An algorithm for verifying Approximate Pure Evolving Functional Dependencies*

Pietro Sala

Department of Computer Science, University of Verona, Italy
pietro.sala@univr.it

Abstract. Fuctional dependencies (FDs) form the foundations of database theory since the beginning of time, given they may represent constraints such as "the employee role determines his monthly salary". If we consider temporal databases where each tuple is timestamped with a valid time (VT) attribute, finer constraints may be imposed on the evolution of the data such as "the employee role before a promotion and his role afterwards determine his new monthly salary". Such constraints are called *Pure Evolving Functional Dependencies* (PEFDs) according to the classification introduced in [3]. By adding approximation to such rules they may be used for extracting knowledge in place of imposing constraints. Then we may want to *discover* that "the employee role before a promotion and his role afterwards *generally* determine his new monthly salary". This kind of *approximate dependencies* are called *Approximate Pure Evolving Functional Dependencies* (APEFDs). Verifying whether or not a given APEFD holds over a database instance is an NP-Complete problem [4]. Despite the discouraging complexity, in this work we propose an algorithms that solve in an efficient way this problem by trying to divide-and-conquer and prune the search space as much as possible.

Keywords: Temporal Databases · Approximate Functional Dependencies · Data Mining.

1 Temporally Mixed Functional Dependencies

According to [3], in the following we study a family of temporal functional dependencies [2, 5, 6] i.e., the *Pure Evolving Functional Dependencies*, called PEFD. From now on our temporal functional dependencies will be given over 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 timestamped with natural numbers (i.e, $\mathcal{D}om(VT) = \mathbb{N}$). Let $J \subseteq U$ be a non-empty 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., $\overline{W} = \{\overline{A} : A \in W\}$). Given an instance $\mathbf{r}$ of R, we define an instance $\tau_J^{\mathbf{r}}$ of schema $R_{ev} = JW\overline{W}\{VT, \overline{VT}\}$ as follows:

* This work has not been previously published and it is an extension of the results published in [4].

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

Schema R_{ev} is called the *evolution schema* of R. We will denote by τ_J^R the view 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. $t_1[J] = t_2[J]$) and $t_1[VT] < t_2[VT]$. Moreover, such tuples are joined if 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 exists a tuple that holds at some point in between the valid times of such tuples). For application purposes, it is correct correct to consider in a evolution schema only those pair of consecutive tuples whose the difference between $\overline{VT}$ and VT respects 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 $\Delta_k(\tau_J^{\mathbf{r}}) = \sigma_{t[\overline{VT}]-t[VT] \leqslant k}(\tau_J^{\mathbf{r}})$ (notice that $\Delta_{+\infty}(\tau_J^{\mathbf{r}}) = \tau_J^{\mathbf{r}}$). $\Delta_k(\tau_J^{\mathbf{r}})$ 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 $\Delta(t)$.

A *Pure Temporally Evolving Functional Dependency* [3] over the temporal schema $R = U \cup \{VT\}$, PEFD 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 $X \neq \varnothing$ and $|Z| = 1$ (Z contains a single attribute). We say that an instance $\mathbf{r}$ of R fulfills a PEFD $[\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}$, written $\mathbf{r} \models [\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}$, if and only if for each pair of tuples $t, t' \in \Delta_k(\tau_J^{\mathbf{r}})$ we have $t[X] = t'[X] \wedge t[\overline{Y}] = t'[\overline{Y}] \rightarrow t[\overline{Z}] = t'[\overline{Z}]$.

Now add approximation to PEFD in a very standard way [7, 2]. First, we provide measurement G to deal with PEFD as follows:

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

By means of G we can define the relative *scaled measurement* g for *TM*-FD as follows:

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

Now we are ready to define the *Approximate Pure Temporally Evolving Functional Dependency* (APEFD for short). Formally, given a real number $0 \leqslant \epsilon \leqslant 1$, we say that an instance $\mathbf{r}$ of R satisfies the APEFD $[\Delta_k(\tau_J^R)]X\ \overline{Y} \xrightarrow{\epsilon} \overline{Z}$, written $\mathbf{r} \models [\Delta_k(\tau_J^R)]X\overline{Y} \xrightarrow{\epsilon} \overline{Z}$, if and only if $g([\Delta_k(\tau_J^R)]X\overline{Y} \rightarrow \overline{Z}, \mathbf{r}) \leqslant \epsilon$.

