| > 0$. Consider the slice $w' = w\|_{s_{w_1}} \cdot w\|_{s_n^2}$: we have just proved that $|w\|_{s_n^2}| > 0$ and we have that w' is periodic being a slice of $w\|_s$. By applying Lemma 1 as we did above, we can claim that $w\|_{s_{w_1}}$ is repetition free and thus $|w\|_{s_{w_1}}| \leq \sigma$. Suppose now by contradiction that $w\|_{s_{w_2}}$ is not repetition free. If $|s_n^3| > 0$ we reach immediately a contradiction by applying Lemma 1 on the word $w\|_{s_{w_2}} \cdot w\|_{s_n^3}$. Then, we have $|s_n^3| = 0$ and by definition $|w\|_{s_{w_2}}| \leq n$. This implies $w\|_s \sqsubseteq 1 \dots n \cdot w\|_{s_{w_1}} \cdot 1 \dots n \cdot w\|_{s_{w_2}}$ which means $|w\|_s| \leq 3n + \sigma$ (contradiction).

At this point, we have that both $w\|_{s_{w_1}}$ and $w\|_{s_{w_2}}$ are repetition free and thus $|w\|_{s_{w_1}}| \leq \sigma$ and $|w\|_{s_{w_2}}| \leq \sigma$. Since $3n + 2\sigma \leq |w\|_s|$, we have that $|w\|_{s_{w_1}}| = \sigma$ and $|w\|_{s_{w_2}}| = \sigma$ and thus both $w\|_{s_{w_1}}$ and $w\|_{s_{w_2}}$ are permutations of Σ. It remains to prove that they are the same permutation. Let us observe that, since $|w\|_{s_{w_1}}| = |w\|_{s_{w_2}}| = \sigma$ and $|w\|_s| \geq 3n + 2\sigma$, by a counting argument we have that $w\|_{s_n^1} = w\|_{s_n^2} = w\|_{s_n^3} = 1 \dots n.$ and thus $w\|_s = 1 \dots n \cdot w\|_{s_{w_1}} \cdot 1 \dots n \cdot w\|_{s_n^2} \cdot 1 \dots n.$

Suppose by contradiction that there exists i, with $1 \leq i \leq \sigma$, such that $w\|_{s_{w_1}}[i] \neq w\|_{s_{w_2}}[i]$, and let i be the minimum index that fulfills such a property. Two cases may arise:

1. If $i = 1$ we have that $w\|_s[n+1] = w\|_{s_{w_1}}[i]$ and $w\|_s[2n + \sigma + 1] = w\|_{s_{w_2}}[i]$. Let us recall that $w\|_s[n] = w\|_s[2n + \sigma] = n$, and thus by periodicity of $w\|_s$, we have $w\|_s[n+1] = w\|_s[2n + \sigma + 1]$ (contradiction).
2. If $i > 1$ since $w\|_s$ is periodic, we have that $w\|_{s_{w_1}}[i-1] \neq w\|_{s_{w_2}}[i-1]$ but this contradicts the minimality in the choice of i. □

Now we reduce Problem *PR*-P to *Check- APE* -FD in logarithmic space. Suppose that we have an instance of *PR*-P consisting of a word $w \in \Sigma^*$ and a natural number k. We define the instance $\mathbf{r}_w$ on the temporal schema $R = \{J, X\} \cup VT$ as $\mathbf{r}_w = \{t \mid t[J] = 0 \wedge \exists i(t[X] = w[i] \wedge t[VT] = i)\}$, and we put $\epsilon_{w,k} = \frac{|w|-k}{|w|}$. The pair w, k is a positive instance of *PR*-P if and only if the triple $[\Delta_{+\infty}(\tau_J^R)]X \to \overline{X}$, $\mathbf{r}_w$ and $\epsilon_{w,k}$ is a positive instance of *Check-APE* -FD. $[\Delta_{+\infty}(\tau_J^R)]X \to \overline{X}, \mathbf{r}_w$ and $\epsilon_{w,k}$ may be built using logarithmic space on the input w, k. Finally, we can conclude this section by explicitly providing the desired result.

Theorem 1 *Problem Check-APE* -FD *is NP-Complete.*

Algorithms for Checking *APE*-FDs

As we proved in "The Computational Complexity of Checking *APE*-FD" section, given an *APE*-FD $[\Delta_k(\tau_J^R)]X\overline{Y} \xrightarrow{\epsilon} \overline{Z}$ and an instance $\mathbf{r}$ of R, the problem Check-*APE*-FD is NP-Complete in $|\mathbf{r}|$. Then, in principle, there is no asymptotically better algorithm than exploring the whole set of possible subsets $\mathbf{r}'$ of $\mathbf{r}$ with $\frac{|\mathbf{r}|-|\mathbf{r}'|}{|\mathbf{r}|} \leq \epsilon$.

In the following, we provide two algorithms that make use of heuristics, for pruning the search space in order to achieve the tractability for many cases.

The first algorithm is the more general one, and it may be applied without assumptions on the input instance $\mathbf{r}$. Such algorithm makes use of two optimization techniques. The first one consists of trying, whenever it is possible, to split the current subset of $\mathbf{r}$ into two subsets, on which the problem may be solved independently (i.e., choices in one subset do not affect those in the other one and vice versa). The latter optimization technique consists of checking whether the current partial solution may not lead to an optimal solution (i.e., a solution $\mathbf{r}'$ where $|\mathbf{r}'|$ is the maximum possible number of tuples that may be kept). If this happens, the subtree is pruned immediately (i.e., we are looking only for optimal solutions).

The second algorithm is applicable under the assumption that we have a bounded and relatively small number of tuples that share the same values for both *VT* and *J* which is often the case in clinical domains, as we will discuss later on. In this setting, we show how to provide an upper bound value for all the candidate solutions that contain the current partial solution and thus we can apply a pure branch-and-bound approach in order to speed up the algorithm even more.

Before discussing in detail the algorithms and their properties, we need to introduce some basic concepts and features for the representation of tuples through graph-based structures.

Graph-Based Structures for Tuple Representation

To this regard, we use a suitable graph representation of tuples. A directed graph is a pair $G = (V, E)$, where V is a finite set of nodes and $E \subseteq \binom{V}{2}$ is its edge set. Our graphs are simple, i.e., there are no loops and no parallel edges. Let $U \subseteq V$: we denote by $G|_U = (U, E|_U)$ the subgraph of G induced by U, that is, the graph on node set U such that, for every $u, v \in U$, (u, v) is an edge in $G|_U$ if and only if (u, v) is an edge in G. A *Layered Directed Acyclic Graph* (*L-DAG* for short) is a triple $\mathcal{L}_G = (V, E, l)$ where (V, E) is a DAG and l is a function $l : V \to \{1, \dots, p\}$ for some $p \in \mathbb{N}$ where for every $(u, v) \in E$ we have $l(u) < l(v)$. We define the *jump value* of an edge $(u, v) \in E$ as $jump(u, v) = l(v) - l(u)$. For each $i \in \{1, \dots, p\}$, we denote

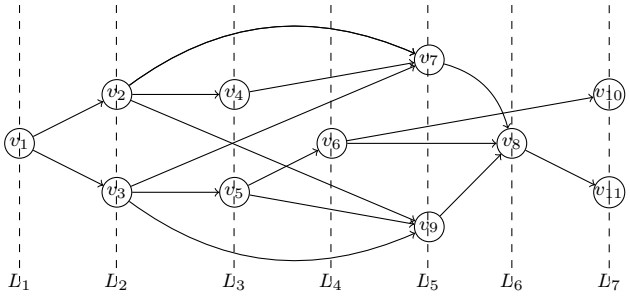


Fig. 11 An example of *L-DAG*

with L_i the set $L_i = \{v \in V : l(v) = i\}$. Obviously, the notion of induced subgraph naturally extends to *L-DAG*. Given an *L-DAG* $\mathcal{L}_G = (V, E, l)$ and a subset U, we define the *L-DAG* induced by U as $\mathcal{L}_G|_U = (U, E|_U, l|_U)$ where $(U, E|_U)$ is the U-induced subgraph of the graph (V, E) and $l|_U : U \to \mathbb{N}$ with $l|_U(v) = l(v)$ for every $v \in U$. An example of *L-DAG* $\mathcal{L}_G = (V, E, l)$ is given in Fig. 11. Edges $(v_2, v_7), (v_2, v_9), (v_3, v_7)$ and (v_3, v_9) have jump value equal to 3. For instance, for $\mathcal{L}_G$, we may change the layering function l into l' where $L_i = L'_i$ for every $i = 1, \ldots 4, L'_5 = L_5 \backslash \{v_7\}$ and $L'_6 = L_6 \cup \{v_7\}$.

We extend the notion of *L-DAG* with weights, denoting it as *wL-DAG*. A *wL-DAG* is expressed through the tuple $\mathcal{WL}_G = (V, E, l, \mathcal{W})$, where $\mathcal{L}_G = (V, E, l)$ is an *L-DAG* and $\mathcal{W}$ is a function $\mathcal{W} : V \to \mathbb{N}^+$. Let us notice that in our notion of weighted *L-DAG*, weights are associated with nodes. Let us now introduce a general problem on *L-DAG*, called *k-Thick Path* (*k-TP* for short).

Problem 4 (*k-TP*). Given an *L-DAG* $\mathcal{L}_G = (V, E, l)$ and a natural number k, determine whether or not there exists a node subset $V' \subseteq V$, such that $|V'| \geq k$ and for every $u, v \in V'$, with $l(u) < l(v)$, there exists a directed path from u to v in $\mathcal{L}_G|_{V'}$.

For instance, if we consider the *L-DAG* in Fig. 11, we have that the set $V' = \{v_1, v_2, v_3, v_7, v_9, v_8, v_{11}\}$ is a possible solution for k-TP with $k \leq 7$ while $V'' = \{v_1, v_2, v_3, v_4, v_7, v_8, v_{11}\}$ is not a candidate solution since there is no path from v_3 to v_4 and $l(v_3) < l(v_4)$. In a solution, we may choose to take more than one node per layer as well as completely ignore all the nodes in a layer. Then, we may see a candidate solution V' as the result of a two-step nondeterministic guess:

1. First we select a set of $p' \leq p$ layers $\{l_1, \ldots l_{p'}\} \subseteq \{1, \ldots, p\}$ (let us assume $l_i < l_j$ for every $1 \leq i < j \leq p'$), which will be all and only the layers which contain at least one node in our solution;
2. For $1 \leq i \leq p'$ we select a nonempty set $V_i \subseteq V$ such that $l(v) = l_i$ for every $v \in V_i$ and for every $(v', v) \in V_{i-1} \times V_i$, we have $(v', v) \in E$.

Going back to the example in Fig. 11, in V'' condition 2 is violated because by choosing v_4 we choose layer 3 as the nonempty layer following layer 2 but $(v_3, v_4) \notin E$. As a matter of fact, $V'' \backslash \{v_3\}$ (i.e., we choose only v_2 in the layer 2) turns out to be candidate solution. In V', we ignore layers 3 and 4 by not choosing any node in them. Instead, we choose layer 5 as the nonempty layer following layer 2 and everything works just fine.

The *k-TP* problem may naturally be extended to *wL-DAG* by imposing the set V' to satisfy $\Sigma_{v \in V'} \mathcal{W}(v) \geq k$. In [10], we prove that the *k-TP* problem on *wL-DAG* is NP-hard. Our proof can be naturally extended to prove that the nonweighted version of the problem is NP-hard too.

The First Algorithm

Both algorithms rely on the concept of *color* that we will explain through an example in the following. Given an *APE-FD* $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ and an instance $\mathbf{r}$ of R, let us suppose that we are solving the problem Check-*APE*-FD on such instance with a simple guess-and-check procedure, which makes use of two, initially empty, subsets $\mathbf{r}^+$ (the tuples to be kept in the solution) and $\mathbf{r}^-$ (the tuples to be deleted in the solution) of $\mathbf{r}$. At each step, the procedure guesses a tuple t in $\mathbf{r} \backslash (\mathbf{r}^+ \cup \mathbf{r}^-)$ and decides nondeterministically (*guessing phase*) either to update $\mathbf{r}^+$ to $\mathbf{r}^+ \cup \{t\}$ (i.e., t is kept in the current partial solution) or to update $\mathbf{r}^-$ to $\mathbf{r}^- \cup \{t\}$ (i.e., t is deleted in the current partial solution). When $\mathbf{r} = \mathbf{r}^+ \cup \mathbf{r}^-$ (*checking phase*), the procedure returns *YES* if $\mathbf{r}^+ \vDash [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ and $|\mathbf{r}^-| \leq \epsilon \cdot |\mathbf{r}|$, otherwise it returns *NO*. Hereinafter, we call *partial solution* a triple $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$, such that $(\mathbf{r}^+ \cup \mathbf{r}^-) \subseteq \mathbf{r}$ and $\mathbf{r}^+ \cap \mathbf{r}^- = \emptyset$. If $\mathbf{r} = \mathbf{r}^+ \cup \mathbf{r}^-$ we simply say that $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ is a *solution*. A solution $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ is *consistent* if and only if $\mathbf{r}^+ \vDash [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$. Given two partial solutions $(\mathbf{r}, \mathbf{r}_1^+, \mathbf{r}_1^-)$ and $(\mathbf{r}, \mathbf{r}_2^+, \mathbf{r}_2^-)$, we say that $(\mathbf{r}, \mathbf{r}_2^+, \mathbf{r}_2^-)$ *extends* $(\mathbf{r}, \mathbf{r}_1^+, \mathbf{r}_1^-)$ if and only if $\mathbf{r}_1^+ \subseteq \mathbf{r}_2^+$ and $\mathbf{r}_1^- \subseteq \mathbf{r}_2^-$.

Is there a way to check whether we are generating an inconsistent solution, possibly without guessing all tuples in $\mathbf{r}$? Violations of the latter constraint (i.e., $|\mathbf{r}^-| \leq \epsilon \cdot |\mathbf{r}|$) are fairly simple to detect during the guessing phase. Indeed, it suffices to check after each insertion in $\mathbf{r}^-$ if $|\mathbf{r}^-|$ exceeds $\epsilon \cdot |\mathbf{r}|$. If it is the case, the procedure may return *NO* immediately without guessing any further. Violations of the first constraint (i.e., $\mathbf{r}^+ \vDash [\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}$) during the guessing phase are trickier to detect.

When two tuples t, t' share the same value for attribute J (i.e., $t[J] = t'[J]$), we say that they are in the same J-group and $t[J]$ is the value of the J-group containing t and t'. For the sake of brevity, for a given $j \in Dom(J)$ we will use

j-group for denoting the J-group with value j. An *ordered pair*, written *o-pair*, is a pair $(t, t') \in \mathbf{r} \times \mathbf{r}$ such that t and t' are in the same J-group and $t[VT] < t'[VT]$. We say that a pair (t, t') is an *edge* if and only if $0 < t'[VT] - t[VT] \leq k$. Given a triple $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$, an o-pair (t, t') is *active* if and only if $t, t' \in \mathbf{r}^+$ and for every tuple $\bar{t}$ in the same J-group of t if $t[VT] < \bar{t}[VT] < t'[VT]$, we have $\bar{t} \in \mathbf{r}^-$ (i.e., t and t' are selected in the current partial solution and all the tuples between t and t' in the same J-group have been deleted). Given two valid times $vt, vt' \in Dom(VT)$ and a value $j \in Dom(J)$, vt and vt' are consecutive in the j-group if and only if there exists an active o-pair (t, t') with $t[VT] = vt, t'[VT] = vt'$ and $t[J] = j$. It is important to observe that we may have two distinct values $j, j' \in Dom(J)$ and two distinct valid times $vt, vt' \in Dom(VT)$ which are consecutive in j-group and not consecutive in j'-group. Moreover, we may have edges (t, t') that are not active and active pairs (t, t') that are not edges; such is the case of active pairs (t, t') with $t'[VT] - t[VT] > k$. A *color* is a tuple c on the schema $C = XYZ$, and we say that two colors (x, y, z) and (x', y', z') are *conflicting* if and only if $x = x'$, $y = y'$, and $z \neq z'$. Given an o-pair (t, t'), its color, denoted by $c(t, t')$, is the tuple $c(t, t') = (t[X], t'[Y], t'[Z])$. Two o-pairs (t, t'), (t'', t''') are *conflicting* if and only if $c(t, t')$ and $c(t'', t''')$ are conflicting.

