The VLDB Journal (2021) 30:425–446
https://doi.org/10.1007/s00778-020-00641-6

REGULAR PAPER

Cleaning timestamps with temporal constraints

Shaoxu Song[1] · Ruihong Huang[1] · Yue Cao[1] · Jianmin Wang[1]

Received: 14 September 2019 / Revised: 7 August 2020 / Accepted: 16 September 2020 / Published online: 23 February 2021

Abstract
Timestamps are often found to be dirty in various scenarios, e.g., in distributed systems with clock synchronization problems or unreliable RFID readers. Without cleaning the imprecise timestamps, temporal-related applications such as provenance analysis or pattern queries are not reliable. To evaluate the correctness of timestamps, *temporal constraints* could be employed, which declare the distance restrictions between timestamps. Guided by such constraints on timestamps, in this paper, we study a novel problem of repairing inconsistent timestamps that do not conform to the required temporal constraints. Following the same line of data repairing, the timestamp repairing problem is to minimally modify the timestamps towards satisfaction of temporal constraints. This problem is practically challenging, given the huge space of possible timestamps. We tackle the problem by identifying a concise set of promising candidates, where an optimal repair solution can always be found. Repair algorithms with efficient pruning are then devised over the identified candidates. Approximate solutions are also presented including simple heuristic and linear programming (LP) relaxation. Experiments on real datasets demonstrate the superiority of our proposal compared to the state-of-the-art approaches.

Keywords Data cleaning · Timestamp repairing · Temporal constraints

1 Introduction

Imprecise timestamps are very prevalent, e.g., owing to clock synchronization, granularity mismatch, latency or out-of-order delivery of events in distributed systems [7]. Cleaning the imprecise timestamps is necessary for reliable applications, such as provenance analysis [25] identifying the sequence of steps leading to a data value, complex event processing (CEP) [15] returning the occurrences of requested event patterns, non-on-time events query [16], etc.

Figure 1 presents some example segments of real event logs in the ERP system of a train manufacturer (see Sect. 8 of experiments for more information). A *trace*, a.k.a. workflow run or process instance, is a collection of events. For instance, trace σ_1 in Fig. 1 records five steps (events) for processing a part design work, including Submit, Normalize, Proofread, etc. Each event is associated with a timestamp on when this event occurred. Every part design process yields a trace, e.g., σ_2 in Fig. 1 is the trace of designing another part.

✉ Shaoxu Song
sxsong@tsinghua.edu.cn
https://sxsong.github.io

[1] Tsinghua University, Beijing, China

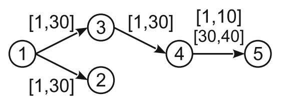
Since the events are collected from various external sources, imprecise timestamps are prevalent, e.g., 23:53 of event 2 in σ_1, which is delayed until just before midnight owing to latency. Another example of granularity mismatching appears in event 3 in σ_2. Proofread of the part is processed by an outsourcing company, which records timestamps in hour granularity, i.e., 14:__. However, the events are obviously not occurring in random, but constrained by certain workflow discipline.

1.1 Temporal constraints

Constraints are essential in evaluating the correctness of data, such as integrity constraints for relational data [18]. Regarding temporal data, we employ *temporal networks* [14], specifying temporal constraints on timestamp differences between nodes/variables. The aforesaid imprecise timestamps could be identified as violations of the temporal constraints.

Figure 1 illustrates a *temporal network* (abstracted from workflow specifications), specifying constraints on occurring timestamps of events (denoted by nodes). For instance, the temporal constraint [1, 30] from events 1 to 3 in Fig. 1 indicates the minimum and maximum restrictions on the dis-

	1 (Submit)	2 (Normalize)	3 (Proofread)	4 (Examine)	5 (Authorize)
σ_1	09:05	**23:53**	09:25	09:48	09:54
σ_2	14:46	14:55	**14:__**	15:22	15:59



Temporal constraint network

Fig. 1 Temporal constraint network

tance (delay) of these two events' timestamps. That is, event 3 (Proofread) should be processed within 30 minutes after event 1 (Submit). Events 3 and 1 in σ_1 satisfy this temporal constraint, since their timestamp distance 09:25−09:05 = 20 is in the range of [1, 30].

Multiple intervals may also be declared between two events. For instance, in Fig. 1, [1,10][30,40] on edge $4 \rightarrow 5$ denote that 5 (Authorize) can be processed after 4 (Examine) either by the department head in 1-10 minutes or by the division head in 30-40 minutes.

The aforesaid imprecise timestamps are then identified as violations of the temporal constraints, such as events 2 and 1 in σ_1 with timestamp distance 23:53 − 09:05 = 888 > 30. Similarly, events 3 and 1 in σ_2 with distance 14:__ − 14:46 = −46 are identified as inconsistent timestamps as well.

The temporal constraints can be obtained in various ways. (1) The rules could be specified by experts with domain knowledge (as in the example in Fig. 1). For instance, workflow/process specifications may specify the maximum/minimum delay of an event after another [7]. (2) They may also be discovered from data, as in the experiments in Sect. 8. The FBLG approach [10] discovers the temporal dependency between different time series data. For instance, in Fig. 1, it may discover that Normalize at time t is caused by Submit at time $t - k$, known as a lag k. Such a lag k could be within an interval, e.g., [1, 30], namely lag interval [26]. While finding the interesting temporal constraints is out of the scope of this study, we directly employ the existing discovery method and focus on repairing the timestamp errors under the (discovered) constraints.