In Section 2 we provide an algorithm for checking an APEFD against an instance $\mathbf{r}$, we call this problem Check-APEFD:

Problem 1. (Check-APEFD). Given a temporal schema R, a PEFD $[\Delta_k(\tau_J^R)]$ $X\overline{Y} \rightarrow \overline{Z}$ on R, an instance $\mathbf{r}$ of R, and a real number $0 \leqslant \epsilon \leqslant 1$ determine whether or not $\mathbf{r} \models [\Delta_k(\tau_J^R)]X\overline{Y} \xrightarrow{\epsilon} \overline{Z}$.

$\mathbf{r} =$

#	*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

Fig. 1. 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*).

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

#	*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. 2. The evolution expression $\tau^{\mathbf{r}}_{Name}$.

Now we consider a scenario, borrowed from the clinical domain, in order to provide examples of how PEFDs and APEFDs work. In particular, this 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, he 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 being received, not the exact time the call has begun which is usually its semantics in real world applications. In addition, the service may be used by people somehow related to patients, as, for instance, relatives afraid of current conditions of a patient. More precisely, attribute $Name$ identifies patients, $Phys$ identifies physicians, CT (*Contact Type*) identifies the physical person who is doing the call (e.g., value 'self' stand for the patient himself, 'family' for a relative), and Dur stores information about total duration of calls (value $\sim n$ means approximately n minutes). An instance $\mathbf{r}$ of R is provided in Figure 1. 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 Figure 2. In the following we will use t for referencing tuples of $\mathbf{r}$ 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 $\mathbf{r}$ that generate u. For instance, the first tuple of $\tau^{\mathbf{r}}_{J}$ in Figure 2 will be denoted by $u_{1,2}$ since it is generated by the join of tuples t_1 and t_2 in $\mathbf{r}$.

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 3 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 $\tau^{\mathbf{r}}_{Name}$. In both scenarios depicted in Figure 3, 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). Basically, each tuple $u \in \tau^{\mathbf{r}}_{Name}$ corresponds to an edge in Figure 3 while we have a node for each tuple in $\mathbf{r}$.

In our example, we have that $\mathbf{r} \models [\Delta_5(\tau^{Contact}_{Name})]Phys, \overline{Phys} \rightarrow \overline{CT}$. However, $\mathbf{r} \not\models [\Delta_6(\tau^{Contact}_{Name})]Phys, \overline{Phys} \rightarrow \overline{CT}$, because of pairs (t_2, t_3) and (t_{10}, t_{11}). More precisely, we have that $t_2[Name] = t_3[Name] = McMurphy$, $t_{10}[Name] = t_{11}[Name] = Lowe$, $t_2[Phys] = t_{10}[Phys] = Sayer$, $t_3[Phys] = t_{11}[Phys] = Maguire$, but $t_3[CT] \neq t_{11}[CT]$ (i.e., $t_3[CT] = family$, and $t_{11}[CT] = self$).

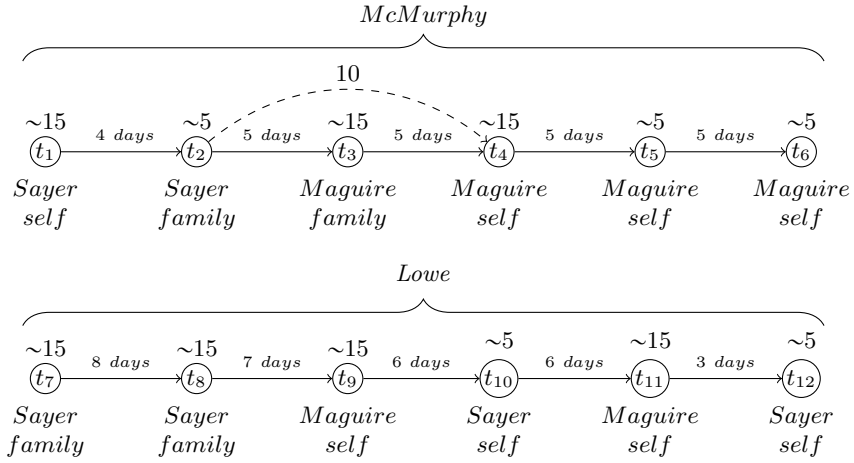


Fig. 3. A graph-based representation of $\tau^{\mathbf{r}}_{Name}$.