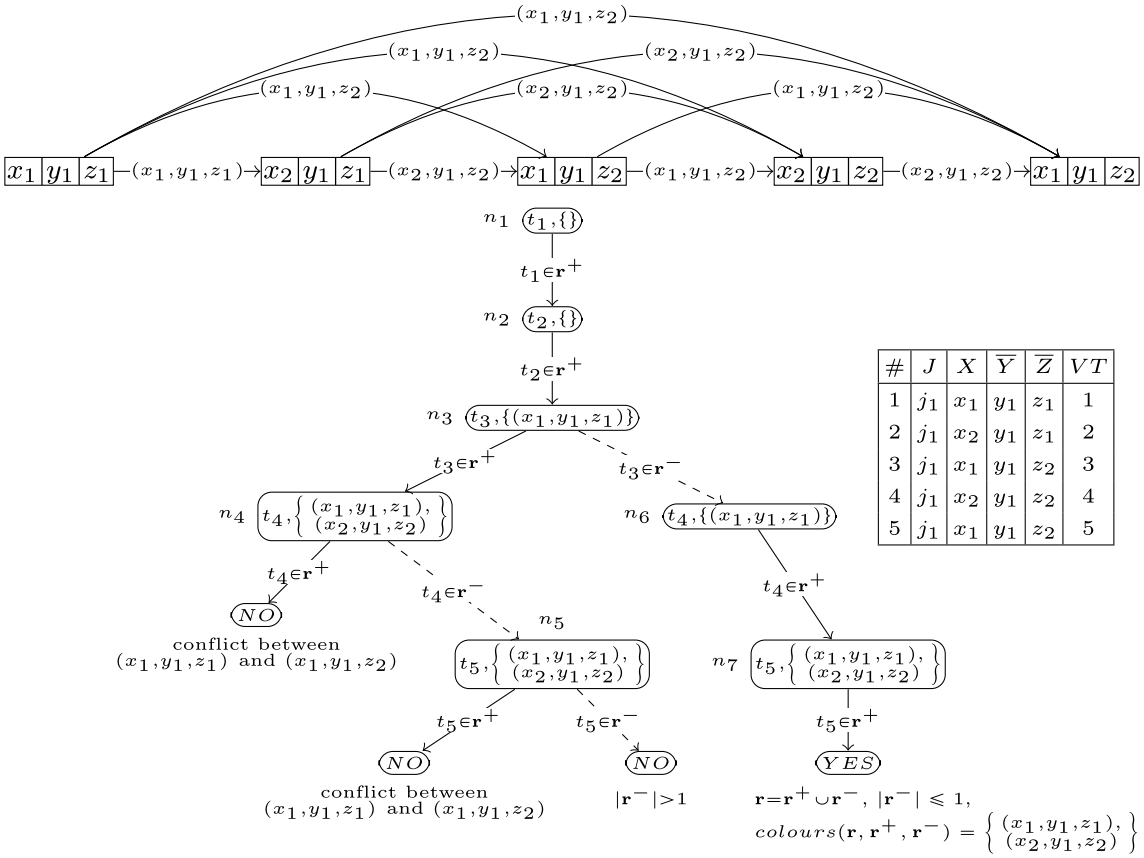
Theorem 2 *Given an APE-FD* $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, *an instance* $\mathbf{r}$ *of* R, *and a partial solution* $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$, *if there exist two active and conflicting edges* (t, t') *and* (t'', t'''), *then for every solution* $(\mathbf{r}, \mathbf{r}_f^+, \mathbf{r}_f^-)$ *that extends* $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ *it holds* $\mathbf{r}_f^+ \not\models [\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}$ *(i.e., the solution is inconsistent).*

The above theorem guarantees that from a partial solution $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ that features at least two conflicting edges, we cannot reach a consistent solution $(\mathbf{r}, \mathbf{r}_f^+, \mathbf{r}_f^-)$. In such a case, we may return immediately *NO* without considering any further $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$. The colors of a partial solution $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ are represented by the set $colors(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-) = \{(t[X], t'[Y], t'[Z]) : (t, t')$ is an active edge in $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)\}$. Clearly, the hypothesis of Theorem 2 applies if and only if set $colors(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ contains at least two conflicting colors.

Then, by means of colors, our above guess-and-check procedure may be improved by adding the control on the size of $\mathbf{r}^-$ and by keeping updated the current set of colors $colors(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$. Once an insertion of a tuple in either $\mathbf{r}^+$ or $\mathbf{r}^-$ introduces a color c that is conflicting with at least one color in $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$, the procedure answers *NO* immediately. An example of how the procedure works is given in Fig. 12, where we have an instance of five tuples with $\epsilon = 0.2$ (i.e., we may delete at most one tuple) and $k = 6$ (all the tuples are in the same window). The execution depicted in Fig. 12 guesses the values of tuples from the oldest (t_1) to the newest one (t_2) according to the value of *VT*. First, it tries to put the current tuple t in $\mathbf{r}^+$; if no violation arises, it continues; if some violation arises, it tries to insert tuple t in $\mathbf{r}^-$; if no violation arises, it continues; otherwise, it goes back to the previous choice (i.e., backtracking). Every internal node is labeled with the current tuple, which will be guessed next; every leaf is labeled either with *YES* (i.e., the current branch is a solution) or *NO* (i.e., a violation has arisen); the current set of colors is reported within the node. Nodes are numbered according to their order of appearance. We have that the root is n_1 followed by the introduction of nodes $n_2 \ldots n_4$ in this precise order. If we introduce t_4 in the partial solution associated to n_4, we violate the first constraint. Since in n_4 adding t_4 in $\mathbf{r}^-$ does not generate any violation, node n_5 is created as child of n_4. However, node n_5 cannot be extended without introducing a violation in the above constraints. Indeed, if we put t_5 in $\mathbf{r}^+$, we introduce a conflicting color; if we put t_5 in $\mathbf{r}^-$, we exceed the maximum number of allowed deletions. We backtrack to n_4. As all the possible choices have been explored, we backtrack to n_3, where the choice of adding t_3 to $\mathbf{r}^-$ is attempted, generating node n_6. From n_6, we put t_5 in $\mathbf{r}^+$ without violating any constraint and thus we have that $\{t_1, t_2, t_4, t_5\} \models [\Delta_6(\tau_J^R)]X\overline{Y} \overset{0.2}{\longrightarrow} \overline{Z}$.

Let us now consider in some more detail the first algorithm. Basically, the algorithm works similarly to the previous procedure, except for some trivial technicalities. Two more heuristics have been introduced, to possibly stop earlier, during the exploration of a branch in the tree of computation. The main procedure of the algorithm is reported in Fig. 15, while auxiliary procedures are reported in Figs. 13 and 14. The algorithm is implemented by function *TupleWiseMin* that takes four arguments. The first argument is $G_{\mathbf{r}}$, which is derived from $\mathbf{r}$ considering the *APE*-FD $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ that has to be checked. More precisely, $G_{\mathbf{r}}$ is an instance of schema J, X, Y, Z, VT, *count*, with $Dom(count) = \mathbb{N}$. We have that $t \in G_{\mathbf{r}}$ if and only if there exists $t' \in \mathbf{r}$ for which $(t'[J], t'[X], t'[Y], t'[Z]) = (t[J], t[X], t[Y], t[Z])$ and $t[count] = |\{t' \in \mathbf{r} : (t'[J], t'[X], t'[Y], t'[Z]) = (t[J], t[X], t[Y], t[Z])\}|$ that is, we count how many tuples in $\mathbf{r}$ share the same values for attributes J, X, Y and Z, respectively. The input parameter k is the length for the grouping sliding window. Sets $G_{\mathbf{r}}^+$ and $G_{\mathbf{r}}^-$, originally initialized to $\emptyset$, represent the tuples of $G_{\mathbf{r}}$ that are either kept or deleted in the current solution, respectively. On instances s of schema J, X, Y, Z, VT, *count*, we denote with $\|r\|$ the sum on the *count* attribute for the tuples in s (i.e., $||s|| = \sum_{t \in s} t[count]$). Finally, $\mathcal{C}$ is a set of *colors* which is initially set to $\emptyset$. A color c is a tuple on the schema X, Y, Z. As we will see, $\mathcal{C}$ keeps track via colors of the constraints introduced so far in the construction of the solution.

Procedure *TupleWiseMin* returns the minimum number of tuples that has to be deleted from $\mathbf{r}$ in order to obtain an instance $\mathbf{r}'$ such that $\mathbf{r}' \models [\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}$. Then, if such minimum is less or equal than $\epsilon \cdot |\mathbf{r}|$ we can conclude $\mathbf{r} \models [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, else we have $\mathbf{r} \setminus \not\models [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$.



#	J	X	$\overline{Y}$	$\overline{Z}$	VT
1	j_1	x_1	y_1	z_1	1
2	j_1	x_2	y_1	z_1	2
3	j_1	x_1	y_1	z_2	3
4	j_1	x_2	y_1	z_2	4
5	j_1	x_1	y_1	z_2	5

Fig. 12 An example of how the use of colors improves a guess and check procedure for solving the problem Check-*APE*-FD

Given $G_{\mathbf{r}}$, $G^+_{\mathbf{r}}$, $G^-_{\mathbf{r}}$ and a set of colors $\mathcal{C}$ we say that an edge $(t,t') \in G_{\mathbf{r}} \times G_{\mathbf{r}}$ is *pending* if and only if the following conditions hold:

1. $t,t' \notin G^-_{\mathbf{r}}$ and $(t[X],t'[Y],z) \notin \mathcal{C}$ for every $z \in Dom(Z)$;
2. for every t'' with $t''[J] = t[J]$ and $t[VT] < t''[VT] < t'[VT]$ we have $t'' \notin G^+_{\mathbf{r}}$;
3. there exists $t'' \in (G_{\mathbf{r}} \cup \{t,t'\}) \backslash (G^-_{\mathbf{r}} \cup G^+_{\mathbf{r}})$ with $t''[J] = t[J]$ and $t[VT] \leq t''[VT] \leq t'[VT]$.

Informally speaking, a pending edge is an edge that is not active in the current partial solution but it may become active during the computation and, if it happens, it introduces a new color in $\mathcal{C}$. In our algorithm, pending edges for the current partial solution are retrieved by procedure $E?$, while active edges are retrieved by procedure $E!$.

Procedure *TupleWiseMin* (Fig. 15) works as follows. If $G^+_{\mathbf{r}} \cup G^-_{\mathbf{r}} = G_{\mathbf{r}}$, it means that we have obtained a solution without violating any constraint and thus we can return $||G^-_{\mathbf{r}}||$ (i.e., the number of deleted tuples). If $G^+_{\mathbf{r}} \cup G^-_{\mathbf{r}} \neq G_{\mathbf{r}}$, the algorithm guesses a tuple $t \in G_{\mathbf{r}} \backslash (G^+_{\mathbf{r}} \cup G^-_{\mathbf{r}})$ and proceeds as follows. First, it checks whether inserting t into $G^+_{\mathbf{r}}$ does not cause any violation of constraints. If so, it stores in m_t the value of the recursive call to *TupleWiseMin* where t belongs to $G^+_{\mathbf{r}}$ and $\mathcal{C}$ has been updated accordingly. By inserting a tuple t in $G^+_{\mathbf{r}}$, the algorithm is asserting that t belongs to the current partial solution, while by inserting t in $G^-_{\mathbf{r}}$ the algorithm is asserting that t does not belong to the current partial solution. If a constraint is violated, the algorithm stores in m_t the value $+\infty$, which means that t may not be kept in the current solution.

Then, it checks whether inserting t into $G^-_{\mathbf{r}}$ does not cause any violation in the constraints. If it is the case, it stores in $m_{\backslash t}$ the value of the recursive call to *TupleWiseMin*, where $G^-_{\mathbf{r}}$ and $\mathcal{C}$ are updated accordingly. If a constraint is violated, the algorithm stores in $m_{\backslash t}$ the value $+\infty$, which means that t must be kept in the current partial solution. In procedure *TupleWiseMin*, the only way in which a constraint may be violated is that, after the insertion a tuple t in $G^+_{\mathbf{r}}$ (resp. $G^-_{\mathbf{r}}$), an edge (t',t'') turns out to be active and its color $(t'[X],t''[Y],t''[Z])$ turns out to be conflicting with at least one color in $\mathcal{C}$.

As pointed out by the example in Fig. 12, checking each step for consistency is itself an optimization, even if it is trivial, since it allows us to prune entire subtrees in the tree of computations without exploring them. We propose

Fig. 13 Auxiliary procedures used by procedures presented in Figs. 14, 15 and 17

procedure InBetween($G_\mathbf{r}, t_1, t_2$)
comment: returns the subset of $G_\mathbf{r}$ consisting of all the tuples in the same J-group of t_1 and t_2 that feature valid times between $t_1[VT]$ and $t_2[VT]$.
return $\{t \in G_\mathbf{r} : t_1[VT] < t[VT] < t_2[VT] \wedge t[J] = t_1[J]\}$

procedure EdgeConflict?($(t_1, t_2), (t'_1, t'_2)$)
comment: returns true if and only if the two input edges feature conflicting colors.
if $t_1[X] = t'_1[X] \wedge t_2[\overline{Y}] = t'_2[\overline{Y}] \wedge t_2[\overline{Z}] \neq t'_2[\overline{Z}]$
then return true
else return false

procedure ColorConflict?($(t_1, t_2), \mathcal{C}$)
comment: returns true if and only if the color of the edge (t_1, t_2) is conflicting with at least one color in the set of colors $\mathcal{C}$.
if $\exists c(c \in \mathcal{C} \wedge t_1[X] = c[X] \wedge t_2[\overline{Y}] = c[\overline{Y}] \wedge t_2[\overline{Z}] \neq c[\overline{Z}])$
then return true
else return false

procedure E!($G_\mathbf{r}, k, G^+_\mathbf{r}, G^-_\mathbf{r}$)
comment: returns the set of all and only active edges in the current partial solution.
return $\left\{(t_1, t_2) : \begin{array}{l} t_1[J] = t_2[J] \wedge 0 < t_2[VT] - t_1[VT] \leqslant k \wedge \\ \{t_1, t_2\} \subseteq G^+_\mathbf{r} \wedge InBetween(G_\mathbf{r}, t_1, t_2) \subseteq G^-_\mathbf{r}) \end{array}\right\}$

procedure E?($G_\mathbf{r}, k, G^+_\mathbf{r}, G^-_\mathbf{r}, \mathcal{C}$)
comment: returns the set of all and only pending edges in the current partial solution.
return $\left\{(t_1, t_2) : \begin{array}{l} t_1[J] = t_2[J] \wedge 0 < t_2[VT] - t_1[VT] \leqslant k \wedge \{t_1, t_2\} \cap G^-_\mathbf{r} = \varnothing \wedge \\ \neg ColorConflict?((t_1, t_2), \mathcal{C}) \wedge InBetween(G_\mathbf{r}, t_1, t_2) \cap G^+_\mathbf{r} = \varnothing \wedge \\ (\{t_1, t_2\} \cup InBetween(G_\mathbf{r}, t_1, t_2)) \nsubseteq (G^+_\mathbf{r} \cup G^-_\mathbf{r}) \end{array}\right\}$

here two further optimizations for this procedure. The first one allows us to restrict the search space by splitting the problem into independent subproblems in a divide-and-conquer fashion. Let us suppose that at a certain step of our computation, there exists a value $j \in \{t[J] : t \in G_\mathbf{r}\}$, for which for each pair of conflicting pending edges (t, t') and $(\bar{t}, \bar{t}')$ we have that either all $t, t', \bar{t}$, and $\bar{t}'$ belong to the j-group or all $t, t', \bar{t}$ and $\bar{t}'$ do not belong to the j-group (such condition is verified by subprocedure *Group Independent?* reported in Fig. 14.) Let $\overline{G}_\mathbf{r} = \{t : t[J] = j\}$. As every edge involving tuples in the j-group is not conflicting with every edge that may be introduced outside the j-group, then we can split the problem into the two subproblems $(\overline{G}_\mathbf{r}, k, G^+_\mathbf{r} \cap \overline{G}_\mathbf{r}, G^-_\mathbf{r} \cap \overline{G}_\mathbf{r})$ and $(G_\mathbf{r} \backslash \overline{G}_\mathbf{r}, k, G^+_\mathbf{r} \backslash \overline{G}_\mathbf{r}, G^-_\mathbf{r} \backslash \overline{G}_\mathbf{r})$. Such problems are independent and may be solved separately. The resulting value for the solution is the sum of the values returned by *TupleWiseMin* applied to both the two subproblems. Let $H = |G_\mathbf{r} \backslash (G^+_\mathbf{r} \cup G^-_\mathbf{r})|$ and $h = |\{t \in (G_\mathbf{r} \backslash (G^+_\mathbf{r} \cup G^-_\mathbf{r})) : t[J] = j\}|$. In this case, the upper bound of the complexity at the current step of computation drops from $\mathcal{O}(2^H)$ to $\mathcal{O}(2^{H-h} + 2^h)$.