1.2 Timestamp repairing

Inspired by data repairing w.r.t. integrity constraints [9], the timestamp repairing proposes to modify the timestamps towards satisfaction of the given temporal constraints. To address the imprecise timestamps, similar to other constraint-based repairing [13], the temporal constraint network is given in advance as inputs. Although only five events are presented in Fig. 1, a trace could be longer, containing events that are not specified in the constraints. The events are not necessary to be ordered by timestamps, since their timestamps are imprecise.

The challenge of repairing the inconsistent timestamps is obvious, given the huge volume of possible timestamps. To capture reasonable repairs, we follow the *minimum change principle* in data repairing [9], i.e., to find a repair that is as close as possible to the original observation. The rationale behind is that systems or human always try to minimize their mistakes in practice. In the meantime, the value change is expected to be minimized to preserve the original information as much as possible [19]. (See a detailed discussion in Sect. 2.2.)

The *timestamp repairing problem* is thus, given an assignment of timestamps violating a temporal network, to find a repaired timestamp assignment that (1) satisfies the temporal constraints and (2) is closest to the initial assignment.

For instance, in Fig. 1, to eliminate the violation between events 2 and 1 in σ_1, one may modify the timestamp of event 1 (e.g., to 23:23, referring to the constraint [1, 30] on $1 \rightarrow 2$). However, it introduces new violations between events 1 and 3, and leads to further modifications on events 3, 4 and 5. Alternatively, we can repair the timestamp of event 2 by 09:35, which satisfies the time constraint and does not evoke further timestamp modification. It is true that the repaired timestamp (09:35 with the minimum change) may not be exactly the original true timestamp. Nevertheless, without further knowledge, repairing *slightly* a single event 2 is more reasonable than modifying *significantly* over almost the entire trace 1, 3, 4 and 5, under the discipline that people and systems always try to minimize mistakes in practice.

Preliminary studies [17,28] handle imprecise timestamps by an uncertainty model on possible timestamps. The probabilistic-based repairing is thus performed via probabilistic inference in Bayesian Networks [22]. A key issue of this method (as analyzed in Sect. 9 of related work and evaluated in Sect. 8 of experiments) is that its repairing heavily relies on an essential preliminary step of correctly ordering data points before adapting the timestamps.

Beside the probabilistic-based approach, one may model the temporal constraints as integrity constraints (e.g., denial constraints [12]), and employ the existing constraint-based data repairing methods [13]. Unfortunately, according to our analysis (in Sect. 9 as well as in the experimental evaluation), the soundness w.r.t. satisfaction of temporal constraints is not guaranteed due to the greedy computation.

1.3 Contributions

The preliminary version of this paper [23] focuses on pruning candidates that already have a larger repairing cost than the currently known solution, or have violations to others. In this paper, we further consider building index to cache the optimal solutions and enable sharing among different problems (Sect. 5.2). Moreover, we introduce another approximation method with linear programming relaxation and rounding by the solution transformation in Algorithm 1 (Sect. 6.2). Finally, we extend our algorithms to support incremental computation in a streaming setting (Sect. 7). Our main contributions are summarized as

1. We propose a solution transformation paradigm in Sect. 3, the key to identify a finite set of timestamp repair candidates. Our essential argument is that any repair solution (including the optimal one) can be transformed to a special form, without increasing modification cost, such that each changed node (in repairing) is tightly connected to some unchanged node. By tightly, we mean the timestamp difference of two nodes equals to the interval endpoint of temporal constraints. Intuitively, the tight relationship is important since it significantly reduces the number of timestamp candidates considered between two nodes.
2. We capture a finite set of timestamp repair candidates, w.r.t. the aforesaid unchanged nodes and tight connections, where an optimal repair solution can always be found (Corollary 5) in Sect. 4. To generate a more concise set of candidates, we show that it is sufficient to consider a special type of provenance chains, instead of arbitrary tight connections.
3. We devise an exact algorithm for repairing timestamps based on the captured candidates in Sect. 5. Unlike the constraint-based greedy repairing [13], the satisfaction of temporal constraints (soundness) is guaranteed in our results. Advanced *pruning* and *indexing* techniques are also developed in Sect. 5.
4. We present two approximate algorithms in Sect. 6. The simple heuristic approach is to greedily consider only one branch of the exact algorithm. Without the costly candidate generation, another linear programming relaxation is also devised by extending the solution transformation method for rounding. Moreover, we extend our algorithm to support incremental computation in a streaming setting in Sect. 7.
5. We report an extensive experimental evaluation over a real dataset in Sect. 8. The results demonstrate that our proposed methods show significantly higher repair accuracy compared to the state-of-the-art approaches, while time cost of our proposal is lower (see Sect. 8.6 for details). In particular, the higher repair accuracy compared to the probabilistic method [22] (that do not follow the minimum modification principle) verifies the rationale of minimizing changes in timestamp repairing.

The remaining of this paper is organized as follows. In Sect. 2, we show NP-hardness of the repairing problem (Theorem 1). The major challenge originates from the numerous possible timestamps. Intuitively, instead of considering arbitrary combination of timestamps in Sect. 3, we show that any repair could be transformed to a special class of solutions with tight chains (Proposition 3). This property on tight chains is important, since it implies an optimal repair, composed of unchanged assignments and tight edges (Corollary 5). Therefore, we can capture a set of candidates via unchanged assignments and tight edges in Sect. 4. We present the exact methods for computing the optimal solution from the captured candidates, together with pruning and indexing techniques, in Sect. 5. Moreover, in Sect. 6, we study two approximate approaches to get a feasible solution more efficiently, including a simple heuristic repair and the linear programming relaxation approximation. Besides, the algorithm is extended to support incremental computation in a streaming setting in Sect. 7. Section 8 reports an experimental evaluation, and Sect. 9 discusses related studies. Finally, in Sect. 10 we conclude this paper. All the proofs can be found in Appendix.