In other words, we have that the set of tuples $\{u_{2,3}, u_{10,11}\}$ does not satisfy the FD $Phys\overline{Phys} \rightarrow \overline{CT}$.

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

In our example, if we consider APEFD $[\Delta_6(\tau^R_{Name})]Phys, \overline{Phys} \xrightarrow{\epsilon} \overline{CT}$ with $\epsilon = \frac{1}{12}$, we have that $\mathbf{r} \models [\Delta_6(\tau^R_{Name})]\ Phys, \overline{Phys} \xrightarrow{\epsilon} \overline{CT}$. It is worth noting that, by considering relation $\mathbf{r'} = \mathbf{r}\backslash\{t_3\}$, 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 $\tau^{\mathbf{r'}}_{Name} = \tau^{\mathbf{r}}_{Name}\backslash\{u_{2,3}, u_{3,4}\} \cup \{u_{2,4}\}$. Notice that tuple $u_{2,4}$ was not originally in $\tau^{\mathbf{r}}_{Name}$ because of tuple t_3. Figure 3 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 that $\mathbf{r'} \models [\Delta_{+\infty}(\tau^R_{Name})]Phys, \overline{Phys} \rightarrow \overline{CT}$. Thus, $\mathbf{r} \models [\Delta_{+\infty}(\tau^R_{Name})]Phys, \overline{Phys} \xrightarrow{\epsilon} \overline{CT}$ with $\epsilon = \frac{1}{12}$.

2 An algorithm for checking APEFDs

As we proved in [4], the problem Check-APEFD given an APEFD $[\tau^R_J, sw(k)]X\overline{Y} \xrightarrow{\epsilon} \overline{Z}$ and an instance $\mathbf{r}$ of R is NP-Complete in $|\mathbf{r}|$. This is not an uncommon situation in the realm of temporal functional dependencies (see [8, 1, 9, 7] for further examples).

Then, in principle, there is no asimptoptycally better algorithm than explore the whole set of of possible subsets $\mathbf{r}'$ of $\mathbf{r}$ with $\frac{|\mathbf{r}|-|\mathbf{r}'|}{|\mathbf{r}|} \leqslant \epsilon$. However, in the following, we provide an algorithm that make use of heuristics for pruning the search space in order to achieve the tractability for many cases.

Such algorithm is general and it may be applied under no assumptions on the input instance $\mathbf{r}$, it makes use of two optimization techniques. The first optimization technique consists of trying, whenever is possible, to split the current subset of $\mathbf{r}$ in two subsets on which the problem may be solved independently (i.e., choices in one subset do not affect choices in the other one and viceversa). The latter optimization technique consists of checking if 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 happen the subtree is pruned immediately (i.e., we are looking only for optimal solutions).

Our algorithm relies on the concept of *color* that we will explain through an example in the following. Given APEFD $[\tau_J^R, sw(k)]X\overline{Y} \xrightarrow{\epsilon} \overline{Z}$ and an instance $\mathbf{r}$ of R, let us suppose that we are solving problem Check-APEFD on such instance with a simple guess-and-check procedure that make use of two, initially empty, subset $\mathbf{r}^+$ (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 guess a tuple t in $\mathbf{r}\backslash(\mathbf{r}^+ \cup \mathbf{r}^-)$ and decides non-deterministically (*guessimg phase*) either to update $\mathbf{r}^+$ to $\mathbf{r}^+ \cup \{t\}$ (i.e., t is kept in the current partial solution) to update $\mathbf{r}^-$ to $\mathbf{r}^- \cup \{t\}$ (i.e, t is deleted in the current partial solution). When $\mathbf{r} = \mathbf{r}^+ \cup \mathbf{r}^-$ (*check phase*), the procedure returns YES if $\mathbf{r}^+ \models [\tau_J^R, sw(k)]X\overline{Y} \rightarrow \overline{Z}$ and $|\mathbf{r}^-| \leqslant \epsilon \cdot |\mathbf{r}|$, otherwise it returns NO. From now on, for us a *partial solution* is a triple $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ such that $(\mathbf{r}^+ \cup \mathbf{r}^-) \subseteq \mathbf{r}$ and $\mathbf{r}^+ \cap \mathbf{r}^- = \varnothing$, if $\mathbf{r} = \mathbf{r}^+ \cup \mathbf{r}^-$ we simply say that $(\mathbf{r}^+ \cup \mathbf{r}^-)$ is a *solution*. A solution is *consistent* $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ if and only if $\mathbf{r}^+ \models [\tau_J^R, sw(k)]X\overline{Y} \xrightarrow{\epsilon} \overline{Z}$. Given two partial solution $(\mathbf{r}, \mathbf{r}_1^+, \mathbf{r}_1^-)$ 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 $\mathbf{r}_1^- \subseteq \mathbf{r}_2^-$.