The second optimization allows us to prune a subtree of computation even before a contradiction arises. It verifies, in many cases, whether every possible solution that may be built starting from the current partial one turns to be not minimal. Suppose that there exists an active o-pair (t, t') in a partial solution $(G_\mathbf{r}, G^+_\mathbf{r}, G^-_\mathbf{r})$, such that there exists $\bar{t} \in G_\mathbf{r}$ in the same J-group of t with $t[VT] < \bar{t}[VT] < t'[VT]$. By definition of active o-pair, we have that $\bar{t}$ belongs to $G^-_\mathbf{r}$ as well as every tuple $\bar{t}'$ in the same J-group of t with $t[VT] < \bar{t}'[VT] < t'[VT]$. Here, the additional condition is that there exists at least one of such tuples. Given a partial solution $(G_\mathbf{r}, G^+_\mathbf{r}, G^-_\mathbf{r})$, we define set $colors(G_\mathbf{r}, G^+_\mathbf{r}, G^-_\mathbf{r}) = \{(x, y, z) :$ there exists an active edge (t, t') with $c(t, t') = (x, y, z)\}$. Let us define the set of colors $pending(G_\mathbf{r}, G^+_\mathbf{r}, G^-_\mathbf{r}) = \{(x, y, z) :$ there exists a pending edge (t, t') with $c(t, t') = (x, y, z)\}$, which collects all and only the colors that may be introduced later on in the current computation.

A color (x, y, z) is *safe* in (vt, vt', j) if and only if one of the following three conditions hold:

1. $(x, y, z) \in colors(G_\mathbf{r}, G^+_\mathbf{r}, G^-_\mathbf{r})$;
2. Every color $(x, y, z') \in pending(G_\mathbf{r}, G^+_\mathbf{r}, G^-_\mathbf{r})$ satisfies $z' = z$ (i.e., (x, y, z) is a pending color and there is no pending color that is conflicting with (x, y, z));
3. The color is not conflicting with any color in $colors(G_\mathbf{r}, G^+_\mathbf{r}, G^-_\mathbf{r}) \cup pending(G_\mathbf{r}, G^+_\mathbf{r}, G^-_\mathbf{r})$ and do not exist two tuples $\bar{t}, \bar{t}' \in (G^+_\mathbf{r} \cup G^-_\mathbf{r}) \cap \{\bar{t}'' \in G_\mathbf{r} : \bar{t}''[J] = j \wedge vt \leq \bar{t}'' \leq vt'\}$, such that $(\bar{t}, \bar{t}')$ is an edge and the color $(\bar{t}[X], \bar{t}'[Y], \bar{t}'[Z])$ is conflicting with (x, y, z).

The three conditions above imply that if a color is safe in (vt, vt', j), then it is neither in conflict with a color in

Fig. 14 Auxiliary procedures used by procedure *TupleWiseMin* (Fig. 15)

procedure GROUPINDEPENDENT?$(j, G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C})$

comment: returns true if and only if in the current partial solution pending edges involving tuples belonging to the J-group with value j are not conflicting with the pending edges introduced by other J-groups.

$E_j \leftarrow \{(t_1, t_2) \in E?(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C}) : t_1[J] = j\}$
$\overline{E}_j \leftarrow E?(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C}) \backslash E_j$
if $\exists t_1 \exists t_2 \exists \bar{t}_1 \exists \bar{t}_2((t_1, t_2) \in E_j \wedge (\bar{t}, \bar{t}_2) \in \overline{E}_j \wedge EdgeConflict?((t_1, t_2), (\bar{t}_1, \bar{t}_2)))$
then return false
else return true

procedure MAXCONSISTENTSUBSET(E)

comment: returns the maximal subset E' of E such that for every edge $(t'_1, t'_2) \in E'$ and for every edge $(t_1, t_2) \in E$ we have that $c(t'_1, t'_2)$ and $c(t_1, t_2)$ are not conflicting.

$E' \leftarrow \varnothing$
for each $(t_1, t_2) \in E$ **do** { **if** $\forall \bar{t}_1 \forall \bar{t}_2((\bar{t}_1, \bar{t}_2) \in E \rightarrow \neg EdgeConflict?((t_1, t_2), (\bar{t}_1, \bar{t}_2)))$ **then** $E' \leftarrow E' \cup \{(t_1, t_2)\}$
return E'

procedure REACH?$(t_1, t_2, Nodes, Edges)$

comment: returns true if and only if there exists a path from t_1 to t_2 in the graph $(Nodes, Edges)$. It is a function that checks whether there exists a path between two nodes in a graph or not .

if $\exists(\{\bar{t}_1 \ldots \bar{t}_m\} \subseteq Nodes)(\bar{t}_1 = t_1 \wedge \bar{t}_m = t_2 \wedge \forall(1 \leqslant i < m)((\bar{t}_i, \bar{t}_{i+1}) \in Edges))$
then return true
else return false

procedure REPLACEPATH?$(t, \bar{t}, G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C})$

comment: returns true if and only if in the current partial solution the consecutive valid times $t[VT]$ and $\bar{t}[VT]$ in $t[J]$-group may be safely replaced.

if $\neg\exists t_1(t_1 \in G_{\mathbf{r}} \wedge t_1[J] = t[J] \wedge t[VT] < t_1[VT] < \bar{t}[VT])$
then return false
$N_s \leftarrow \{t_1 \in G^+_{\mathbf{r}} : t_1[J] = t[J] \wedge t_1[VT] = t[VT]\}$
$N_e \leftarrow \{t_1 \in G^+_{\mathbf{r}} : t_1[J] = \bar{t}[J] \wedge t_1[VT] = \bar{t}[VT]\}$
$N_m \leftarrow \{t_1 \in G^-_{\mathbf{r}} : t_1[J] = t[J] \wedge t[VT] < t_1[VT] < \bar{t}[VT]\}$
$N \leftarrow N_s \cup N_e \cup N_m$
$\tilde{E} \leftarrow \{(t_1, t_2) : t_1 \in N \wedge t_2 \in N \wedge t_1[VT] < t_2[VT] \wedge (t_1 \notin N_s \vee t_2 \notin N_e)\}$
$Pairs \leftarrow \{(t_1, t_2) : t_1 \in N \wedge t_2 \in N \wedge t_1[VT] + k < t_2[VT] \wedge (t_1 \notin N_s \vee t_2 \notin N_e)\}$
$\tilde{E}_{ok} \leftarrow (MaxConsistentSubset(E?(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C})) \cap \tilde{E}) \cup Pairs$
$NotSafe \leftarrow \varnothing$
for each $(t_1, t_2) \in \tilde{E} \backslash \tilde{E}_{ok}$
do { **for each** $(\bar{t}_1, \bar{t}_2) \in (E?(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C}) \cup E!(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}) \cup \tilde{E})$ **do** { **if** $EdgeConflict?((t_1, t_2), (\bar{t}_1, \bar{t}_2))$ **then** $NotSafe \leftarrow NotSafe \cup \{(t_1, t_2)\}$
$\tilde{E} \leftarrow \tilde{E} \backslash NotSafe$
if $\exists t_1 \exists t_2 \forall \bar{t}_1 \forall \bar{t}_2(t_1, t_2 \in N_m \wedge \bar{t}_1 \in N_s \wedge \bar{t}_2 \in N_e \wedge (\bar{t}_1, t_1), (t_2, \bar{t}_2) \in \tilde{E} \wedge Reach?(t_1, t_2, N, \tilde{E}))$
then return true
return false

$colors(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$ nor with a color in $pending(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$. However, this is just a necessary but not sufficient condition. Let us consider the example shown in Fig. 16 and assume that $k \geq 7$ (i.e., every o-pair in the example is also an edge). We have that the active edges are $(t_1, t_2), (t_2, t_3), (t_3, t_4)$, and (t_4, t_5) for the j_1-group and $(t_7, t_{12}), (t_8, t_{12})$ for the j_2-group, since we have $t_9, t_{10}, t_{11} \in G^-_{\mathbf{r}}$. Thus, we have $colors(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}) = \{(x_1, y_4, z_4), (x_2, y_5, z_6), (x_5, y_6, z_6), (x_4, y_6, z_5), (x_1, y_6, z_6), (x_2, y_6, z_6)\}$ and, since we have to decide the status of tuple t_6, we have $pending(G_{\mathbf{r}}, G^+_{\mathbf{r}},$

Algorithm 6.1: TUPLEWISEMIN$(G_{\mathbf{r}}, k, G^+_{\mathbf{r}} = \emptyset, G^-_{\mathbf{r}} = \emptyset, \mathcal{C} = \emptyset)$

procedure MAXIMALPATHS?$(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C})$

comment: returns true if and only if in the current partial solution for every J-group j-group every pair of consecutive valid times vt and vt' in j-group cannot be safely replaced.

if $\exists t_1 \exists t_2((t_1, t_2) \in E!(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}) \wedge ReplacePath?(t_1, t_2, k, G_{\mathbf{r}}, G^+_{\mathbf{r}} G^-_{\mathbf{r}}, \mathcal{C}))$
then return false
else return true

procedure ISCONSISTENT?$(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C})$

comment: returns true if and only if the current partial solution features consistent colors in $\mathcal{C}$ and for every J-group j-group every pair of consecutive valid times vt and vt' in j-group cannot be safely replaced.

if $\exists t_1 \exists t_2((t_1, t_2) \in E!(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}) \wedge ColorConflict?((t_1, t_2), \mathcal{C}))$
then return false
if $MaximalPaths?(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C})$
then return true
else return false

main

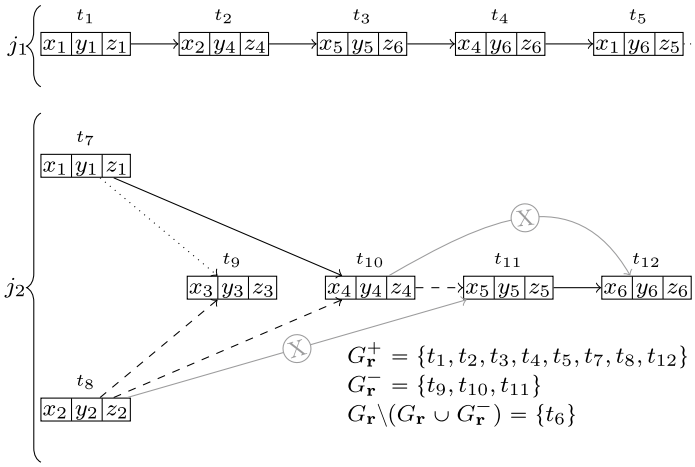
comment: returns the minimum value $m = \min ||G_{\mathbf{r}} \backslash G'_{\mathbf{r}}||$ for $G'_{\mathbf{r}}$ in $\{G''_{\mathbf{r}} \subseteq G_{\mathbf{r}} : G'_{\mathbf{r}} \models [\tau^{XYZcount}_{G_{\mathbf{r}}}]X\overline{Y} \rightarrow \overline{Z}\}$.

if $G_{\mathbf{r}} = \emptyset$ **then return** $||G^-_{\mathbf{r}}||$
if $\exists j(j \in Dom(J) \wedge GroupIndependent?(j, G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C}))$
then $\begin{cases} \overline{G}_{\mathbf{r}} \leftarrow \{t \in G_{\mathbf{r}} : t[J] = j\} \\ \textbf{return} \begin{pmatrix} TupleWiseCheck(\overline{G}_{\mathbf{r}}, k, G^+_{\mathbf{r}} \cap \overline{G}_{\mathbf{r}}, G^-_{\mathbf{r}} \cap \overline{G}_{\mathbf{r}}, \mathcal{C}) + \\ TupleWiseCheck(G_{\mathbf{r}} \backslash \overline{G}_{\mathbf{r}}, k, G^+_{\mathbf{r}} \backslash \overline{G}_{\mathbf{r}}, G^-_{\mathbf{r}} \backslash \overline{G}_{\mathbf{r}}, \mathcal{C}) \end{pmatrix} \end{cases}$
let $t \in G_{\mathbf{r}}$
if $IsConsistent?(G_{\mathbf{r}} \backslash \{t\}, k, G^+_{\mathbf{r}} \cup \{t\}, G^-_{\mathbf{r}}, \mathcal{C})$
then $\begin{cases} \mathcal{C}' \leftarrow \mathcal{C} \cup \{(t'[X], t''[Y], t''[Z]) : (t', t'') \in E!(G_{\mathbf{r}} \backslash \{t\}, k, G^+_{\mathbf{r}} \cup \{t\}, G^-_{\mathbf{r}})\} \\ m_t \leftarrow TupleWiseCheck(G_{\mathbf{r}} \backslash \{t\}, k, G^+_{\mathbf{r}} \cup \{t\}, G^-_{\mathbf{r}}, \mathcal{C}') \end{cases}$
else $m_t \leftarrow +\infty$
if $IsConsistent?(G_{\mathbf{r}} \backslash \{t\}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}} \cup \{t\}, \mathcal{C})$
then $\begin{cases} \mathcal{C}' \leftarrow \mathcal{C} \cup \{(t'[X], t''[Y], t''[Z]) : (t', t'') \in E!(G_{\mathbf{r}} \backslash \{t\}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}} \cup \{t\})\} \\ m_{\not t} \leftarrow TupleWiseCheck(G_{\mathbf{r}} \backslash \{t\}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}} \cup \{t\}, \mathcal{C}') \end{cases}$
else $m_{\not t} \leftarrow +\infty$
return $\min(m_t, m_{\not t})$

Fig. 15 The main procedure for a tuple-wise check of *APE*-FDs. Notice that we use a compact notation for the recursive procedure which is initially called *TupleWiseMin*$(G_{\mathbf{r}}, k)$. Here, when $G^+_{\mathbf{r}}$, $G^-_{\mathbf{r}}$ and $\mathcal{C}$ are omitted in the procedure call, they get their respective default values specified in the procedure declaration (i.e., $\emptyset$ for each of them in this case)

$G^-_{\mathbf{r}}) = \{(x_1, y_3, z_2)\}$. Suppose that we are interested in the colors which are safe in $(1, 5, j_2)$. For instance, $c(t_7, t_9) = (x_1, y_3, z_3)$ (i.e., the dotted edge in the j_2-group in Fig. 16) is not safe in $(1, 5, j_2)$ because it does not belong neither to $colors(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$ nor to $pending(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$ (conditions 1 and 2) and it is in conflict with the unique color $(x_1, y_3, z_2) \in pending(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$. On the other hand, colors $c(t_8, t_9) = (x_2, y_3, z_3)$, $c(t_8, t_{10}) = (x_2, y_4, z_4)$ and $c(t_{10}, t_{11}) = (x_4, y_5, z_5)$ (i.e., the dashed edges in Fig. 16) are safe in $(1, 5, j_2)$, because they are neither in conflict with colors either in $colors(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$ or in $pending(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$ and there are no other edges in j_2 with valid times between 1 and 5 that exhibit either (x_2, y_3), or (x_4, y_5), or (x_2, y_3) as first two components of their colors (thus condition 3 applies to these colors). Colors $c(t_7, t_{10}) = (x_1, y_4, z_4)$ and $c(t_{11}, t_{12}) = (x_5, y_6, z_6)$ (i.e., the continuous edges in j_2-group in Fig. 16) are safe in $(1, 5, j_2)$, because they belong to $colors(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$ and thus both satisfy condition 1. Finally, colors $c(t_8, t_{11}) = (x_2, y_5, z_5)$, and $c(t_8, t_{10}) = (x_4, y_6, z_6)$ are not safe in $(1, 5, j_2)$ (i.e., the X-labeled edges in Fig. 16), because they are in conflict with two colors in $colors(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$; more precisely, (x_2, y_5, z_5) is in conflict with (x_2, y_5, z_6) and (x_4, y_6, z_6) is in conflict with (x_4, y_6, z_5).