2 Preliminaries

In this section, we first formally introduce the employed temporal constraints. The timestamp repairing problem is then presented together with hardness analysis.

2.1 Temporal constraints

Consider a set of variables, $X_1, \ldots, X_n$. Each variable X_i represents an absolute value of the time of event i, taking values from a domain D of possible timestamps. For instance, the timestamp of event 1 in trace 1 is 09:05 in Fig. 1.

A *simple temporal network* (STN) is a directed constraint graph, $S = (N, E)$, whose nodes $N = \{1, \ldots, n\}$ represent variables/events and an edge $i \rightarrow j$ $(i, j \in N)$ indicates that a constraint S_{ij} is specified. Each constraint S_{ij} specifies a single interval, $[a_{ij}, b_{ij}]$, to constrain the permissible values for the distance $X_j - X_i$, represented by $a_{ij} \leq X_j - X_i \leq b_{ij}$.

A tuple $x = (x_1, \ldots, x_n)$ is the assignment of variables $X_1, \ldots, X_n$, $\{X_1 = x_1, \ldots, X_n = x_n\}$, denoting the timestamps of events $1, \ldots, n$ in a trace. $X_1, \ldots, X_n$ can be in any order, e.g., in alphabetical order of event names. For example, in Fig. 1, events $1, \ldots, 5$ are ordered referring to the prerequisites of events in a process specification. There are two tuples presented, denoting the timestamps of events $1, \ldots, 5$ in two traces, respectively. It is possible that some event i may not present in a trace, with different trace lengths (dimensionality). The corresponding assignment x_i is thus null. We make an agreement that a null assignment of x_i would not violate

Table 1 Frequently used notations

	Description
X	A set of n variables $\{X_1, \dots, X_n\}$
x	A tuple $(x_1, \dots, x_n)$ of assignments of n variables
M	Minimal network of temporal constraints
N	Nodes in the network of temporal constraints
d_{ij}	A label of temporal constraint from nodes i to j
M_{ij}	Temporal constraints $[-d_{ji}, d_{ij}]$ on variables X_i, X_j
$\vDash$	Timestamps satisfy temporal constraints
Δ	Distance cost of repairing timestamps
t_i	A timestamp candidate for X_i
T_i	A set of timestamp candidates for X_i
T	A set of timestamp candidate sets $\{T_i \mid 1 \le i \le n\}$
$\langle x, T \rangle$	A problem of repairing x with candidates T
N_m	A set of changed nodes connected via tight chains, which are proposed to vary together
N_p	The nodes in N_m preferring to decrease
N_q	The nodes in N_m who want increasing
θ	The amount of variation that makes x'_j unchanged
η	The maximum amount of allowed variation such that no violation is introduced to any $k, k \notin N_m$

any temporal constraints (and thus is ignored in timestamp repairing). A tuple $x = (x_1, \dots, x_n)$ is called a *solution* if the assignment $\{X_1 = x_1, \dots, X_n = x_n\}$ satisfies all the constraints. Let $(x_i, x_j) \vDash S_{ij}$ denote x_i, x_j satisfying the constraint S_{ij}, i.e., $a_{ij} \le x_j - x_i \le b_{ij}$. The solution x satisfying all the constraints, $(x_i, x_j) \vDash S_{ij}, \forall i, j \in N$, is denoted by $x \vDash S$.

It is notable that the constraints between two events i and j declared in a given simple temporal network (STN) might not be tight, referring to the constraints on i and j derived from other events. Thereby, the distance graph of the simple temporal network is employed to find the corresponding minimal (tight) constraints. A *distance graph*, $S_d = (N, E_d)$, is a directed edge-weighted graph. Initially, the structure of the distance graph is determined by the input simple temporal network S. It has the same node set as S. Each directed edge $i \rightarrow j$ is associated with a weight c_{ij} (a fixed threshold value). It denotes that various possible timestamp values of events i and j in a trace should always have difference no greater than c_{ij}, i.e., $X_j - X_i \le c_{ij} = b_{ij}$ exactly the same as $[a_{ij}, b_{ij}]$ given in S. ($X_i - X_j \le c_{ji} = -a_{ij}$ corresponds to the edge $j \rightarrow i$ in the other direction.) Let d_{ij} denote the length of the shortest path from i to j, w.r.t. the edge weights c_{ij} in the distance graph. An equivalent *minimal network representation*, M, of S is defined by $M_{ij} = [-d_{ji}, d_{ij}], \forall i, j \in N$. It implies $-d_{ji} \le X_j - X_i \le d_{ij}$.

A *general temporal network* G generalizes STN by labeling multiple intervals to an edge. A tuple x satisfies G, $x \vDash G$, if one of the intervals in G_{ij} is satisfied for each edge $i \rightarrow j$, denoted by $(x_i, x_j) \vDash G_{ij}$. Considering the combinations of intervals among edges, G can be represented by a set of STNs S. A minimal network M for G is obtained by the union of minimal network representations of all S [14].

As studied in [14], by applying all-pair-shortest-path algorithm to the distance graph, the minimal network representation M can be constructed from the simple temporal network S, in $O(n^3)$ time. Since such a construction is out the scope of this study and could be done in preprocessing, we start directly from the minimal network M given as input, and focus on the timestamp repairing w.r.t. M.