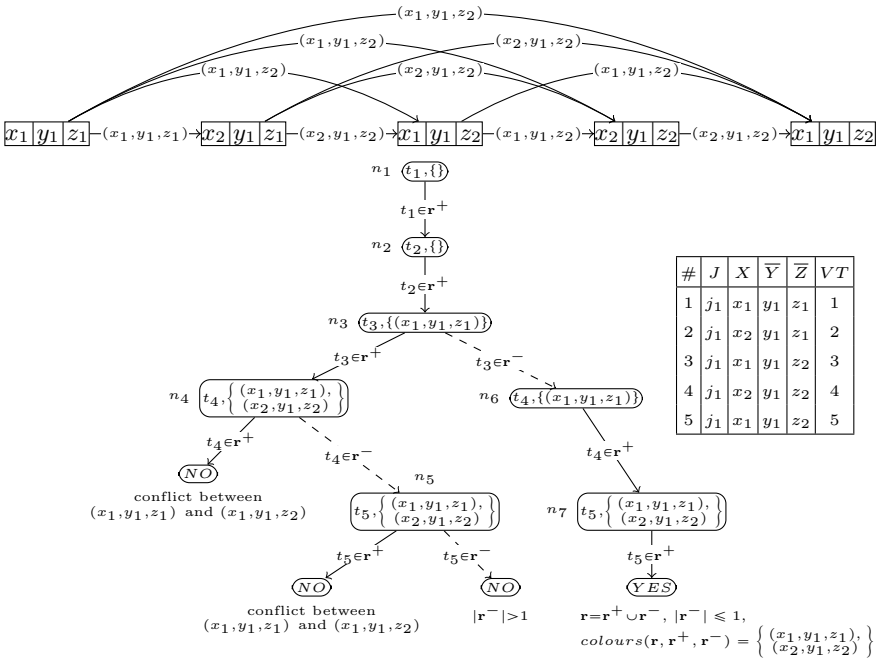
Is there a way to check if we are generating an inconsistent solution possibly without guessing all the tuples in $\mathbf{r}$? Violations of the latter constraint (i.e., $|\mathbf{r}^-| \leqslant \epsilon \cdot |\mathbf{r}|$) are fairly simple to detect during the guessing phase, it suffices to check 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., $\mathbf{r}^+ \models [\tau_J^R, sw(k)]X\overline{Y} \rightarrow \overline{Z}$) during the guessing phase are trickier to detect, then we need the following additional definitions. From now on when two tuples t, t' share the same value for the attribute J (i.e., $t[J] = t'[J]$) we will say that they are in the same J-group and we will say that $t[J]$ 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 $t[VT] < t'[VT]$. Given an o-pair $t, t' \in \mathbf{r}$ we say that the pair (t, t') is an *edge* if and only if $0 < t'[VT] - t[VT] \leqslant k$. Given triple $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$, an o-pair $(t, t') \in \mathbf{r}$ 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 $t[VT] < \bar{t} < t'[VT]$ 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 are deleted). Given two valid times $vt, vt' \in Dom(VT)$ and a value $j \in Dom(J)$ we say that vt and vt' are consecutive in j-group if and only if there exists an active o-pair (t, t') with $t[VT] = vt$, $t'[VT] = vt'$, and $t[J] = j$. Notice 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 $C = XYZ$, 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'), 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. The following result it is easy to prove and its proof is left as exercise for the reader.