Given a partial solution $(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$ and the triple (vt, vt', j), a (vt, vt', j)-*replace DAG* is a DAG (V, E) where $V = \{t \in G^+_{\mathbf{r}} : t[VT] = vt \wedge t[J] = j\} \cup \{t \in G^-_{\mathbf{r}} : vt < t[VT] < vt' \wedge t[J] = j\} \cup \{t \in G^+_{\mathbf{r}} : t[VT] = vt' \wedge t[J] = j\}$ and



#	J	X	Y	Z	VT	$count$
1	j_1	x_1	y_1	z_1	2	1
2	j_1	x_2	y_4	z_4	4	1
3	j_1	x_5	y_5	z_6	5	1
4	j_1	x_4	y_6	z_6	6	1
5	j_1	x_1	y_6	z_5	7	1
6	j_1	x_2	y_3	z_2	6	1
7	j_2	x_1	y_1	z_1	1	1
8	j_2	x_2	y_2	z_2	2	1
9	j_2	x_3	y_3	z_2	3	1
10	j_2	x_4	y_4	z_3	4	1
11	j_2	x_5	y_5	z_4	5	1
12	j_2	x_6	y_6	z_5	6	1

Fig. 16 An example of how a partial solution may be improved

$$E = \left\{ (t,t') \in V \times V : \begin{array}{c} (t[VT] \neq vt \vee t'[VT] \neq vt') \wedge t[VT] < t'[VT] \wedge \\ c(t,t') \text{ is safe in } (vt, vt', j) \end{array} \right\} \cup \{(t,t') \in V \times V : (t[VT] \neq vt \vee t'[VT] \neq vt') \wedge t'[VT] - t[VT] > k\}$$

A node $t \in V$ is a *starting node* (resp. *ending node*) if and only if $vt < t[VT] < vt'$ and, for every $t' \in V$ with $t'[VT] = vt$ (resp. $t'[VT] = vt'$), we have $(t', t) \in E$ (resp. $(t, t') \in E$). A *replace path* in a (vt, vt', j)-replace DAG (V, E) is any path $t_1 \ldots t_m$ in (V, E) for which t_1 is a starting node and t_m is an ending node. We say that vt and vt' in j can be *safely replaced* if and only if there exists a replace path in the (vt, vt', j) -replace DAG (V, E). Figure 16 depicts the $(1, 5, j_2)$-replace DAG, where t_{10} is the only initial node that is not an ending one, and t_{11} is the only ending node that is not a starting one. Since t_{10} is connected to t_{11}, we have that $t_{10}t_{11}$ is a replace path in the $(1, 5, j_2)$-replace DAG and thus 1 and 5 can be safely replaced in j_2. Using the above definitions of replace DAGs/paths, we can provide the following result.

Theorem 3 *Given a partial solution* $(G_{\mathbf{r}}, G_{\mathbf{r}}^{+}, G_{\mathbf{r}}^{-})$, *if there exists a group j with two consecutive valid times vt and* vt' *such that vt and* vt' *can be safely replaced in j, then every consistent solution that follows* $(G_{\mathbf{r}}, G_{\mathbf{r}}^{+}, G_{\mathbf{r}}^{-})$ *is not optimal.*

The proof of the theorem is straightforward. Let us suppose that $t_1 \ldots t_m$ is a replace path in the (vt, vt', j)-replace DAG. By definition, we have $t_1, \ldots, t_m \in G_{\mathbf{r}}^{-}$. It suffices to take any consistent solution $(G_{\mathbf{r}}, \overline{G_{\mathbf{r}}^{+}}, \overline{G_{\mathbf{r}}})$ that follows $(G_{\mathbf{r}}, G_{\mathbf{r}}^{+}, G_{\mathbf{r}}^{-})$ and such that $(G_{\mathbf{r}}, \overline{G_{\mathbf{r}}^{+}} \cup \{t_1, \ldots, t_m\}, \overline{G_{\mathbf{r}}} \setminus \{t_1, \ldots, t_m\})$ is still a consistent solution. Nonoptimality immediately follows.

We take advantage of Theorem 3 by pruning every computation rooted in a partial solution $(G_{\mathbf{r}}, G_{\mathbf{r}}^{+}, G_{\mathbf{r}}^{-})$ that features a J-group j-group and two consecutive valid times vt and vt' in the j-group, such that vt and vt' can be safely replaced in j. Verifying whether such condition applies or not may be performed in polynomial time. In procedure *TupleWiseMin*, this optimization is realized by subprocedures *Maximal Paths?* and *Replace Path?* reported in Figs. 14 and 15, respectively.

The Second Algorithm

Let us now propose another algorithm with some auxiliary procedures, reported in Figs. 17 and 18, for solving problem Check-*APE*-FD. Such an algorithm, whose main procedure is called *EdgeWiseMin*, strongly differs from *TupleWiseMin* in approaching the problem. In principle, it works better, but it may work only under a quite reasonable assumption on the input, which we will discuss in detail later on.

At every step, procedure *EdgeWiseMin*, instead of guessing if a tuple belongs to the current partial solution, guesses if a color is forbidden or allowed in the current partial solution. Informally, forbidding a color (x, y, z) means avoiding all the active edges $(t, t') \in G_{\mathbf{r}} \times G_{\mathbf{r}}$ for which $c(t, t') = (x, y, z)$. On the other hand, allowing a color (x, y, z) means forbidding all the active edges $(t, t') \in G_{\mathbf{r}} \times G_{\mathbf{r}}$ whose colors are conflicting with (x, y, z). In order to do that, we introduce the concept of *color-partial solution*. A color-partial solution is a triple $(G_{\mathbf{r}}, \mathcal{C}^{+}, \mathcal{C}^{-})$, such that $\mathcal{C}^{+}, \mathcal{C}^{-} \subseteq Dom(X) \times Dom(Y) \times Dom(Z)$ are disjoint subsets of colors (i.e., $\mathcal{C}^{+} \cap \mathcal{C}^{-} = \emptyset$) and for every pair of colors $(x, y, z), (x', y', z') \in \mathcal{C}^{+}$ (x, y, z) is not conflicting with (x', y', z') (i.e., if $x' = x$ and $y' = y$ then $z' = z$).

Fig. 17 Auxiliary procedures for the main ones in Fig. 18. Procedure *BuildDag* builds a single-source, single-sink DAG whose nodes are nonempty subsets of $G_{\mathbf{r}}$. Each subset is formed by tuples sharing the same value for *VT*, and thus function *Time* is well defined. Procedure *SourceSinkShortestPath* returns the shortest path from source to sink on the DAG provided by *BuildDag*. The solution is given as a set of nodes (i.e., subsets of $G_{\mathbf{r}}$), and it omits *source* and *sink* nodes

procedure SourceSinkShortestPath$(Nodes, Edges, Time, Weight)$

comment: returns the shortest path from a *source* node to a *sink* node in a Weighted DAG $(Nodes, Edges)$. The weight of each edge is provided by the function $Weight$. The topological order of the elements of $Nodes$ is provided by the function $Time$ whch is used for identifying the *source* $(Time(source) = -\infty)$ and the *sink* $(Time(sink) = +\infty)$ nodes. Finally, the shortest path is returned as a subset of $Nodes$.

for each $n \in Nodes$ $\begin{cases} Predecessor(n) \leftarrow nil \\ Value(n) \leftarrow +\infty \end{cases}$
$Value(source) \leftarrow 0$
$current\_time \leftarrow \min(Img(Time) \backslash \{-\infty\})$
while $current\_time < +\infty$
do $\begin{cases} \textbf{for each } n\{n' \in Nodes : Time(n') = current\_time\} \\ \quad \textbf{do} \begin{cases} Value(n) \leftarrow \min_{(n',n) \in Edges} Value(n') + Weight(n, n') \\ Predecessor(n) \leftarrow \min_{(n',n) \in Edges} n' \end{cases} \\ current\_time \leftarrow \min\{time \in Img(Time) : time > current\_time\} \end{cases}$
$current\_node \leftarrow Predecessor(sink)$
$shortest\_path \leftarrow \varnothing$
while $current\_node \neq source$
do $\begin{cases} shortest\_path \leftarrow shortest\_path \cup \{current\_node\} \\ current\_node \leftarrow Predecessor(current\_node) \end{cases}$
return $shortest\_path$

procedure BuildDag$(G_{\mathbf{r}}, k, \mathcal{C}^+, \mathcal{C}^-)$

comment: given a set $G_{\mathbf{r}}$ for which there exists $j \in Dom(J)$ such that for every $t \in G_{\mathbf{r}}$ we have $t[J] = j$. Let $CPS = (G_{\mathbf{r}}, \mathcal{C}^+, \mathcal{C}^-)$ The procedure returns the unfolded DAG for $\mathcal{L}^j_{CPS}$.

$Nodes \leftarrow \{S \subseteq G_{\mathbf{r}} : S \neq \varnothing \wedge \forall t \forall t' (t \in S \wedge t' \in S \rightarrow t[VT] = t'[VT])\}$
$EndNodes \leftarrow \{source, sink\}$
$AllNodes = Nodes \cup EndNodes$
let $Time : AllNodes \rightarrow Dom(VT) \cup \{-\infty, +\infty\}$ **s.t.** $Time(S) = \begin{cases} t[VT] & S \in Nodes \wedge t \in S \\ -\infty & S = source \\ +\infty & S = sink \end{cases}$
$Edges \leftarrow \begin{array}{l} \{(source, S) : S \in Nodes\} \cup \{(S, sink) : S \in Nodes\} \\ \cup \{(S, S') : S \in Nodes, S' \in Nodes, Time(S') - Time(S) > k\} \end{array}$
$Edges \leftarrow \left\{(S, S') : \forall t \forall t' \left(\begin{pmatrix} t \in S \wedge t' \in S' \wedge \\ t'[VT] - t[VT] > 0 \end{pmatrix} \rightarrow \begin{pmatrix} \neg ColorConflict((t[X], t'[Y], t'[Z]), \mathcal{C}^+) \\ \wedge (t[X], t'[Y], t'[Z]) \notin \mathcal{C}^- \end{pmatrix} \right) \right\}$
let $Weight : Edges \rightarrow \mathbb{N}$ **s.t.**
$Weight(S, S') = \begin{cases} \sum_{t \in G_{\mathbf{r}} : Time(S) < t[VT] \leqslant Time(S') \wedge t \notin S'} ||t|| & \text{if } S, S' \in Nodes \\ \sum_{t \in G_{\mathbf{r}} : t[VT] \leqslant Time(S') \wedge t \notin S'} ||t|| & \text{if } S = source \\ \sum_{t \in G_{\mathbf{r}} : t[VT] > Time(S) \wedge t \notin S'} ||t|| & \text{if } S' = sink \end{cases}$
return $(Nodes \cup EndNodes, Edges, Time, Weight)$

Let $CPS = (G_{\mathbf{r}}, \mathcal{C}^+, \mathcal{C}^-)$ be a color-partial solution: we say that a solution $(G_{\mathbf{r}}, G_{\mathbf{r}} \backslash G^-_{\mathbf{r}}, G^-_{\mathbf{r}})$ is *induced by CPS* if and only if the two following conditions hold:

1. for every color (x, y, z) in $\mathcal{C}^+$ and for each edge $(t, t') \in G_{\mathbf{r}} \times G_{\mathbf{r}}$ for which $c(t, t') = (x, y, z')$, if (t, t') is active then $z' = z$.
2. for every color (x, y, z) in $\mathcal{C}^-$ and for each edge $(t, t') \in G_{\mathbf{r}} \times G_{\mathbf{r}}$, if $c(t, t') = (x, y, z)$ then (t, t') is not active in $(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$. It means that one of the following two conditions holds:
   - $t \in G^-_{\mathbf{r}}$ or $t' \in G^-_{\mathbf{r}}$;
   - there exists a tuple $t'' \in G^+_{\mathbf{r}}$ such that $t[VT] < t''[VT] < t'[VT]$ and $t''[J] = t[J]$ ($t[VT]$ and $t'[VT]$ are not consecutive in the current partial solution for the J-group with value $t[J]$).

An induced solution $(G_{\mathbf{r}}, G_{\mathbf{r}} \backslash G^-_{\mathbf{r}}, G^-_{\mathbf{r}})$ by *CPS* is *minimal* if and only if an induced solution $(G_{\mathbf{r}}, G_{\mathbf{r}} \backslash \overline{G^-_{\mathbf{r}}}, \overline{G^-_{\mathbf{r}}})$ by *CPS* does not exist with $|G^-_{\mathbf{r}}| > |\overline{G^-_{\mathbf{r}}}|$. In this case, we say that $(G_{\mathbf{r}}, G_{\mathbf{r}} \backslash G^-_{\mathbf{r}}, G^-_{\mathbf{r}})$ is an *induced minimal solution*. Since all the induced minimal solutions by *CPS* have the same size, we say that the *value of CPS*, denoted by *val*(*CPS*), is the value $||G^-_{\mathbf{r}}||$, where $(G_{\mathbf{r}}, G_{\mathbf{r}} \backslash G^-_{\mathbf{r}}, G^-_{\mathbf{r}})$ is a minimal induced partial solution.