2.2 Repair model

A trace may have events with timestamps conflicting the temporal constraints M (specified by weights on graphs). That is, the tuple x corresponding to the trace does not satisfy the temporal constraints M, denoted by $x \nvDash M$. In this case, timestamp repairing is applicable to find another tuple x' by modifying the assignment in x such that the temporal constraints are satisfied $x' \vDash M$.

Along the same line of minimum change principle in data repairing [9,11] (with an intuition that human or machines always try to minimize their mistakes), the repair cost is evaluated by

$$\Delta(x, x') = \sum_{i=1}^{n} |x_i - x'_i|, \tag{1}$$

where $|x_i - x'_i|$ denotes the absolute difference between the original timestamp x_i and the repaired timestamp x'_i.

Given a tuple x of assignment over temporal constraints M, the *timestamp repairing problem* is to find a repair x' of x such that $x'_i \in D$, $x' \vDash M$ and $\Delta(x, x')$ is minimized.

Theorem 1 *The timestamp repairing problem is* NP*-hard. (Proof can be found in Appendix A.1.)*

The rationale of the minimum change repair model lies in two aspects. (1) Errors often occur minimally on only a fraction of attributes, since human or machines always try to avoid mistakes in practice [11]. For example, usually only one or several RFID tags are broken at a time. (2) The value change is expected to be minimized, following the intuition that the modification should avoid losing information of the original data [19]. For instance, the timestamp 14:__ in σ_2 in Fig. 1 records in hour granularity. It is sufficient to repair the minute value, e.g., 14:52, under the temporal constraints. Further excessively changing the hour value, such as 15:02, is unnecessary, meaningless and leading to information loss in the original hour value 14. It is true that the minimum change repair model [9,11] cannot find the true solution if another viable solution has smaller repair cost. Without further knowledge, one can hardly distinguish the viable solution from the truth.

Since the timestamp repairing problem is hard, we propose the exact solutions with pruning and index, as well as approximate methods that can be adopted in different use cases. The exact algorithms are devised for the scenario that requires higher accuracy, while the approximate algorithms are more flexible in time efficiency.

3 Solution transformation

In this section, we transform a given repair x' to another x'' such that each changed node ($x''_i \neq x_i$) is tightly connected to some unchanged node (see *tight* definition below). Intuitively, this transformation applies to the optimal solution as well, and enlightens the candidate generation w.r.t. unchanged timestamps and tight connections (in Sect. 4). Interestingly, the transformation algorithm can also be adapted to round the solution into the domain of possible timestamps for linear programming approximation (in Sect. 6.2).

3.1 Tightly connected nodes

Consider any repair $x' \vDash M$. We call $i \to j$ a *tight edge* if the time difference of events i and j equals exactly the endpoint of the interval in temporal constraints M, i.e., $x'_j - x'_i = d_{ij}$. Nodes connected via tight edges are then grouped together as follows (for transformation).

Definition 1 A *tight chain* between i and j, denoted by $\langle k_0 = i, k_1, \ldots, k_\ell = j\rangle$, includes ℓ tight edges, having either

$$x'_{k_{y-1}} - x'_{k_y} = d_{k_y k_{y-1}} \quad \text{(i.e., tight edge } k_y \to k_{y-1}) \quad \text{or}$$
$$x'_{k_y} - x'_{k_{y-1}} = d_{k_{y-1} k_y} \quad \text{(i.e., tight edge } k_{y-1} \to k_y),$$

$\forall y = 1, \ldots, \ell$.

Let N_u denote a set of nodes i that are either unchanged in repairing ($x'_i = x_i$) or connected to some unchanged j via a tight chain between i and j. The goal of transformation is to move all nodes into N_u without increasing repair cost.

Moving Tightly Connected Nodes Together

Consider a changed node i, $x'_i \neq x_i$ (say $x'_i > x_i$; similar moving transformation can be made for $x'_i < x_i$ too). To ensure the non-increasing repair cost, we could decrease x'_i. However, there may exist some other x'_j having $x'_j - x'_i = d_{ij}$. That is, x'_i could not decrease solely, owing to the temporal constraints. Instead, we need to alter some other assignments, such as x'_j with tight edge $i \to j$, together with the decrease of x'_i.

Let N_m denote a set of changed nodes connected via tight chains, which are proposed to vary together, such as the aforesaid i, j connected by tight edge $i \to j$. We consider

$$N_p = \{j \in N_m \mid x'_j > x_j\}, N_q = \{j \in N_m \mid x'_j < x_j\},$$

where N_p are the nodes preferring to decrease, while N_q are the nodes who want increasing.

3.2 Transformation without increasing cost

Intuitively, if $|N_p| \geq |N_q|$, by decreasing a very small δ, $\delta > 0$, for all x'_j in N_m, we can obtain another x'', having $x''_j = x'_j - \delta$, $j \in N_m$, such that for any $x'_j > x_j$ it retains $x''_j > x_j$. That is, the sets N_p, N_q have no change. It follows

$$\Delta(x, x') - \Delta(x, x'') = \sum_j |x_j - x'_j| - |x_j - x''_j|$$
$$= |N_p|\delta - |N_q|\delta \geq 0 \qquad (2)$$

If there is no other node k outside N_m which prevents the decrease, we still have $x'' \vDash M$ after transformation.

For the amount δ that is allowed to move, we consider

$$\eta = \min_{j \in N_m, k \notin N_m, d_{jk} \in M} d_{jk} - (x'_k - x'_j). \qquad (3)$$

It denotes the maximum amount of allowed variation such that no violation will be introduced to any k, $k \notin N_m$. Eq. (3) ensures that, after reducing x'_j by η, $x'_k - (x'_j - \eta) \leq d_{jk}$ is still satisfied, for all $j \in N_m$, $k \notin N_m$, $d_{jk} \in M$. That is, decreasing x'_j by η, $\forall j \in N_m$, is allowed.