Theorem 1. *Given an* APEFD $[\tau_J^R, sw(k)]X\overline{Y} \xrightarrow{\epsilon} \overline{Z}$*, an instance* $\mathbf{r}$ *of* R*, and a partial solution* $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$*, if there exists two active edges* (t, t') *and* (t'', t''') *then every solution* $(\mathbf{r}, \mathbf{r}^+{}_f, \overline{\mathbf{r}^-}_f)$ *that follows* $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ *satisfy* $\mathbf{r}_f^+ \not\models [\tau_J^R, sw(k)]X\overline{Y} \rightarrow \overline{Z}$ *(i.e., is inconsistent).*

The above theorem guarantees that from a partial solution $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ that feature at least two conflicting edges we cannot reach a solution $(\mathbf{r}, \mathbf{r}^+{}_f, \overline{\mathbf{r}^-}_f)$ that satisfy the first constraint and in such a case we may return immediately NO without going any further from $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$. The colours of a partial solution $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ is the set $colours(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-) = \{(t[X], t'[Y], t'[Z]) : (t, t')$ is an active edge in $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)\}$. It is easy to see that the hypothesis of Theorem 1 applies if and only if $colours(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ contains at least two conflicting colours. Then, by means of colours, 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 colours $colours(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$. Once an insertion of a tuple in either $\mathbf{r}^+$ or $\mathbf{r}^-$ introduce a color c that is conflicting with at least one colour in $(\mathbf{r}, \mathbf{r}^+, \mathbf{r}^-)$ the procedure answers NO immediately.

An example of how the procedure works is given in Figure 4 where we have an instance of 5 tuples with $\epsilon = 0.2$ (i.e., we may delete at most one tuple) and $sw(k) = 6$ (all the tuples are in the same window). The execution depicted in Figure 4 guesses the values of the 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 the 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 labelled with the current tuple which will be guessed next, every leaf is labelled either with YES (i.e., the current branch is a solution) or NO (i.e, a violation has arisen), the current set of colours 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 the nodes $n_1 \ldots n_4$ 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



#	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. 4. An example of how the use of colors improve a guess and check procedure for solving the problem Check-APEFD.

n_5 cannot be extended without introducing a violation in the above constraints because if put t_5 in $\mathbf{r}^+$ we introduce a conflicting color, while if we put t_5 in $\mathbf{r}^-$ we exceed the maximum number of allowed deletions. We backtrack to n_4 and we have that even for it all the possible choices have been explored and thus we backtrack to n_3 where the choice of adding t_3 to $\mathbf{r}^-$ is attempted generating the node n_6. From n_6 we put t_5 in $\mathbf{r}^+$ without violating any constraint and thus we have that $\{t_1,t_2,t_4,t_5\} \models [\tau_J^R,6]X\overline{Y} \overset{0.2}{\rightarrow} \overline{Z}$.

Let us consider deeper in the description of the first algorithm. On an higher lever the algorithm operates like the procedure above apart from some trivial technicalities. However, two more heuristics are introduced in order to possibly stop early during the exploration of a branch in the tree of computation. The main procedure of the algorithm is reported in Figure 7 with auxiliary procedures reported in Figure 5 and in Figure 6. The algorithm is implemented by the function $TupleWiseMin$ that takes 4 arguments. The first argument is $G_{\mathbf{r}}$ which is a pre-processing of $\mathbf{r}$ based on the APEFD $[\tau_J^R, sw(k)]X\overline{Y} \overset{\epsilon}{\rightarrow} \overline{Z}$ that has to be checked. More precisely, $G_{\mathbf{r}}$ is an instance of the 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

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 $\left\{(t_1, t_2) : \begin{array}{l} t_1[J] = t_2[J] \wedge 0 < t_2[VT] - t_1[VT] \leqslant k \wedge \{t_1, t_2\} \cap G^-_{\mathbf{r}} = \varnothing \wedge \\ \wedge \neg ColorConflict?((t_1, t_2), \mathcal{C}) \wedge \\ \wedge InBetween(G_{\mathbf{r}}, t_1, t_2) \cap G^+_{\mathbf{r}} = \varnothing \wedge \\ \wedge (\{t_1, t_2\} \cup InBetween(G_{\mathbf{r}}, t_1, t_2)) \nsubseteq (G^+_{\mathbf{r}} \cup G^-_{\mathbf{r}}) \end{array}\right\}$

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 wether or not there exits a path between two nodes in a graph.
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

Fig. 5. Auxiliary procedures used by procedures presented in Figures 6 and 7.