Fig. 18 Main procedure for the edge-wise checking of a *APE* -FD $[\Delta_k(\tau_{G_\mathbf{r}}^{XYZcount})]X\overline{Y}\xrightarrow{\epsilon}\overline{Z}$. The procedure returns the minimum number of tuples to delete in $\mathbf{r}$ in order to obtain an instance $\mathbf{r}' \subseteq \mathbf{r}$ such that $\mathbf{r}' \models [\Delta_k(\tau_{\mathbf{r}}^{R})]X\overline{Y} \rightarrow \overline{Z}$. Like procedure *TupleWiseMin* of Figure 15, the initial call to the recursive procedure is *EdgeWiseMin*$(G_\mathbf{r}, k)$ with $\mathcal{C}^+$, $\mathcal{C}^-$, and *optimal* initialized to their respective default values

Algorithm 6.2: EdgeWiseMin$(G_\mathbf{r}, k, \mathcal{C}^+ = \varnothing, \mathcal{C}^- = \varnothing, optimal = +\infty)$

procedure PartialSolution$(G_\mathbf{r}, k, \mathcal{C}^+, \mathcal{C}^-)$
comment: let $CPS = (G_\mathbf{r}, \mathcal{C}^+, \mathcal{C}^-)$, the procedure returns a pair $(Value, Colors)$ for which $Value = val(CPS)$ and there exists a minimal solution $(G_\mathbf{r}, G_\mathbf{r}\backslash G_\mathbf{r}^-, G_\mathbf{r}^-)$ induced by $(G_\mathbf{r}, \mathcal{C}^+, \mathcal{C}^-)$ for which $Colors = colors(G_\mathbf{r}, G_\mathbf{r}\backslash G_\mathbf{r}^-, G_\mathbf{r}^-)$.

$Value \leftarrow 0$
$Colors \leftarrow \varnothing$
for each $j \in \{t[J] : t \in G_\mathbf{r}\}$
do
- $G_\mathbf{r}^j \leftarrow \{t \in G_\mathbf{r} : t[J] = j\}$
- $ShortestPath \leftarrow SourceSinkShortestPath(BuildDAG(G_\mathbf{r}^j, k, \mathcal{C}^+, \mathcal{C}^-))$
- $\overline{G}_\mathbf{r} \leftarrow \bigcup_{S \in ShortestPath} S$
- $Value \leftarrow Value + \sum_{t \in \overline{G}_\mathbf{r}} ||t||$
- $ActiveEdges \leftarrow \left\{(t,t') : \begin{array}{l} t \in \overline{G}_\mathbf{r} \wedge t' \in \overline{G}_\mathbf{r} \wedge 0 < t'[VT] - t[VT] \leqslant k \wedge \\ \forall t''(t'' \in \overline{G}_\mathbf{r} \rightarrow t''[VT] \leqslant t[VT] \vee t''[VT] \geqslant t'[VT]) \end{array}\right\}$
- $Colors \leftarrow Colors \cup \{(t[X], t'[Y], t'[Z]) : (t,t') \in ActiveEdges\}$

return $(Value, Colors)$

main
comment: returns the minimum value $m = \min ||G_\mathbf{r}\backslash G'_\mathbf{r}||$ for $G'_\mathbf{r}$ in $\{G''_\mathbf{r} \subseteq G_\mathbf{r} : G'_\mathbf{r} \models [\tau_{G_\mathbf{r}}^{XYZcount}]X\overline{Y} \rightarrow \overline{Z}\}$.

$m \leftarrow 0$
$Colors \leftarrow \varnothing$
for each $g \in \{g : \exists(t \in G_\mathbf{r})(t[J] = g)\}$
do
- $(Value, C) \leftarrow PartialSolution(Gbr, k, \mathcal{C}^+, \mathcal{C}^-)$
- $m \leftarrow m + Value$
- $Colors \leftarrow Colors \cup C$

if $m \geqslant optimal$ **then return** $optimal$
for each $c \in Colors$
- **for each** $c' \in Colors$
- **if** $c[X] = c'[X] \wedge c[\overline{Y}] = c'[\overline{Y}] \wedge c[\overline{Z}] \neq c'[\overline{Z}]$
- **then**
  - $m_c \leftarrow EdgeWiseMin(G_\mathbf{r}, k, \mathcal{C}^+ \cup \{c\}, \mathcal{C}^-, optimal)$
  - $m_{\neg c} \leftarrow EdgeWiseMin(G_\mathbf{r}, k, \mathcal{C}^+, \mathcal{C}^- \cup \{c\}, optimal)$
  - $m \leftarrow \min(m_c, m_{\neg c})$
  - **return** $\min(m, optimal)$

return m

In an opposite way from *TupleWiseMin*, in this algorithm color-partial solutions induce complete minimal solutions. However, such solutions may be inconsistent. The algorithm tries to obtain consistency by either forcing or forbidding one color at a time. This is done by means of sets $\mathcal{C}^+$ and $\mathcal{C}^-$, which are both initialized to $\emptyset$ at the beginning of the procedure. As we informally said above, if a color (x, y, z) belongs to $\mathcal{C}^+$, it means that the current partial solution must avoid all the active edges (t, t') such that $t[X] = x$, $t'[Y] = y$, and $t'[Z] \neq z$; if a color (x, y, z) belongs to $\mathcal{C}^-$, it means that the current partial solution must avoid all the active edges (t, t') such that $t[X] = x$, $t'[Y] = y$, and $t'[Z] = z$.

As a general overview of the algorithm, let us consider the following simplified procedure (let $CPS = (G_\mathbf{r}, \mathcal{C}^+, \mathcal{C}^-)$ to be the current color-partial solution):

1. compute a minimal solution $(G_\mathbf{r}, G_\mathbf{r}\backslash G_\mathbf{r}^-, G_\mathbf{r}^-)$ induced by CPS;
2. if $(G_\mathbf{r}, G_\mathbf{r}\backslash G_\mathbf{r}^-, G_\mathbf{r}^-)$ is consistent, return $val(CPS) = ||G_\mathbf{r}^-||$;
3. if $(G_\mathbf{r}, G_\mathbf{r}\backslash G_\mathbf{r}^-, G_\mathbf{r}^-)$ is inconsistent, let (t, t') be an active edge in $(G_\mathbf{r}, G_\mathbf{r}\backslash G_\mathbf{r}^-, G_\mathbf{r}^-)$, such that there exists an active edge (t'', t''') in $(G_\mathbf{r}, G_\mathbf{r}\backslash G_\mathbf{r}^-, G_\mathbf{r}^-)$, which is conflicting with (t, t'). Then, return the minimum value between those returned by two recursive calls, one where $\mathcal{C}^+$ is updated to $\mathcal{C}^+ \cup \{(t[X], t'[Y], t'[Z])\}$, and the other one where $\mathcal{C}^-$ is updated to $\mathcal{C}^- \cup \{(t[X], t'[Y], t'[Z])\}$.

Two observations are omitted in the above procedure w.r.t. function *EdgeWiseMin*. The first one is that the procedure does not take into account the fact that the value $||G_\mathbf{r}^-||$ of the color-partial solution computed at point 1 is a lower bound for the optimal solution that may be achieved in the current branch of the computation. Procedure *EdgeWiseMin* uses it in a classical branch-and-bound fashion by propagating the value of the current optimal solution (if any) in the tree of recursive calls (in Fig. 18 this is done by means of parameter *optimal*). In step 1., if the computed value $||G_\mathbf{r}^-||$ is greater than the optimal one, we immediately return from the recursive call, because no better solution may be found.

The last omitted observation regards how the value of the color-partial solution $val(CPS)$ is computed, where $CPS = (G_\mathbf{r}, \mathcal{C}^+, \mathcal{C}^-)$. For every J-group in $G_\mathbf{r}$, i.e. any set of tuples having value j for attribute J, we build

the following *wL-DAG* $\mathcal{L}^j_{CPS} = (V^j, E^j, \mathcal{L}^j, \mathcal{W}^j)$ where $V^j = \{t \in G_{\mathbf{r}} : t[J] = j\}$. For each $t \in V^j$, we have $\mathcal{W}^j(t) = t[count]$ and $\mathcal{L}^j(t) = t[VT]$, and

$$E^j = \begin{array}{c} \{(t,t') \in V^j \times V^j : t'[VT] - t[VT] > k\} \\ \cup \\ \{(t,t') \in V^j \times V^j : 0 < t'[VT] - t[VT] \leq k \wedge c(t,t') \in \mathcal{C}^+\} \\ \cup \\ \left\{(t,t') \in V^j \times V^j : \begin{array}{l} 0 < t'[VT] - t[VT] \leq k \wedge c(t,t') \notin \mathcal{C}^- \wedge \\ \forall x \forall y \forall z((x,y,z) \in \mathcal{C}^+ \rightarrow c(t,t') \text{ is not conflicting with } (x,y,z) \end{array}\right\} \end{array}$$

Let $J(G_{\mathbf{r}})$ be the set $\{j : \exists t(t \in G_{\mathbf{r}} \wedge t[J] = j)\} = \{j_1, \ldots, j_h\}$. For every $1 \leq i \leq h$, let M^i_{CPS} be the maximum value for which the *wL-DAG* $\mathcal{L}^{j_i}_{CPS}$ admits an M^i_{CPS}-thick path. The following result is straightforward.

Theorem 4 *For every color-partial solution* $CPS = (G_{\mathbf{r}}, \mathcal{C}^+, \mathcal{C}^-)$ *we have*

$$val(CPS) = ||G_{\mathbf{r}}|| - \sum_{j\ J(G_{\mathbf{r}})} M^j_{CPS}$$

Theorem 4 tells us that for computing $val(CPS)$, we need to compute for every $j \in J(G_{\mathbf{r}})$ the maximum value M^j_{CPS}, for which $\mathcal{L}^j_{CPS}$ admits an M^j_{CPS}-thick path. Given a *wL-DAG* $\mathcal{L}_{\mathcal{G}}$, we will call MAX-ThickPath (Max-TP for short) the problem of finding the maximum M for which $\mathcal{L}_{\mathcal{G}}$ admits a M-Thick Path. Max-TP may be solved by a simple dichotomic search having a decision procedure that solves the problem M-TP (Problem 4), which is NP-Complete [10]. Here, our assumption comes into play and allows us to find M^j_{CPS} for every $j \in J(G_{\mathbf{r}})$ in a "reasonable" time. Indeed, in the instances that are used to prove the NP-completeness, the number of nodes in any layer, roughly corresponding to the number of tuples of the given relation at a corresponding time point, is supposed to increase as the number of time points/layers increases.

This is not the case in many daily applications, especially in the clinical domain, where we may have a great number of tuples but scattered along the timeline. In thefollowing, we will provide a formal definition of our assumption. Given $j \in J(G_{\mathbf{r}})$ we define $VT(G_{\mathbf{r}}, j) = \{vt : \exists t(t \in G_{\mathbf{r}} \wedge t[VT] = vt \wedge t[J] = j)\}$, $MaxLevel(G_{\mathbf{r}}, j) = \max_{vt \in VT(G_{\mathbf{r}}, j)} |\{t : t[J] = j \wedge t[VT] = vt\}|$, $MaxCount(G_{\mathbf{r}}, j) = \max_{vt \in VT(G_{\mathbf{r}}, j)} \left(\sum_{t \in G_{\mathbf{r}} \wedge t[VT] = vt \wedge t[J] = j} ||t||\right)$ and the value $space\ (j, G_{\mathbf{r}})$ as the value $2^{MaxLevel(G_{\mathbf{r}}, j)} \cdot |VT(G_{\mathbf{r}}, j)| \cdot \log_2(MaxCount(G_{\mathbf{r}}, j))$. Let $MaxSpace(G_{\mathbf{r}}) = \max_{j \in J(G_{\mathbf{r}})} space(j, G_{\mathbf{r}})$. We will see that *EdgeWiseMin* is applicable to our instance, if we have $\mathcal{O}(MaxSpace(G_{\mathbf{r}})^2)$ bits for computing it. The problem with $MaxSpace(G_{\mathbf{r}})$ is that it is exponential in $MaxLevel(G_{\mathbf{r}}, j)$, but this value depends on the maximum number of tuples that shares the same values for attributes VT and J and differs on at least one among the attributes X, Y and Z in the original instance $\mathbf{r}$. As we say above, we assume this value to be manageable as it happens in many real-world applications. Hereinafter, we will suppose to have $\mathcal{O}(MaxSpace(G_{\mathbf{r}})^2)$ bits for performing our computation.

Let us suppose to have a *wL-DAG* $\mathcal{L}_{\mathcal{G}} = (V, E, \mathcal{L}, \mathcal{W})$, and we want to solve MAX-TP on it. Given a *wL-DAG* $\mathcal{L}_{\mathcal{G}} = (V, E, \mathcal{L}, \mathcal{W})$, a subset $V' \subseteq V$ is a *level-subset* if and only if $V' \neq \emptyset$ and $\mathcal{L}(v) = \mathcal{L}(v')$ for every $v, v' \in V'$. Let $\mathcal{V} = \{V' \subseteq V : V' \text{ is a level-subset}\}$. By definition of level subset, the function $\mathcal{L}' : \mathcal{V} \rightarrow \mathbb{N}$, such that for every $V' \in \mathcal{V}$ we have $\mathcal{L}'(V') = \mathcal{L}(v)$ for some $v \in V'$, turns out to be well defined. We define the unfolding of *wL-DAG* $\mathcal{L}_{\mathcal{G}} = (V, E, \mathcal{L}, \mathcal{W})$ as the weighted DAG $U(\mathcal{L}_{\mathcal{G}}) = (\mathcal{V} \cup \{source, sink\}, E', \mathcal{W}_{E'})$, where

$$E' = \begin{array}{c} \{(source, V') : V' \in \mathcal{V}\} \cup \{(V', sink) : V' \in \mathcal{V}\} \cup \\ \{(V', V'') \in \mathcal{V} \times \mathcal{V} : \forall v \forall v'(v \in V' \wedge v' \in V'' \rightarrow (v, v') \in E)\} \end{array},$$

For every $(source, V') \in E'$, we have

$$\mathcal{W}'(source, V') = \sum_{v \in V \setminus V' : \mathcal{L}(v) \leq \mathcal{L}(V')} \mathcal{W}(v),$$

For every $(V', sink) \in E'$, we have

$$\mathcal{W}'(V', sink) = \sum_{v \in V \mathcal{L}'(V') < \mathcal{L}(v)} \mathcal{W}(v),$$

And for every $(V', V'') \in E'$ with $V', V'' \in \mathcal{V}$ we have

$$\mathcal{W}'(V', V'') = \sum_{v \in V \setminus V'' : \mathcal{L}'(V') < \mathcal{L}(v) \mathcal{L}'(V'')} \mathcal{W}(v).$$

For instance, the DAG in Fig. 19 is the result of unfolding the *wL-DAG* in Fig. 11. The unfolding of a *wL-DAG*, in the worst-case scenario, is exponential in the size of $\mathcal{L}_{\mathcal{G}}$. Given a *wL-DAG* $\mathcal{L}_{\mathcal{G}} = (V, E, \mathcal{L}, \mathcal{W})$, we define $W_{all}(\mathcal{L}_{\mathcal{G}}) = \sum_{v \in V} \mathcal{W}(v)$ as the sum of all the weights associated with nodes in V. It is straightforward to prove that the union of all the internal nodes in a *source*-to-*sink* path in the unfolding of a *wL-DAG* $\mathcal{L}_{\mathcal{G}}$ is a thick path in $\mathcal{L}_{\mathcal{G}}$ and, on the other hand, every thick path in $\mathcal{L}_{\mathcal{G}}$ may be associated with a *source*-to-*sink* path in its unfolding. Moreover, for every *source*-to-*sink* path p in the unfolding of $\mathcal{L}_{\mathcal{G}}$, let w_p be its weight. The weight of the thick path associated with p is exactly $W_{all}(\mathcal{L}_{\mathcal{G}}) - w_p$ (i.e., $\mathcal{L}_{\mathcal{G}}$ admits a thick path with value $W_{all}(\mathcal{L}_{\mathcal{G}}) - w_p$). With these premises, we can prove the following result.

Theorem 5 *Given a wL-DAG* $\mathcal{L}_{\mathcal{G}} = (V, E, \mathcal{L}, \mathcal{W})$, *let w the value of the shortest source-to-sink path in its unfolding. We have that the value of MAX-TP on* $\mathcal{L}_{\mathcal{G}} = (V, E, \mathcal{L}, \mathcal{W})$ *is equal to* $W_{all} - w$.