Recall that the goal of solution transformation is to show that a repair x'_j is either unchanged ($x'_j = x_j$) or tightly connected to some other unchanged x'_i. We consider the following amount θ of variation that can make x'_j unchanged,

$$\theta = \min_{j \in N_p} x'_j - x_j. \tag{4}$$

The min operator ensures that any variation less than θ will not change the relationship between x'_j and x_j for all $j \in N_p$. And thus, $|N_p| \geq |N_q|$ retains (as decreasing x'_j will never affect $x'_j < x_j$ in N_q).

While θ denotes the variation that is sufficient to obtain an unchanged node, η specifies the maximum variation amount allowed. The moving amount is thus determined by $\delta = \min(\theta, \eta)$, which corresponds to two cases below:

Case 1 For $\theta > \eta$, we assign $x''_j = x'_j - \eta, \forall j \in N_m$. It creates a new solution with tight edge $j \to k, x''_k - x''_j = d_{jk}$, for some $j \in N_m, k \notin N_m, d_{jk} \in M$, having $d_{jk} - (x'_k - x'_j) = \eta$ before decreasing x'_j by η.

If $k \in N_u$, all the nodes j in N_m find their connections to unchanged nodes in N_u (recall that nodes in N_m are connected with each other by tight edges so that have to vary together), and N_m can be merged with N_u; otherwise, k is moved to N_m, and the transformation carries on over N_m.

Case 2 For $\theta \leq \eta$, we assign $x''_j = x'_j - \theta, \forall j \in N_m$. It creates a new solution with unchanged $x''_j = x_j$, for some $j \in N_m$, having $x'_j - x_j = \theta$ before reducing x'_j by θ.

Hence, we move all the nodes in N_m to N_u.

Proposition 2 *The transformation from repair x' to another x'' satisfies that (1) the repair cost does not increase, $\Delta(x, x'') \leq \Delta(x, x')$, and (2) each changed node ($x''_i \neq x_i$) in the new x'' is tightly connected to some unchanged node. (Proof can be found in Appendix A.2.)*

3.3 Transformation algorithm

Algorithm 1 shows the procedure of the aforesaid transformation from x' to x''. Lines 12 to 14 assemble N_m w.r.t. tight edges. For $|N_p| \geq |N_q|$, N_m proposes to decrease as presented in Sect. 3.2. Otherwise, Lines 27 to 30 increase the assignment for nodes in N_m.

The running example of Algorithm 1 is presented in Sect. 5.4.2.

Proposition 3 *Algorithm 1 runs in $O(n^2)$ time, and outputs a repair x'', such that (1) $\Delta(x, x'') \leq \Delta(x, x')$ and (2) for each $x''_j \neq x_j$, there is a tight chain, $\langle k_0 = i, k_1, \ldots, k_\ell = j\rangle$, where $x''_i = x_i$. (Proof can be found in Appendix A.3.)*

Algorithm 1: Transform(M, x, x')

```
Input: a repair $x'$ of $x$
Output: a repair $x''$ where each changed node is connected to
        some unchanged node by a tight chain
1  $N_v \leftarrow$ the set of $n$ (unvisited) nodes;
2  $N_u \leftarrow \emptyset$; $N_m \leftarrow \emptyset$;
3  while $N_v$ is not empty do
4      move one node $i$ from $N_v$ to $N_m$;
5      $\alpha_i \leftarrow \max_{k \in N_v \cup N_u, d_{ik} \in M} x'_k - x'_i - d_{ik}$;
6      $\beta_i \leftarrow \max_{k \in N_v \cup N_u, d_{ki} \in M} x'_i - x'_k - d_{ki}$;
7      if $\alpha_i > 0$ then            // make feasible solution
8          $x'_i \leftarrow x'_i + \alpha_i$;
9      else if $\beta_i > 0$ then
10         $x'_i \leftarrow x'_i - \beta_i$;
11     while $N_m$ is not empty do
12         for each $j \in N_m, i \in N_v, d_{ji}, d_{ij} \in M$ do
13             if $x'_i - x'_j = d_{ji}$ or $x'_j - x'_i = d_{ij}$ then
14                 move node $i$ with $\alpha_i \leq 0$ and $\beta_i \leq 0$ from $N_v$ to
                   $N_m$;
15         for each $j \in N_m, k \in N_u, d_{jk}, d_{kj} \in M$ do
16             if $x'_k - x'_j = d_{jk}$ or $x'_j - x'_k = d_{kj}$ then
17                 move all nodes $j$ from $N_m$ to $N_u$;
18         for each $j \in N_m$ do
19             if $x'_j = x_j$ then          // unchanged repair
20                 move all nodes $j$ from $N_m$ to $N_u$;
21         $N_p \leftarrow \{j \in N_m \mid x'_j > x_j\}$;
22         $N_q \leftarrow \{j \in N_m \mid x'_j < x_j\}$;
23         if $|N_p| \geq |N_q|$ then          // decrease $N_m$
24             $\eta \leftarrow \min_{j \in N_m, k \in N_v \cup N_u, d_{jk} \in M} d_{jk} - (x'_k - x'_j)$;
25             $\theta \leftarrow \min_{j \in N_p} x'_j - x_j$;
26             $x'_j \leftarrow x'_j - \min(\eta, \theta), \forall j \in N_m$;
27         else                              // increase $N_m$
28             $\eta \leftarrow \min_{j \in N_m, k \in N_v \cup N_u, d_{kj} \in M} d_{kj} - (x'_j - x'_k)$;
29             $\theta \leftarrow \min_{j \in N_q} x_j - x'_j$;
30             $x'_j \leftarrow x'_j + \min(\eta, \theta), \forall j \in N_m$;
31 return $x'$ as a new repair $x''$
```

4 Candidate generation

Intuitively, given any optimal repair, we transform it to a special form that (1) consists of unchanged assignments and tight edges, and (2) is still optimal, referring to the non-increasing cost during transformation. Such a property enlightens us on capturing a set of candidates via unchanged assignments and tight edges (in this section), and finding the optimal solution from the candidates (in Sect. 5). Moreover, the simple heuristic approximation (in Sect. 6.1) is also built upon the generated candidates.