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 the others 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 MACCONSISTENTSUBSET(E)
comment: returns the maximal subset E' of E such that d 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** $\begin{cases} \textbf{if } \forall \bar{t}_1 \forall \bar{t}_2((\bar{t}_1, \bar{t}_2) \in E \rightarrow \neg EdgeConflict?((t_1, t_2), (\bar{t}_1, \bar{t}_2))) \\ \quad \textbf{then } E' \leftarrow E' \cup \{(t_1, t_2)\} \end{cases}$
return E'

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 $\begin{cases} \textbf{for each } (\bar{t}_1, \bar{t}_2) \in (E?(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}, \mathcal{C}) \cup E!(G_{\mathbf{r}}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}}) \cup \tilde{E}) \\ \quad \textbf{do } \begin{cases} \textbf{if } EdgeConflict?((t_1, t_2), (\bar{t}_1, \bar{t}_2)) \\ \quad \textbf{then } NotSafe \leftarrow NotSafe \cup \{(t_1, t_2)\} \end{cases} \end{cases}$
$\tilde{E} \leftarrow \tilde{E} \backslash NotSafe$
if $\exists t_1 \exists t_2 \forall \bar{t}_1 \forall \bar{t}_2 \left(\begin{array}{c} t_1, t_2 \in N_m \wedge \bar{t}_1 \in N_s \wedge \bar{t}_2 \in N_e \wedge \\ \wedge (\bar{t}_1, t_1), (t_2, \bar{t}_2) \in \tilde{E} \wedge Reach?(t_1, t_2, N, \tilde{E}) \end{array} \right)$
then return true
return false

Fig. 6. Auxiliary procedures used by procedure $TupleWiseMin$ (Figure 7).

Algorithm 2.1: TUPLEWISEMIN($G_{\mathbf{r}}, k, G^+_{\mathbf{r}} = \varnothing, G^-_{\mathbf{r}} = \varnothing, \mathcal{C} = \varnothing$)

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 $G_{\mathbf{r}} = \varnothing$ **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 $\begin{cases} \mathcal{C}' \leftarrow \mathcal{C} \cup \{(t'[X], t''[Y], t''[Z]) : (t', t'') \in E!(G_{\mathbf{r}} \backslash \{t\}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}} \cup \{t\})\} \\ m_{\mathfrak{t}} \leftarrow TupleWiseCheck(G_{\mathbf{r}} \backslash \{t\}, k, G^+_{\mathbf{r}}, G^-_{\mathbf{r}} \cup \{t\}, \mathcal{C}') \end{cases}$
else $m_{\mathfrak{t}} \leftarrow +\infty$
return $\min(m_t, m_{\mathfrak{t}})$

Fig. 7. The main procedure for checking APEFDs in a tuple-wise fashion. Notice that we use a compact notation for the recursive procedure which is initially called as $TupleWiseMin(G_{\mathbf{r}}, k)$ where when $G^+_{\mathbf{r}}$, $G^-_{\mathbf{r}}$, and $\mathcal{C}$ are ommitted in the procedure call they get their respective default values specified in the procedure declaration (i.e., $\varnothing$ for each of them in this case).

many tuples in $\mathbf{r}$ share the same values for attributes J, X, Y, and Z. The input parameter k is the length for the sliding window grouping $sw(k)$. The sets $G^+_{\mathbf{r}}$ and $G^-_{\mathbf{r}}$, originally initialized to $\varnothing$, represent the tuples of $G_{\mathbf{r}}$ that are either kept or deleted in the current solution respectively. On instances r of schema R satisfying $count \in R$ we denote with $||r||$ the sum on the *count* attribute for the tuples in r (i.e., $||r|| = \sum_{t \in r} t[count]$). Finally, $\mathcal{C}$ is a set of *colors* which is initially set to $\varnothing$. A color c is a tuple on the schema X, Y, Z. As we will see, $\mathcal{C}$ keeps track via colours of the constraints introduced so far in the construction of the solution.