Given a color-partial solution $CPS = (G_{\mathbf{r}}, \mathcal{C}^+, \mathcal{C}^-)$, procedure *EdgeWise* computes the value $val(CPS)$ (performed

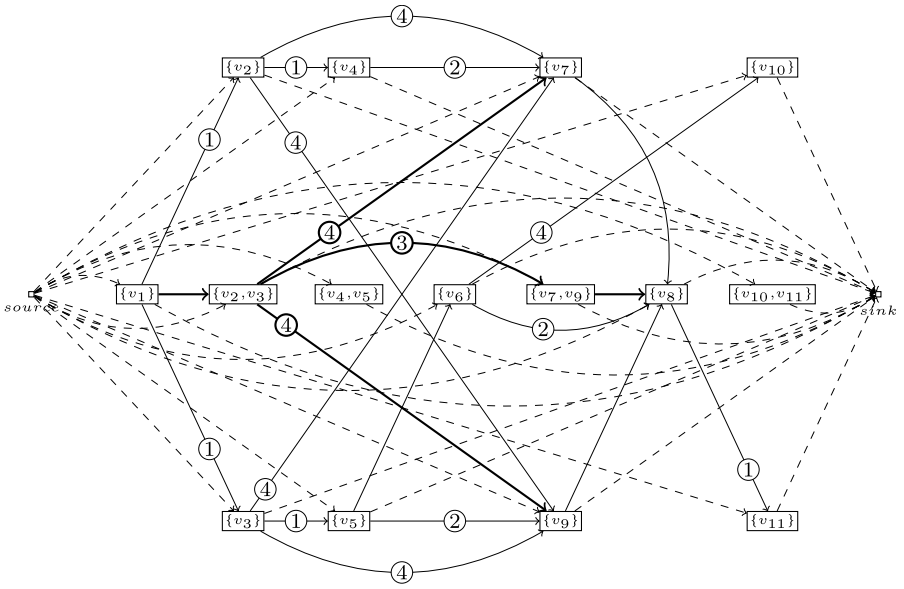


Fig. 19 The unfolding of the *wL-DAG* of Figure 11 into a weighted DAG for solving the MAX-TP problem. The table below the graph provides the weights for source-to-node edges and node-to-sink edges, which are both represented by dashed lines. Continuous edges without labels have weight 0. $P = source\{v_1\}\{v_2, v_3\}\{v_7, v_9\}\{v_8\}\{v_{11}\}sink$ is a source-sink shortest path with value 4

V'	$\mathcal{W}'(source,V')$	$\mathcal{W}'(V',sink)$	V'	$\mathcal{W}'(source,V')$	$\mathcal{W}'(V',sink)$	V'	$\mathcal{W}'(source,V')$	$\mathcal{W}'(V',sink)$
$\{v_1\}$	0	10	$\{v_6\}$	5	5	$\{v_{11}\}$	10	0
$\{v_2\}$	2	8	$\{v_7\}$	7	3	$\{v_2,v_3\}$	1	8
$\{v_3\}$	2	8	$\{v_8\}$	8	2	$\{v_4,v_5\}$	3	6
$\{v_4\}$	4	6	$\{v_9\}$	7	3	$\{v_7,v_9\}$	6	3
$\{v_5\}$	4	6	$\{v_{10}\}$	10	0	$\{v_{10},v_{11}\}$	9	0

by procedure *PartialSolution* in Fig. 18) summing up all the values M^j_{CPS} for every $j \in J(G_{\mathbf{r}})$. Each M^j_{CPS} is computed as value of a *source*-to-*sink* shortest path (performed by procedure *SourceSinkShortestPath* in Figure 17) on the unfolding of $\mathcal{L}^j_{CPS}$ (built by procedure *BuildDag* in Fig. 17). For building $U(\mathcal{L}^j_{CPS})$, on which we will compute value of a *source*-to-*sink* shortest path, we may need $\mathcal{O}(MaxSpace(G_{\mathbf{r}})^2)$ bits.

Finally, let us observe that procedure *PartialSolution* does not return only the value $val(CPS)$ of the current color-partial solution $CPS = (G_{\mathbf{r}}, \mathcal{C}^+, \mathcal{C}^-)$. Since it effectively computes a minimal solution $(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$ induced by CPS in order to provide $val(CPS)$, it returns the set $colors(G_{\mathbf{r}}, G_{\mathbf{r}} \backslash G^-_{\mathbf{r}}, G^-_{\mathbf{r}})$, that is, the set of all and only the colors associated with active edges in $(G_{\mathbf{r}}, G_{\mathbf{r}} \backslash G^-_{\mathbf{r}}, G^-_{\mathbf{r}})$. If such a set does not contain two colors (x, y, z) and (x, y, z') such that $z \neq z'$, then we have that $(G_{\mathbf{r}}, \mathcal{C}^+, \mathcal{C}^-)$ is a consistent solution and we may return $val(CPS)$. Otherwise, procedure *EdgeWiseMin* takes a color (x, y, z) in $colors(G_{\mathbf{r}}, G_{\mathbf{r}} \backslash G^-_{\mathbf{r}}, G^-_{\mathbf{r}})$ such that there exists (x, y, z') in $colors(G_{\mathbf{r}}, G_{\mathbf{r}} \backslash G^-_{\mathbf{r}}, G^-_{\mathbf{r}})$ with $z \neq z'$ and performs two recursive calls, one in which $\mathcal{C}^+$ is updated to $\mathcal{C}^+ \cup \{(x, y, z)\}$ and the other in which $\mathcal{C}^-$ is updated to $\mathcal{C}^- \cup \{(x, y, z)\}$.

Mining *APE*-FDs

In this section, we consider the problem of mining *APE* -FDs on a given instance $\mathbf{r}$ of a temporal schema R from a practical point of view. We will describe a prototype that performs such task. In particular, we point out two big computational challenges we addressed in the implementation of our prototype.

Let us start with the formal definition of our problem. Given a temporal schema $R = U \cup \{VT\}$, an instance $\mathbf{r}$ of R, a nonempty set $J \subseteq U$, a threshold $0 \leq \epsilon \leq 1$, and a value $k \in \mathbb{N}$, we denote as $\mathcal{PE}(\mathbf{r}, k, \epsilon)$ the set of all the *APE*-FDs $[\Delta_k(\tau^R_J)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, formally introduced in "Discovering Pure Temporally Evolving Functional Dependencies" section, such that $\mathbf{r} \models [\Delta_k(\tau^R_J)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$.

A set $\mathcal{S}$ of *APE*-FDs is *complete* if and only if for every $[\Delta_k(\tau^R_J)]\ X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ belonging to $\mathcal{S}$, there exists $X' \subseteq X$ and $\overline{Y}' \subseteq \overline{Y}$ such that $[\Delta_k(\tau^R_J)]\ X'\overline{Y}' \overset{\epsilon}{\rightarrow} \overline{Z} \in \mathcal{PE}(\mathbf{r}, k, \epsilon)$. A complete *APE*-FD-set $\mathcal{PE}(\mathbf{r}, k, \epsilon)$ is *minimal* if and only if for every $[\Delta_k(\tau^R_J)]\ X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z} \in \mathcal{S}$, set $\mathcal{S} \backslash \{[\Delta_k(\tau^R_J)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}\}$ is not complete anymore. Given a complete minimal *APE*-FD-set $\mathcal{PE}(\mathbf{r}, k, \epsilon)$, every subset $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon) \subseteq \mathcal{PE}(\mathbf{r}, k, \epsilon)$ is called a *minimal partial*.

Given a temporal schema $R = U \cup \{VT\}$, an instance $\mathbf{r}$ of R, a nonempty set $J \subseteq U$, a threshold $0 \leq \epsilon \leq 1$ and a value $k \in \mathbb{N}$, we are interested in finding a minimal complete set

$\mathcal{PE}(\mathbf{r}, k, \epsilon)$. However, in order to do that, we have to deal with two computational problems:

- the smallest minimal complete set $\mathcal{S}$ may be exponential in the size of U with respect to some given temporal schemata $R = U \cup \{VT\}$, instances $\mathbf{r}$ of R, nonempty sets $J \subseteq U$, thresholds $0 \leq \epsilon \leq 1$, and values $k \in \mathbb{N}$;
- given a single *APE*-FD $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ on a schema $R = U \cup \{VT\}$, and an instance $\mathbf{r}$ of R, deciding whether or not $\mathbf{r} \models [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ is a NP-complete problem.

The first result may be derived by leveraging a result of Kivinen et al. [16] on approximate functional dependencies. The second result is proved in "The Computational Complexity of Checking *APE*-FD" section. However, such theoretical bounds are both difficult to achieve in real-world domains. For instance, the size of $\mathcal{PE}(\mathbf{r}, k, \epsilon)$ could be exponential in $|U|$, but in real-case scenarios, we have that $|U|$ is often less than 50/60 elements. Moreover, the instance built in [16] for achieving the exponential lower bound does not occur in real-world instances. The complexity of checking a single *APE*-FD is even worse. Indeed, checking a single *APE*-FD is NP-Complete in the number of tuples, which may be very high and increasing time after time. This problem is known as the *curse of cardinality*, and its relevance has been recently considered for temporal inference of sequential patterns in [18]. Even in this case, the instance specified in "The Computational Complexity of Checking *APE*-FD" section for proving the NP-hardness result has been built in a very complex and constrained way. Such an instance does not even remotely resemble some real-world scenario. Thus, we are allowed to design and implement a prototype for the practical mining of such dependencies on real-world datasets and evaluate its performances.

Prototype Overview

Even though the results reported in "The Computational Complexity of Checking *APE*-FD" section are not completely encouraging and according to the last comments in the previous section, we developed a prototype. Given a temporal schema $R = U \cup \{VT\}$, an instance $\mathbf{r}$ of R, a nonempty set $J \subseteq U$, a threshold $0 \leq \epsilon \leq 1$ and a value $k \in \mathbb{N}$, it returns a minimal complete set $\mathcal{PE}(\mathbf{r}, k, \epsilon)$. The prototype is named *Attila*(*Approximate Temporal Tailored Inference Lean Application*). In the following, after providing a high-level description of *Attila*modules and their interaction, we focus with a detailed description on novel ideas underlying the design of the prototype. *Attila*was implemented according to the principles of distributed programming. Different tasks are executed by different processes (possibly executed on different machines). *Attila* is composed by three main processes:

- Worker is responsible for maintaining a representation of the minimal partial $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$ of $\mathcal{PE}(\mathbf{r}, k, \epsilon)$. Thus, Worker maintains a compact representation both of the set of the *APE*-FDs already checked and of the *APE*-FDs that remain to be checked. It updates its state according to the last *APE*-FD $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ that has been checked. The next state depends on the fact that either $\mathbf{r} \models [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ or $\mathbf{r} \setminus \models [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$. In both cases, Worker marks $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ to not be checked anymore. In this way, it can determine precisely the next *APE*-FD to check;
- Contributor has to check a single *APE*-FD against the instance $\mathbf{r}$.It retrieves *APE*-FD from Worker. When Contributor gets an *APE*-FD $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, it schedules jobs among the computational units that will effectively check the given *APE*-FD;
- Sub-Contributor is the basic computational unit and depends on Contributor. Sub-Contributor leverages the graph representation generated by Contributor for resolving a subproblem of the original one. More precisely, it receives a subproblem of the one stored by Contributor for checking a given $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ against it. During the computation, Contributor may send a reduction of the subproblem Sub-Contributor is dealing with. Indeed, Contributor may assign portions of a problem to several Sub-Contributors.

Processes composing *Attila*are hierarchically organized. A Worker could manage minimal partial $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$, but more Contributors are needed for checking multiple dependencies at the time. Each Contributor may have several Sub-Contributors in order to speed up the checking procedure.

Now, let us consider in some detail each process type, in order to give a general idea of how computations are handled. Worker has to manage the minimal partial $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$. At the end of computation, $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$ turns out to be a minimal complete set, according to the goal of our distributed procedure. Worker interacts only with its pool of Contributors. Figure 20 depicts how such interaction happens, by a BPMN choreography [22]. A Contributor can register to the Worker at any time incrementing by one the number of *APE*-FDs that may be checked simultaneously. Worker may receive *APE*-FD request only by registered Contributors. To this regard, it keeps two auxiliary *APE*-FD-sets *Pending* and *Assigned* $\subseteq$ *Pending*. *Pending* is a minimal set of *APE*-FDs such that $Pending \cap \overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon) = \emptyset$ and, if for every $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z} \in Pending$ $\mathbf{r} \models [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, then $Pending \cup \overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$ is a minimal complete *APE* -FD-set. *Pending* contains all *APE*-FDs among whom Worker must choose every time it is asked for an *APE*-FD by some of its Contributors. *Assigned* represents all *APE* -FDs in *Pending* already assigned to Contributors. Every time a Contributor returns the result of *APE*-FD-check

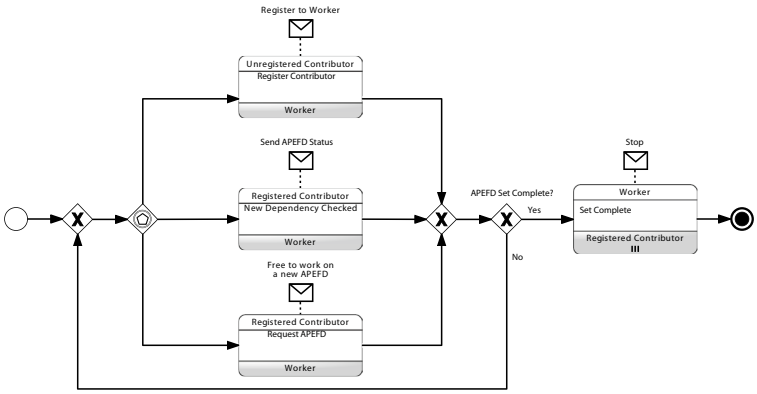


Fig. 20 A BPMN choreography showing the interaction between a Worker and its (possibly) many Contributors

on an element of *Assigned*, *Pending* and $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$ are updated according to the fact that either $\mathbf{r} \models [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ or $\mathbf{r} \setminus \models [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$. More precisely, we have that $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ is removed from both *Pending* and *Assigned*, and

- if $\mathbf{r} \models [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, then Worker inserts such *APE*-FD in $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$;
- if $\mathbf{r} \setminus \models [\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, then for each attribute $W \in U \setminus (X \cup Y \cup \{Z\})$, dependencies $[\Delta_k(\tau_J^R)]XW\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ and $X\overline{YW} \overset{\epsilon}{\rightarrow} \overline{Z}$ are inserted in *Pending*.

The main operations Worker performs are:

1. (i) it pulls an *APE*-FD $[\Delta_k(\tau_J^R)]\, X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ out of *Pending*,
2. it updates $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$ if needed, and
3. (iii) it updates *Pending* and *Assigned* as soon as the status of a new dependency is discovered.

We use *Ordered Binary Decision Diagrams* (OBDDs) [1] as data structures allowing an efficient execution of such operations. An OBDD is a single-rooted directed acyclic graph that represents a propositional formula ψ. A propositional variable is associated with every node as a label, except for the only terminal node 1.[2] Any nonterminal node v may have at most two outgoing edges $low(v)$ (dotted line, depicted in Fig. 21) and $high(v)$ (solid lines, depicted in Fig. 21). $low(v)$ ($high(v)$) means that variable v is taken with value 0 (1). A variable truth assignment is represented by a path from the root to terminal node 1. Thus, an OBDD represents the set of all truth assignments for a given formula ψ. Worker uses three different OBDDs corresponding to formulas $\psi_{\overline{\mathcal{PE}}}$, ψ_P and ψ_A, to keep track of sets $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$, *Pending* and *Assigned*, respectively. *APE*-FDs in $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$ correspond to all and only the solutions of formula $\psi_{\overline{\mathcal{PE}}}$ (and the same for *APE*-FDs in *Pending* with respect to formula ψ_P and for *APE*-FDs in *Assigned* with respect to ψ_A). Hereinafter, we will use ψ for denoting both the formula and the OBDD corresponding to all its possible solutions. Informally, updates of these three sets are implemented by adding conjuncts/disjuncts to their respective formulas.