4.1 Candidates from tight chains

Consider an optimal repair solution x^* of x whose repair cost $\Delta(x, x^*)$ is minimized and $x^* \models M$. We first show that the nodes must be tightly connected in the assignment.

Lemma 4 *For any $x^*_i > x_i$ in an optimal solution $x^* = (x^*_1, \ldots, x^*_n)$, there must exist some j such that $x^*_j - x^*_i =$*

d_{ij}, where d_{ij} is the endpoint of the interval in temporal constraints M. (Proof can be found in Appendix A.4.)

Similarly, for $x^*_i < x_i$, there must exist an tight edge in the form of $j \rightarrow i$ that $x^*_i - x^*_j = d_{ji}$, i.e., increasing x^*_i is impossible. In the following, we consider $x^*_i > x_i$ by default, while the same results apply to the other case $x^*_i < x_i$.

Moreover, the following conclusion states that there is an optimal solution x^* whose nodes are not only tightly connected but also connected to unchanged nodes.

Corollary 5 *An optimal solution $x^* = (x^*_1, \dots, x^*_n)$ can always be found such that each changed $x^*_j, x^*_j \neq x_j$, is connected to some unchanged $x^*_i = x_i$ via a tight chain. (Proof can be found in Appendix A.5.)*

We now generate the repair candidate for node j w.r.t. unchanged node i and tight chain $\langle k_0 = i, k_1, \dots, k_\ell = j\rangle$. By summation of $x'_{k_y} - x'_{k_{y-1}} = d_{k_{y-1}k_y}$ (or $-x'_{k_{y-1}} + x'_{k_y} = -d_{k_y k_{y-1}}$) for all tight edges in the chain, the repair candidate for x'_j is computed by

$$x'_j = x_i + \sum_{\substack{y=1 \\ k_{y-1}\rightarrow k_y \text{ in chain}}}^{\ell} d_{k_{y-1}k_y} - \sum_{\substack{y=1 \\ k_y\rightarrow k_{y-1} \text{ in chain}}}^{\ell} d_{k_y k_{y-1}}. \tag{5}$$

Considering all the tight chains connecting to possible unchanged node i, we generate a set of repairing candidates T_j for each node j. According to Corollary 5, an optimal repair solution can always be found over T_j for all nodes j.

For a node j, there are $n-1$ possible unchanged nodes for tight chains with length 1. Each may suggest $2c$ candidates, where c is the maximum number of intervals labeling an edge in M. For tight chains with length 2, there are at most $(2c)^2(n-1)(n-2)$ candidates. For tight chains with length $n-1$, the maximum size of candidates is $(2c)^{n-1}(n-1)!$.

4.2 Towards more concise candidates

In the following, we show that it is not necessary to consider all the possible tight chains with arbitrary tight edge combinations. Instead, the chains in the transformation result follow certain patterns (namely *provenance chains*, a particular class of structures with alternating edges). Intuitively, since any tight chain can be reduced to a provenance chain (Lemma 6), it is sufficient to consider provenance chains in candidate generation (Propositions 7).

Definition 2 A *provenance chain* between i and j is a tight chain, $\langle k_0 = i, k_1, \dots, k_\ell = j\rangle$, such that the tight edges are in the form of either

$$k_0 \rightarrow k_1, k_1 \leftarrow k_2, k_2 \rightarrow k_3, k_3 \leftarrow k_4, k_4 \rightarrow k_5, \dots \quad \text{or}$$
$$k_0 \leftarrow k_1, k_1 \rightarrow k_2, k_2 \leftarrow k_3, k_3 \rightarrow k_4, k_4 \leftarrow k_5, \dots$$

That is, the directions of consecutive tight edges are always flipped in the chain. (See the running example in Sect. 5.4.3.)

Lemma 6 (Transitivity on tight edges) *For any x', if there are two tight edges $i \rightarrow j$ and $j \rightarrow k$, having $x'_j - x'_i = d_{ij}$ and $x'_k - x'_j = d_{jk}$, respectively, it always implies the tight edge $i \rightarrow k$ with $x'_k - x'_i = d_{ik}$. (Proof can be found in Appendix A.6.)*

With this transitivity on tight edges, all the tight chains (by transformation) can be reduced to provenance chains.

Proposition 7 *An optimal solution $x^* = (x^*_1, \dots, x^*_n)$ can always be found such that each changed $x^*_j, x^*_j \neq x_j$, is connected to some unchanged $x^*_i = x_i$ via a provenance chain. (Proof can be found in Appendix A.7.)*

According to Proposition 7, it is sufficient to consider candidates w.r.t. provenance chains. Instead of two alternative directions in expanding a tight chain, the provenance chain has only one choice determined by the preceding one. The number of candidates is thus significantly reduced.

Provenance chains with length 2 suggest at most $2c^2(n-1)(n-2)$ candidates rather than $(2c)^2(n-1)(n-2)$ by tight chains, where c is the maximum number of intervals labeling an edge in M. For provenance chains with length $n-1$, the maximum size of candidates is $2c^{n-1}(n-1)!$.