The 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 $\mathbf{r}' \models [\tau^R_J, sw(k)]X\overline{Y} \rightarrow \overline{Z}$. Then if such minimum is less or equal than $\epsilon \cdot |\mathbf{r}|$ we can conclude $\mathbf{r} \models [\tau^R_J, sw(k)]X\overline{Y} \xrightarrow{\epsilon} \overline{Z}$ else we have $\mathbf{r} \not\models [\tau^R_J, sw(k)]X\overline{Y} \xrightarrow{\epsilon} \overline{Z}$. Given $G_{\mathbf{r}}$, $G^+_{\mathbf{r}}$, $G^-_{\mathbf{r}}$, and a set of colours $\mathcal{C}$ 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 $t[VT] \leqslant t''[VT] \leqslant t'[VT]$.

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 the procedure $E?$ while active edges are retrieved by the procedure $E!$.

The procedure $TupleWiseMin$ (Figure 7) 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 proceed as follows. First, it checks if inserting t into $G^+_{\mathbf{r}}$ does not cause any violation in the 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. Notice that 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 if inserting t into $G^-_{\mathbf{r}}$ does not cause any violation in the constraints, if so it stores in $m_{\not t}$ 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 $m_{\not t}$ 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 $(t'[X], t''[\overline{Y}], t''[\overline{Z}])$ turns out to be conflicting with at least one color in $\mathcal{C}$.

The first one allows us to restrict the search space by splitting the problem into independent sub-problems 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 j-group or all $t, t', \bar{t}$, and $\bar{t}'$ do not belong to j-group (such condition is verified by sub-procedure *GroupIndependent*? reported in Figure 6.). Let $\overline{G}_{\mathbf{r}} = \{t : t[J] = j\}$, if every edge involving tuples in j-group is not conflicting with every edge that may be introduced outside j-group then we can split the problem into the two sub-problems $(\overline{G}_{\mathbf{r}}, k, G^+_{\mathbf{r}} \cap \overline{G}_{\mathbf{r}}, G^-_{\mathbf{r}} \cap \overline{G}_{\mathbf{r}})$ 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 resolved in a separate way. The resulting value for the solution is the sum of the values returned by $TupleWiseMin$ applied to both the two sub-problems. Let $H = |G^-_{\mathbf{r}} \backslash (G^+_{\mathbf{r}} \cup G^-_{\mathbf{r}})|$ and $h = |\{t \in (G^-_{\mathbf{r}} \backslash (G^+_{\mathbf{r}} \cup G^-_{\mathbf{r}})) : t[J] = j\}|$, notice that, 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 sub-tree of computation even before a contradiction arises. The second optimization implements a criteria to detect, in many cases, if 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 denote with $colors(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$ the 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 $\bar{t}, \bar{t}' \in (G^+_{\mathbf{r}} \cup G^-_{\mathbf{r}}) \cap \{\bar{t}'' \in G_{\mathbf{r}} : \bar{t}''[J] = j \wedge t[VT] \leqslant \bar{t}'' \leqslant t'[VT]\}$ 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)

Notice that the above three conditions imply that if a color is safe in (vt, vt', j) then it is neither in conflict with a color $colors(G_{\mathbf{r}}, G^+_{\mathbf{r}}, G^-_{\mathbf{r}})$ 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. 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

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

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* is (vt, vt', j)-replace DAG (V, E) is any path $t_1 \ldots t_m$ in (V, E) for whicht 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).

Using the above definitions of replace DAGs/paths we can provide the following result.

Theorem 2. *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 very simple and left as exercise to the reader. Let us suppose that $t_1 \ldots t_m$ is a replace path in the (vt, vt', j)-replace DAG, it must exists by hypothesis and 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 that $(G_{\mathbf{r}}, \overline{G}^+_{\mathbf{r}} \cup \{t_1, \ldots, t_m\}, \overline{G}^-_{\mathbf{r}} \backslash \{t_1, \ldots, t_m\})$ is still a consistent solution, non-optimality immediately follows. We take advantage of Theorem 2 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 j-group such that vt and vt' can be safely replaced in j. Notice that verify wether or not such condition applies may be performed in polynomial time. In the procedure $TupleWiseMin$, this optimization is realized by the joint work of sub-procedures $MaximalPaths$? and $ReplacePath$? reported in Figure 7 and 6 respectively.