For representing set $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$ as all and only the solutions of a formula, it suffices to assign to each attribute $X_i \in U$ three variables $x_i, \overline{x}_i, \vec{x}_i$. Then, formula $\psi_{\overline{\mathcal{PE}}}$ has be satisfied by assignments $\sigma : \{x_1, \ldots, \underset{\epsilon}{x_n}, \overline{x}_1, \ldots, \overline{x}_n, \vec{x}_1, \ldots, \vec{x}_n\} \rightarrow \{0, 1\}$. An *APE*-FD $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ belongs to $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$ if and only if the assignment $\sigma(x_i) = 1 \forall X_i \in X$, $\sigma(\overline{y}_j) = 1 \forall \overline{Y}_j \in \overline{Y}$, and $\sigma(\vec{z}) = 1$ satisfies $\psi_{\overline{\mathcal{PE}}}$. According to this approach, for instance, $a_1 \wedge a_2 \wedge \overline{a}_3 \wedge \vec{a}_4$ represents *APE*-FD$[\Delta_k(\tau_J^R)]A_1A_2\overline{A}_3 \rightarrow \overline{A}_4$. Clearly, if a formula $\psi_{\overline{\mathcal{PE}}}$ represents all the possible *APE*-FDs, an OBDD for $\psi_{\overline{\mathcal{PE}}}$ represents them as well. The same approach may be used for sets *Pending* and *Assigned*. Hereinafter, we refer to ψ as the OBDD representing all and only the assignments σ such that $\sigma \models \psi$.

A Worker begins the *APE*-FD mining task by initializing two OBBDs representing $\psi_{\overline{\mathcal{PE}}} = \psi_A = \bot$. Initially, $\psi_{\overline{\mathcal{PE}}} = \bot$ (i.e., $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon) = \emptyset$) since the distributed procedure has not discovered any valid *APE*-FD yet. ψ_P is true only for those assignments that represent well-formed *APE*-FDs. In order to extract from ψ_P an *APE*-FD to be tested, we take the solution associated with any root-to-terminal path in the OBDD for $\psi_P \wedge \neg\psi_A$. For inserting an *APE*-FD $[\Delta_k(\tau_J^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$

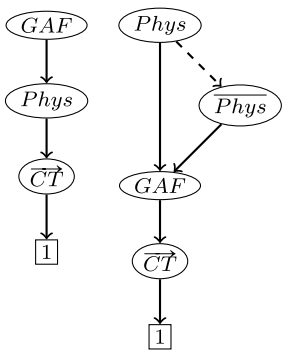


Fig. 21 The update of the set $\overline{\mathcal{PE}}(\mathbf{r}, \underset{\epsilon}{k}, \epsilon) = \{[\Delta_k(\tau_{PatId}^{Contact})]GAF, \underline{Phys \rightarrow CT}\}$ (left) into the set $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon) = \{[\Delta_k(\tau_{PatId}^{Contact})]GAF, \underset{\epsilon}{Phys \rightarrow CT}, [\Delta_k(\tau_{PatId}^{Contact})]GAF, \overline{Phys} \overset{\epsilon}{\rightarrow} \overline{CT}\}$ (right)

[2] We use OBDD without the 0 terminal node.

in $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$, it suffices to update $\psi_{\overline{\mathcal{PE}}}$ to $\psi_{\overline{\mathcal{PE}}} \vee \psi$ where $\psi = \bigwedge_{X_i \in X} x_i \wedge \bigwedge_{X_i \in Y} \overline{x}_i \wedge \vec{z}$. Moreover, for deleting a solution from *Pending* it suffices to update ψ_P:

- if we have that $\mathbf{r} \vDash [\Delta_k(\tau_j^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, then we update ψ_P to $\psi_P \wedge \neg\psi$;
- if we have that $\mathbf{r} \setminus \vDash [\Delta_k(\tau_j^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$, then we update ψ_P to $\psi_P \wedge \overline{\psi}$ where $\overline{\psi} = \vec{z} \rightarrow (\bigvee_{X_i \in U \setminus X} x_i \vee \bigvee_{X_i \in U \setminus Y} \overline{x}_i)$ (i.e., if $\vec{z}$ holds, then at least one of variables not contained in the formula for the given *APE*-FD must hold).

It is worth noting that we have a search operation for *APE* -FDs that is linear in $|U|$. Moreover, Boolean operations on OBDDs are implemented in very efficient way by many packages on the market (in our prototype, we used BuDDy [17]). This solution allows us to have a compact representation of sets of *APE*-FDs that can be manipulated efficiently.

Figure 21 shows an example of how $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$ is updated if we represent it through formula $\psi_{\overline{\mathcal{PE}}}$. In this example, we borrow two dependencies from the psychiatric case register introduced in a simplified way in "Discovering Pure Temporally Evolving Functional Dependencies" section and discussed in detail in "Mining *APE*-FDs on Clinical Domains" section. The real-world schema differs from the example since patients are identified by *PatId* attribute in place of their names. Furthermore, several attributes are used for storing information regarding registered calls. The most significant attribute is *Global Assessment of Functioning* (*GAF*): it is a numeric value provided by the physician at the end of the call, and it scores the patient's mental health status. Figure 21 shows an update operation on $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon) = \{[\Delta_k(\tau^{Contact}_{PatId})]GAF, Phys \overset{\epsilon}{\rightarrow} \overline{CT}\}$, where $\overline{\mathcal{PE}}(\mathbf{r}, k, \epsilon)$ is updated to $\overline{\mathcal{PE}}'(\mathbf{r}, k, \epsilon) = \{[\Delta_k(\tau^{Contact}_{PatId})]GAF, Phys \overset{\epsilon}{\rightarrow} \overline{CT}, [\Delta_k(\tau^{Contact}_{PatId})]GAF, \overline{Phys} \overset{\epsilon}{\rightarrow} \overline{CT}\}$. As one may notice, each node v has at most two outgoing edges, one solid and one dashed representing $high(v)$ and $low(v)$ respectively. Taking a solid edge $high(v)$ in a root-to-terminal path denotes that the attribute corresponding to v belongs to the dependency. Taking a dashed edge $low(v)$ in a root-to-terminal path denotes that the attribute corresponding to v does not belong to the dependency. As an example, the OBDD shown in Fig. 21 (right) features two paths from the root node, labeled *Phys*, to terminal node 1. Since the edge $(Phys, \overline{Phys})$ is dashed, the path $Phys, \overline{Phys}, GAF, \overrightarrow{CT}, 1$ represents the dependency $[\Delta_{+\infty}(\tau^{Contact}_{PatId})]GAF, \overline{Phys} \overset{\epsilon}{\rightarrow} \overline{CT}$. Such dashed edge implies that attribute *Phys* is not taken in X while it is taken in Y because the outgoing edge the path takes from $\overline{Phys}$ is a continuous one.

Let us consider now the Contributor process. Until now we just considered it as a process which asks Worker for a dependency and eventually answers whether $\mathbf{r} \vDash [\Delta_k(\tau_j^R)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ holds or not, as shown by the BPMN choreography in Fig. 20. Thus, Contributor is responsible for checking a single *APE*-FD at a time. Since the complexity is intractable (recall that the problem in NP-Complete), Contributor does not deal directly with the computation. Indeed, as mentioned before, it splits a problem among several computational units called Sub-Contributor *s*.

The way in which a Contributor deals with its pool of Sub-Contributors is very close to the interaction between Worker and its Contributors, and it is described by the BPMN choreography diagram provided in Fig. 22. The status of the problem is managed by Contributor, and it is represented by a binary tree, where each node is labeled with a tuple and its two children represent either the case in which the tuple is inserted in the current solution or it is removed from it. Subproblems are generated by asserting that a tuple belongs or not to the final solution. This procedure generates a tree. Initially, the whole tree is given to a single Sub-Contributor to visit. Suppose that a new Sub-Contributor registers himself to the same Contributor during a computation. Such Contributor selects the subproblem of a Sub-Contributor and a tuple t in $\mathbf{r}$. Contributor splits the subproblem into two parts, one in which t must belong to the solution and the other in which t does not belong to the solution. One portion is given to the new Sub-Contributor and the "old" Sub-Contributor is notified to reduce its problem to the other portion. Usually, we have multiple Sub-Contributors that work in a subtree rooted at the node where the reduce operation happens, and thus, as reported in Fig. 22, we have to notify all of them about the reduction.

Figure 23 shows an example of how Contributor works. Suppose that there is exactly one Sub-Contributor sc_1 that is exploring the tree (a) on the top of Fig. 23. At a certain point, a new Sub-Contributor sc_2 registers himself to the Contributor and requests a subproblem. Now Contributor looks at the active subproblem and chooses the one of sc_1. At the root of such problem, there is tuple t_1. Therefore, Contributor splits the subproblem into two more subproblems: one where t_1 is forced to be deleted (tree (b) in Fig. 23), and the other where t_1 is kept (tree (c) in Fig. 23). Finally, the exploration of subtree (c) is given to sc_2 and sc_1 is notified that its exploration of the tree (a) is reduced to the exploration of the subtree (b).

Sub-Contributor is the minimal computation unit: it simply performs tasks assigned by its master Contributor. Sub-Contributor listens constantly to its Contributor in order to receive reduction of its current subproblem for speeding up the process; meanwhile, it explores its current subproblem searching for a solution. Sub-Contributor operates in two symmetric ways that may be seen as two concurrent threads. The first thread assumes that its subproblem contains the solution and performs a depth-first search of the tree in order to find it. The other thread assumes that the subproblem does not contain the solution and tries to find a counterexample.

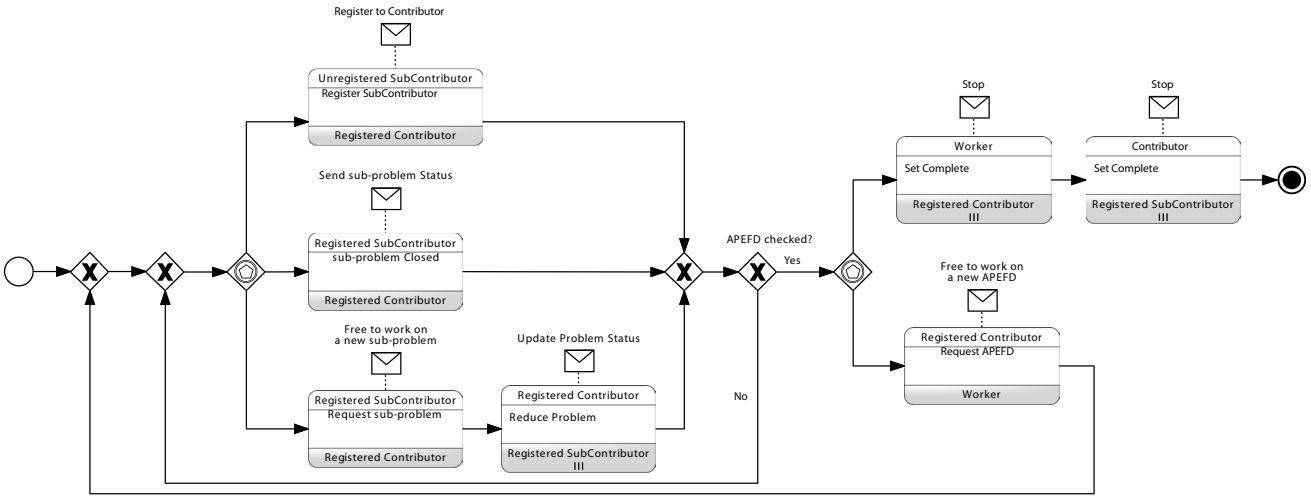


Fig. 22 A BPMN choreography showing the interaction between a Contributor and its (possibly) many Sub-Contributors

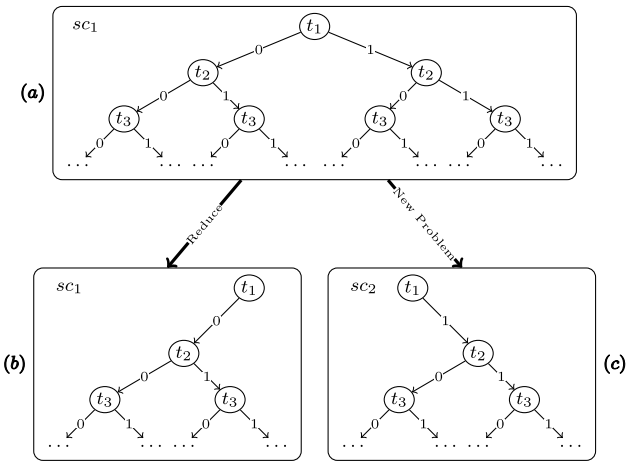


Fig. 23 A graphical account of how the tree is split among Sub-Contributors

In order to deal with the latter task, the Sub-Contributor translates the subproblem into a linear programming problem and verifies its feasibility. This symmetric approach turns out to be very efficient. *Attila*makes use of the open-source linear programming library *GNU Linear Programming Kit* (GLPK) [24] to perform such linear programming tasks.

Mining *APE*-FDs on Clinical Domains

In this section, we discuss *APE*-FDs obtained through *Attila*. These results may be also considered as an early validation of our prototype. As we already mentioned, we focused on two different clinical domains. The first one is that of *psychiatry*. In this domain, one of the main sources of information consists of data acquired during the (mainly, telephonic) contacts between patients and psychiatrists. "Discovering Pure Temporally Evolving Functional Dependencies" section provides a detailed description of this domain. *Attila* allowed us to extract the following *APE*-FDs from relation *Contact*:

- $[\Delta_{+\infty}(\tau_{PatId}^{Contact})]GAF \xrightarrow{\epsilon} \overline{numPsychologists}$ with $\epsilon = 0.1$. This dependency represents the fact that, considering two consecutive calls of the same patient, the number of psychologists involved in the second call uniquely depends on the GAF score of the patient during the first call. It may highlight that some (maybe implicit) policy determines the number of psychologists required for a contact, according to the conditions the given patient showed in the previous call.
- $[\Delta_{+\infty}(\tau_{PatId}^{Contact})]Service, GAF \xrightarrow{\epsilon} \overline{CT}$ with $\epsilon = 0.1$. Informally, it means that for each pair of consecutive calls for the same patient, the previous patient's GAF score and Service (clinical psychiatry, medical psychology, psychotherapy, ...) uniquely determines the next contact type (family member, a neighbor, the police, ...).
- $[\Delta_{+\infty}(\tau_{PatId}^{Contact})]GAF, Physician \xrightarrow{\epsilon} \overline{Request}$ with $\epsilon = 0.1$. This dependency says that the next actions on patients are mainly based on the previous GAF score and physician. For each pair of consecutive calls, the request (it could be group psychotherapy, family psychotherapy, legal medical evaluation, ...) decided during the second call depends on the physician and on the GAF score of the given patient during the first call.