4.3 Candidate generation algorithm

Algorithm 2 generates a finite set of candidates for timestamp repairing, by considering (all) the possible provenance chains. (See the running example in Sect. 5.4.3.)

Line 2 initializes the start point of all possible provenance chains, whose timestamps are not changed, i.e., the original x_i. Procedure Generate(N_c, t, i, direction) recursively expands the chain on the remaining variables, where N_c is the currently processed nodes, t is the tuple of candidates over N_c, i is the current ending (latest expanded) point of the chain, and "direction" is the direction of the last edge (on i). Finally, the algorithm returns T, where each $T_i \in T$ is a set of candidate timestamps for variable X_i.

Suppose that a solution $x_{\min}$ is known in advance to be feasible w.r.t. M (see Sect. 6 below for how to obtain such a solution from the aforesaid solution transformation). Let $\Delta_c(x, t) = \sum_{i \in N_c} |x_i - t_i|$ denote the currently paid cost for generating the chain over N_c. If $\Delta_c(x, t)$ has already exceeded $\Delta(x, x_{\min})$ of the given repairing solution $x_{\min}$,

Algorithm 2: Candidate($M, x, x_{\min}$)

```
Input: a minimal network $M$, a tuple $x$, a currently known
       feasible solution $x_{\min}$
Output: $T$ where each $T_i \in T$ is a set of candidate timestamps
        for variable $X_i$
1  $N := \{1, \ldots, n\}$;
2  initialize $T := \{T_i \mid i \in N\}$ where each $T_i := \{x_i\}$;
3  visited := $\emptyset$;
4  for each $i \in N$ do
5      $t_i := x_i$;
6      Generate ($\{i\}, t, i$, out);
7      Generate ($\{i\}, t, i$, in);
8  return $T$;
1  Procedure Generate ($N_c, t, i$, direction)
2      if $\Delta_c(x, t) < \Delta(x, x_{\min})$ and $t \vDash_c M$ and
        ($N_c, t, j$, direction) $\notin$ visited then
3          visited := visited $\cup$ $\{(N_c, t, i, \text{direction})\}$;
4          if $N_c = N$ then
5              $x_{\min} := t$;
6              return;
7          for each $j \in N \setminus N_c, d_{ij}, d_{ji} \in M$ do
8              if direction = out then
9                  $t_j := t_i + d_{ij}$;
10                 flipped := in;
11             else if direction = in then
12                 $t_j := t_i + d_{ji}$;
13                 flipped := out;
14             $T_j := T_j \cup \{t_j\}$ for $T_j \in T$;
15             Generate($N_c \cup \{j\}, t, j$, flipped);
```

there is no need to further expand the chain, i.e., pruning candidates by $x_{\min}$ in Line 2 in Generate.

Moreover, if the currently generated candidates in the chain already violates the temporal constraints M, the expansion terminates. We say that t over N_c partially satisfies M, denoted by $t \vDash_c M$, if $\forall i, j \in N_c$ having $(t_i, t_j) \vDash M_{ij}$. Line 2 in Generate carries on chain expansion if $t \vDash_c M$.

Lines 7 to 15 of Generate consider the possible chain expansion on each remaining node $j \in N \setminus N_c$.

The correctness is verified by showing that the product of candidates in T_i, i.e., $\prod_{T_i \in T} T_i$, includes all the possible provenance chains. In other words, an optimal solution always exists by assembling candidate timestamps in T_i for each X_i. (It is worth noting that there may be multiple unchanged nodes in an optimal solution, that is, we might not be able to obtain the optimal solution by simply considering all the provenance chains with length $n - 1$.)

Although the candidate size could be very large w.r.t. n, as shown in the experiments, by restricting a maximum length of provenance chains in expanding, the number of candidates as well as generation time costs can be significantly reduced, while the corresponding repair accuracy keeps high.

5 Exact algorithm

Once a finite set of timestamp candidates T_i are obtained for each variable X_i, we next compute the (optimal) repair solution over the generated timestamp candidates. Intuitively, candidates can be efficiently pruned if their corresponding costs exceed certain bounds (in Sect. 5.1). Moreover, sharing solutions among problems is possible by building a solution index (in Sect. 5.2). Running examples of the consolidated repair procedure are presented in Sect. 5.4

5.1 Repair algorithm

We rewrite the repairing problem as

$$\min \quad \Delta(x, x') \tag{6}$$

$$\text{s.t.} \quad x'_i \in T_i, \quad 1 \le i \le n, \quad x' \vDash M \tag{7}$$

denoted by $\langle x, T\rangle$, where T consists of all candidate sets T_i.

5.1.1 Branch and bound

We call T' a *branch* of T on a $t_k \in T_k, T_k \in T$, where (1) $T'_k = \{t_k\}$, and (2) $T'_j = T_j, \forall j, j \neq k$. The candidate is fixed to t_k in T'_k, when branching from T to T'. A branch T' on t_k is *feasible*, if $\forall T_i \in T$ with $|T_i| = 1$, we have $(t_i, t_k) \vDash M_{ik}, t_i \in T_i$. That is, the new fixed t_k does not introduce violations to the previously fixed candidate t_i.

Let $\Delta_p(x, T)$ denote the repair cost paid on those X_i whose candidates are fixed with $|T_i| = 1$, i.e., $\Delta_p(x, T) = \sum_{t_i \in T_i, |T_i|=1, T_i \in T} |x_i - t_i|$.