3 Conclusion

In this work we propose an algorithm for solving the problem of verifying whether or not a given APEFD $\mathcal{EF}$ holds over a given instance $\mathbf{r}$ of a temporal schema. Such problem is known to be NP-Complete in the size of $\mathbf{r}$ (i.e., data-complexity). We expect our algrithm to perform considerably better w.r.t. standard branch and bound procedures since it considerably prune the search space in most cases. We are building a prototype in order to measure the effective improvement w.r.t. frameworks that solve generic NP-Complete problems (i.e., Integer Linear Programmming tools, SAT solvers, and logic programming), obviously this implies that we have to encode our problem in such formalisms.

References

1. 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. Comp. in Bio. and Med. **62**, 306–324 (2015). https://doi.org/10.1016/j.compbiomed.2014.08.004, https://doi.org/10.1016/j.compbiomed.2014.08.004
2. 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, August 23-26, 2017. pp. 248–257. IEEE (2017). https://doi.org/10.1109/ICHI.2017.80, https://doi.org/10.1109/ICHI.2017.80
3. Combi, C., Montanari, A., Sala, P.: A uniform framework for temporal functional dependencies with multiple granularities. In: Pfoser, D., Tao, Y., Mouratidis, K., Nascimento, M.A., Mokbel, M.F., Shekhar, S., Huang, Y. (eds.) Advances in Spatial and Temporal Databases - 12th International Symposium, SSTD 2011, Minneapolis, MN, USA, August 24-26, 2011, Proceedings. Lecture Notes in Computer Science, vol. 6849, pp. 404–421. Springer (2011). https://doi.org/10.1007/978-3-642-22922-0_24, https://doi.org/10.1007/978-3-642-22922-0$_$24
4. Combi, C., Rizzi, R., Sala, P.: The price of evolution in temporal databases. In: Grandi, F., Lange, M., Lomuscio, A. (eds.) 22nd International Symposium on Temporal Representation and Reasoning, TIME 2015, Kassel, Germany, September 23-25, 2015. pp. 47–58. IEEE Computer Society (2015). https://doi.org/10.1109/TIME.2015.24, https://doi.org/10.1109/TIME.2015.24
5. Combi, C., Sala, P.: Temporal functional dependencies based on interval relations. In: Combi, C., Leucker, M., Wolter, F. (eds.) Eighteenth International Symposium on Temporal Representation and Reasoning, TIME 2011, Lübeck , Germany, September 12-14, 2011. pp. 23–30. IEEE (2011). https://doi.org/10.1109/TIME.2011.15, https://doi.org/10.1109/TIME.2011.15
6. Combi, C., Sala, P.: Interval-based temporal functional dependencies: specification and verification. Ann. Math. Artif. Intell. **71**(1-3), 85–130 (2014). https://doi.org/10.1007/s10472-013-9387-1, https://doi.org/10.1007/s10472-013-9387-1
7. Combi, C., Sala, P.: Mining approximate interval-based temporal dependencies. Acta Inf. **53**(6-8), 547–585 (2016). https://doi.org/10.1007/s00236-015-0246-x, https://doi.org/10.1007/s00236-015-0246-x
8. Sala, P.: Approximate interval-based temporal dependencies: The complexity landscape. In: Cesta, A., Combi, C., Laroussinie, F. (eds.) 21st International Symposium on Temporal Representation and Reasoning, TIME 2014, Verona, Italy, September 8-10, 2014. pp. 69–78. IEEE Computer Society (2014). https://doi.org/10.1109/TIME.2014.20, https://doi.org/10.1109/TIME.2014.20
9. 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, S.M., Wang, F. (eds.) 2015 International Conference on Healthcare Informatics, ICHI 2015, Dallas, TX, USA, October 21-23, 2015. pp. 293–302. IEEE Computer Society (2015). https://doi.org/10.1109/ICHI.2015.42, https://doi.org/10.1109/ICHI.2015.42