The second clinical domain is that of *pharmacovigilance*, which is the science related to the management and prevention of suspected adverse reactions induced by drugs [34]. Premarketing trials are not able to discover all adverse reactions induced by the investigated drug. This is due to trial

limitations, e.g., short timespan of the study, highly selected test groups, and so on. Adverse drug reactions (ADRs) may, indeed, go undetected and become evident when the drug is already on the market [28]. Therefore, marketed drugs require a continuous monitoring of their possible effects. The spontaneous reporting of ADRs allows healthcare stakeholders to identify unexpected reactions and to inform regulating authorities about them. This practice is extremely important, provides early warnings and requires limited economic and organizational efforts and resources [21]. Among its multiple advantages, spontaneous reporting allows one to consider every drug on the market and any category of patients. It investigates possible relationships between one or more adverse reactions and one or more drugs. physicians, chemists or citizens are allowed to submit reports. The analysis focuses on unknown or completely undocumented relationships and may suggest a potential cause–effect link between ADRs and drugs, classified as "suspected" or "concomitant." Any report contains both demographic and specific pharmacovigilance data, as patient 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 [20], the entry date, the period of the adverse reaction and the periods of drug administrations. These temporal data are then processed and analyzed to possibly discover any cause–effect relationship among drugs and reaction(s) in different time periods, or according to the exposure timespan. In this case, we consider the evolution of reports for the same drug (by using *PhProd* (Pharmaceutical Product, i.e., active principle) for performing the join in the evolution expression). This way, we may observe whether the therapy decided by physicians is influenced by previous adverse reactions. As an example, the fact that physicians are aware of past cases of adverse events could determine changes in drug dosages. Changing the prescribed drug quantities for patients because of previously suspected drugs could be considered as attempt of avoiding such adverse reactions.

Among *APE*-FDs extracted from a recent instance of *Reports* schema of the Italian Network of Pharmacovigilance, we introduce here the following *APE*-FD.

- $[\Delta_{+\infty}(\tau^{Reports}_{PhProd})]PhProd, Dos \overset{\epsilon}{\rightarrow} \overline{Dos}$ with $\epsilon = 0.2$. Such a dependency may highlight that, when an ADR is reported, the dose is usually adjusted in the same way for most patients, depending on the previous administrations. Such *APE*-FD may suggest that Italian physicians methodically consider the Italian Network of Pharmacovigilance in managing drug therapies.

Performance Analysis

In this section, we present a short and preliminary performance analysis of *Attila* and of its components. We executed two kinds of test. The first test was done using a single machine. This way, we obtained a first evaluation of the time required for mining multiple *APE*-FDs on a large real-world database. The second test focused on a single *APE*-FD, but considering some distributed architecture, using a server and at most two distinct remote machines. This test allows us to observe whether the time required for checking a single *APE* -FD decreases, when the problem is distributed among different computational units.

We started by analyzing the performances of the whole system, when the computation is entirely done by a single machine. We tested *Attila* on an instance of schema *Contact*, consisting of approximately $1.5 \cdot 10^6$ rows. *APE*-FDs as $[\Delta_{+\infty}(\tau^{Contact}_{PatId})]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ were extracted, with a threshold $\epsilon = 0.1$. Figure 24 depicts the result of this first experiment. *Attila*verified almost 4500 *APE*-FDs in about 10 days. Figure 24 shows through a pie chart that checked *APE*-FDs (i.e., holding and not holding) are less than half of possible *APE*-FDs. Indeed, many *APE*-FDs denoted as *superset* are subsumed by the checked and holding *APE*-FDs. Thus, they have not been tested (i.e., only minimal *APE*-FDs have been tested). Moreover, Worker is often idling, as Contributors perform most of the computation. As described in "Mining *APE*-FDs" section, a Contributor has to visit a tree, which is exponentially large w.r.t. the instance size. Even though this operation is theoretically unfeasible for large instances, by employing simple pruning conditions (as, for example, too many tuples deleted, existence of violated constraints, and so on), the tree size may be reduced.

Finally, we analyzed the interactions between Contributor and Sub-Contributors by checking *APE*-FD $[\Delta_{+\infty}(\tau^{Reports}_{PhProd})]PhProd, Dos \overset{\epsilon}{\rightarrow} \overline{Dos}$ with $\epsilon = 0.2$, discussed in "Mining *APE*-FDs on Clinical Domains" section and related to the pharmacovigilance domain. Figure 25a depicts some comparisons between various configurations of Sub-Contributors when checking the given dependency. We considered five possible situations:

(1) a single local Sub-Contributor(Server), running on the same machine where Contributor is running;
(2) a single local Sub-Contributor, and a remote Sub-Contributor(Remote);
(3) two local Sub-Contributors and a single remote Sub-Contributor,
(4) two remote Sub-Contributors, i.e., two separate physical machines with identical hardware/specs running a Sub-Contributor each;
(5) a single local Sub-Contributor, and two remote Sub-Contributors.

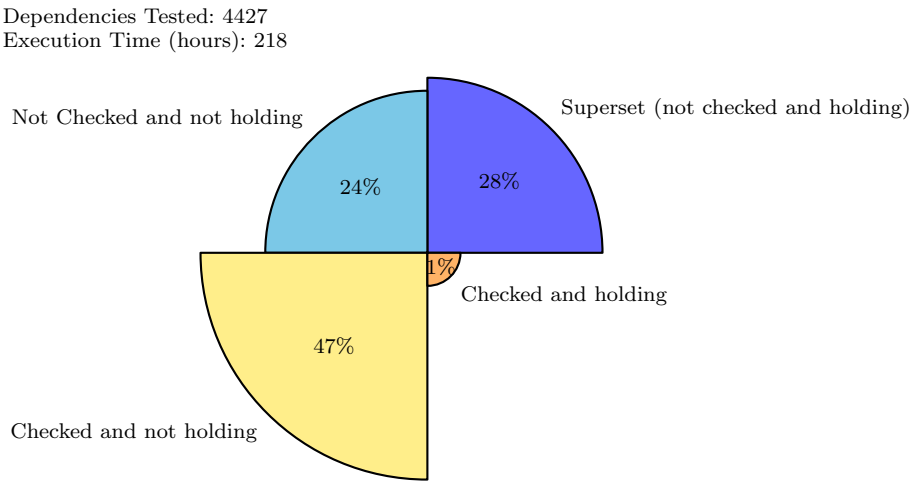


Fig. 24 Result of the execution of *Attila*on a single machine (Intel Core i3(TM) CPU M 330 2.13 GHz, 4GB) on an instance of table *Contact* ($\sim 1.5 \cdot 10^6$ rows)

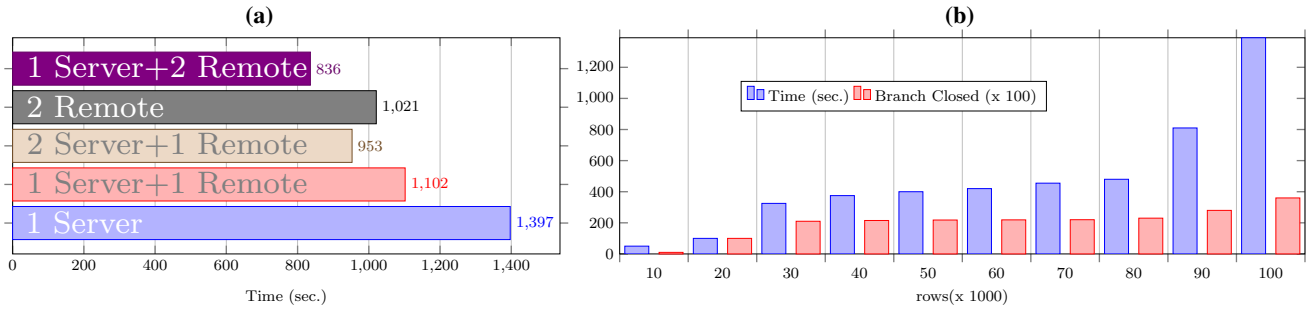


Fig. 25 **a** Result of *Attila*execution varying Sub-Contributors configurations on the same instance of the schema *Reports* consisting of $\sim 1.5 * 10^5$ rows (Server: 6 Core AMD Opteron(TM) 4284 3GHz, 8GB, Remote: AMD Phenom(TM) II X6 1055T Processor 2.8 GHz, 8GB). **b** Number of closed branches considering incremental portions of the same instance of schema *Reports* (the time refers to the execution on a Intel Core i3(TM) CPU M 330 2.13GHz, 4GB machine)

As expected, we observed that performances improved when distributing the task among different machines.

Figure 25b depicts the number of closed branches, which increases according to the size of the instance. These results confirm that our database instance is easier to evaluate than the artificial instances we built to prove NP-hardness results.

Conclusions

In this paper, we proposed a framework for discovering *Approximate Pure Temporally Evolving Functional Dependencies* (*APE*-FDs for short) from a temporal database. We have addressed in depth the data complexity of such problem. Unfortunately, this complexity turns out to be NP-Complete even for a single dependency. Moreover, moving to mining the set of *APE*-FDs holding on an instance **r**, the size of the result set depends also on the number of attributes of the schema of **r**. For some instances, the lower bound of such size is exponential. We faced these problems in a real-world context, by proposing the use of model checking techniques, distributed computations and linear programming techniques. The implemented prototype *Attila* was tested on two real-world clinical scenarios and proved to be efficient. Moreover, we discussed the meaning of some interesting *APE*-FDs mined from the databases in the psychiatry and pharmacovigilance domains, previously introduced. These results may provide (clinical) stakeholders with some new, previously unknown, understanding of the underlying data. We plan to further improve and extend our prototype, by integrating it in a platform allowing the discovery of different types of (temporal) approximate functional dependencies. Finally, we plan to perform an extended validation of mined *APE*-FDs with clinical experts.

Compliance with Ethical Standards

Conflict of interest On behalf of all authors, the corresponding author states that there are no conflict of interest.

Ethical Approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Bryant RE. Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput. 1986;35(8):677–91. https://doi.org/10.1109/TC.1986.1676819.
2. Codd EF. Normalized data base structure: a brief tutorial. In: Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) workshop on data description, access and control. ACM; 1971, pp. 1–17.
3. Combi C, Keravnou-Papailiou E, Shahar Y. Temporal information systems in medicine. Berlin: Springer; 2010.
4. Combi C, Mantovani M, Sabaini A, Sala P, Amaddeo F, Moretti U, Pozzi G. Mining approximate temporal functional dependencies with pure temporal grouping in clinical databases. Comput Biol Med. 2014;62:306–24.
5. Combi C, Mantovani M, Sala P. Discovering quantitative temporal functional dependencies on clinical data. In: 2017 IEEE international conference on healthcare informatics, ICHI 2017, Park City, UT, USA, Aug 23–26, 2017. IEEE Computer Society; 2017. pp. 248–257. https://doi.org/10.1109/ICHI.2017.80.
6. Combi C, Montanari A, Pozzi G. The T4SQL temporal query language. In: Silva MJ, Laender AHF, Baeza-Yates RA, McGuinness DL, Olstad B, Olsen ØH, Falcão AO, editors. CIKM. ACM; 2007. pp. 193–202.
7. Combi C, Montanari A, Sala P. A uniform framework for temporal functional dependencies with multiple granularities. In: Advances in spatial and temporal databases. Springer; 2011. pp. 404–421.
8. Combi C, Oliboni B, Pozzi G. Modeling and querying temporal semistructured data. In: New trends in data warehousing and data analysis. Springer; 2009. pp. 1–25.
9. Combi C, Pozzi G, Rossato R. Querying temporal clinical databases on granular trends. J Biomed Inform. 2012;45(2):273–91.
10. Combi C, Rizzi R, Sala P. The price of evolution in temporal databases. In: Grandi F, Lange M, Lomuscio A, editors, 22nd international symposium on temporal representation and reasoning, TIME 2015, Kassel, Germany, Sept 23–25, 2015. IEEE Computer Society; 2015. pp. 47–58. https://doi.org/10.1109/TIME.2015.24.
11. Combi C, Sala P. Mining approximate interval-based temporal dependencies. Acta Inf. 2016;53(6–8):547–85. https://doi.org/10.1007/s00236-015-0246-x.
12. Dvorský M. Common permutation problem. CoRR; 2008. arXiv:abs/0803.4261.
13. Huhtala Y, Karkkainen J, Porkka P, Toivonen H. Efficient discovery of functional and approximate dependencies using partitions. In: Proceedings of the 14th international conference on data engineering, 1998. IEEE; 1998. pp. 392–401.
14. Huhtala Y, Kärkkäinen J, Porkka P, Toivonen H. TANE: an efficient algorithm for discovering functional and approximate dependencies. Comput J. 1999;42(2):100–11.
15. Jensen CS, Snodgrass RT, Soo MD. Extending existing dependency theory to temporal databases. IEEE Trans Knowl Data Eng. 1996;8(4):563–82.
16. Kivinen J, Mannila H. Approximate inference of functional dependencies from relations. Theor Comput Sci. 1995;149(1):129–49.
17. Lind-Nielsen J. BuDDy—a binary decision diagram package. http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.itu.dk/research/buddy/. Accessed 3 Mar 2020.
18. Liu C, Zhang K, Xiong H, Jiang G, Yang Q. Temporal skeletonization on sequential data: patterns, categorization, and visualization. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2014. pp. 1336–1345.
19. Lopes S, Petit JM, Lakhal L. Functional and approximate dependency mining: database and fca points of view. J. Exp. Theor. Artif. Intell. 2002;14(2–3):93–114.
20. MedDRA. Medical Dictionary for Regulatory Activities. https://www.meddra.org/. Accessed 23 Apr 2020.
21. Meyboom R, Lindquist M, Egberts A, Edwards I. Signal selection and follow-up in pharmacovigilance. Drug Saf. 2002;25(6):459–65.
22. Object Management Group: Business Process Model and Notation (BPMN), v2.0.2. http://www.omg.org/spec/BPMN/2.0.2/PDF/. Accessed 3 Mar 2020.
23. Olson DL, Lauhoff G. Descriptive data mining. Computational risk management. 2nd ed. Berlin: Springer; 2019. https://doi.org/10.1007/978-981-13-7181-3.
24. Pryor J, Chinneck JW. Faster integer-feasibility in mixed-integer linear programs by branching to force change. Comput Oper Res. 2011;38(8):1143–52. https://doi.org/10.1016/j.cor.2010.10.025.
25. Sacchi L, Larizza C, Combi C, Bellazzi R. Data mining with temporal abstractions: learning rules from time series. Data Min Knowl Discov. 2007;15(2):217–47. https://doi.org/10.1007/s10618-007-0077-7.
26. Sala P. Approximate interval-based temporal dependencies: the complexity landscape. In: 2014 21st international symposium on temporal representation and reasoning (TIME). IEEE; 2014. pp. 69–78.
27. Sala P, Combi C, Cuccato M, Galvani A, Sabaini A. A framework for mining evolution rules and its application to the clinical domain. In: Balakrishnan P, Srivatsava J, Fu W, Harabagiu SM, Wang F, editors. 2015 International conference on healthcare informatics, ICHI 2015, Dallas, TX, USA, Oct 21–23, 2015. IEEE Computer Society; 2015. pp. 293–302. https://doi.org/10.1109/ICHI.2015.42.
28. Sordo M, Ochoa G, Murphy SN. A PSO/ACO approach to knowledge discovery in a pharmacovigilance context. In: GECCO (Companion); 2009. pp. 2679–2684.
29. Vaisman AA, Zimányi E. Data warehouse systems—design and implementation. Data-centric systems and applications. Berlin: Springer; 2014. https://doi.org/10.1007/978-3-642-54655-6.
30. Vianu V. Dynamic functional dependencies and database aging. J ACM (JACM). 1987;34(1):28–59.
31. Wang XS, Bettini C, Brodsky A, Jajodia S. Logical design for temporal databases with multiple granularities. ACM Trans. Database Syst. (TODS). 1997;22(2):115–70.
32. Wijsen J. Temporal FDs on complex objects. ACM Trans Database Syst. 1999;24(1):127–76.
33. Wijsen J. Temporal dependencies. In: Liu L, Özsu MT, editors. Encyclopedia of database systems. Berlin: Springer; 2009. p. 2960–6. https://doi.org/10.1007/978-0-387-39940-9_396.
34. World Health Organization and WHO Collaborating Centre for International Drug Monitoring: The Importance of Pharmacovigilance. Safety monitoring of medicinal products. World Health Organization; 2002.