Algorithm 4 considers a feasible branch T' in each iteration in Line 15, and carries on branching if $\Delta_p(x, T')$ is less than the bound $\Delta(x, x_{\min})$, where $x_{\min}$ is the currently known best solution.

5.1.2 Candidate pruning during repairing

Given any T together with a currently known best solution $x_{\min}$, Algorithm 3 considers the pruning of candidates in $T_i \in T$ in the following aspects.

1. Any $t_i \in T_i$ can be removed, if $|t_i - x_i| > \Delta(x, x_{\min})$, in Lines 4-5 of Algorithm 3. That is, the cost of repairing by t_i is already greater than the currently known solution.
2. For a $T_i = \{t_i\}$, any $t_j \in T_j$ such that $(t_i, t_j) \nvDash M_{ij}$ can be pruned. In other words, the remaining candidates t_j should not have violation to any t_i with no other choices.
3. For any $t_i, t'_i \in T_i$, $|t_i - x_i| \le |t'_i - x_i|$, if $(t_i, t_j) \vDash M_{ij}$ and $(t'_i, t_j) \vDash M_{ij}, \forall j$, then t'_i can be pruned, in Lines 6-10. The rationale is that t_i, t'_i have no difference in determining the remaining repairs.

Algorithm 3: Prune($M, T, x, x_{\min}$)

```
   Input: $M, T, x, x_{\min}$ the currently known best solution
   Output: $T$ pruned candidates
1  for each $T_i \in T, |T_i| > 1$ do
2      $w_{\min} := +\infty$;
3      for each $t_i \in T_i$ do
4          if $|t_i - x_i| > \Delta(x, x_{\min})$ then
5              $T_i := T_i \setminus \{t_i\}$;
6          if $(t_i, t_j) \vDash M_{ij}, \forall t_j \in T_j, T_j \in T$ then
7              $T_i := T_i \setminus \{t_i\}$;
8              if $|t_i - x_i| < w_{\min}$ then
9                  $w_{\min} := |t_i - x_i|, t_{\min} := t_i$;
10     $T_i := T_i \cup \{t_{\min}\}$;
11 for each $t_i \in T_i, |T_i| = 1, T_i \in T$ do
12     for each $t_j \in T_j, |T_j| > 1, T_j \in T$ do
13         if $(t_i, t_j) \nvDash M_{ij}$ then
14             $T_j := T_j \setminus \{t_j\}$ for $T_j \in T$;
15 return $T$
```

Proposition 8 *The pruning in Algorithm 3 is safe, and runs in $O(a^2n^2)$ time, where a is the maximum size of candidates in T_i. (Proof can be found in Appendix A.8).*

5.1.3 Repairing with pruning

In Algorithm 4, Line 1 employs candidate pruning by Prune($M, T, x, x_{\min}$) in Algorithm 3. In each iteration, Line 13 chooses a branch. By removing Lines 2–7 and 21 (which are used for indexing, see details in Sect. 5.2) and Lines 19–20 (which are used for heuristic approximation, see details in Sect. 6.1) , the branching will continue to compute other solutions. Finally, the program outputs $x_{\min}$ as the optimal solution.

Proposition 9 *Algorithm 4 (without Lines 2–7 and 21 for indexing and Lines 19–20 for heuristic) returns the optimal solution, and runs in $O(a^n)$ time, where a is the maximum size of candidates in T_i. (Proof can be found in Appendix A.9.)*

5.2 Indexing optimal solutions

The aforesaid repair procedure only considers to prune candidates that cannot be the optimal solution by using Algorithm 3. However, there exist problems sharing the same optimal solutions during the recursive repair procedure, as introduced in Sects. 5.2.1 and 5.2.2. A natural idea is thus to cache (and index) the solutions to avoid repetitive computation for the same/similar problems. Sharing solutions when given the same $\langle x, T\rangle$ is trivial, whereas its chance of being shared is limited. To enable sharing among different problems, we introduce a *subproblem* relationship. To further increase the sharing chance, an advanced *safe-subproblem* is presented (in Proposition 12).

Algorithm 4: Repair($M, T, x, x_{\min}, k$) with pruning and index

```
   Input: $M, T, x, x_{\min}$ the currently known best solution, $k$ the
          node to branch
   Output: $x_{\min}$
1  $T :=$ Prune($M, T, x, x_{\min}$);
2  for each $\langle\tilde{x}, \tilde{T}\rangle$ with solution $\tilde{x}'$ returned by range query do
3      if $\langle x, T\rangle$ is (safe-)subproblem of $\langle\tilde{x}, \tilde{T}\rangle$ and $\tilde{x}'_i \in T_i$, for all
        (non-safe) $T_i \in T$ then
4          $x' :=$ solution computed from $\tilde{x}'$ by Eq. (8);
5          if $\Delta(x, x') < \Delta(x, x_{\min})$ then
6              $x_{\min} := x'$;
7          return $x_{\min}$
8  if $k > n$ then
9      $x' :=$ solution where $x'_i = t_i, T_i = \{t_i\}, \forall T_i \in T$;
10     return $x'$
11 BC $:= T_k$;
12 while BC $\neq \emptyset$ do
13     remove a $t_k$ from BC;
14     $T' :=$ a branch of $T$ on $t_k$;
15     if $T'$ is feasible and $\Delta_p(x, T') < \Delta(x, x_{\min})$ then
16         $x' :=$ Repair($M, T', x, x_{\min}, k + 1$);
17         if $\Delta(x, x') < \Delta(x, x_{\min})$ then
18             $x_{\min} := x'$;
19     if $x_{\min}$ is feasible/not null then
20         break;      // for heuristic approximation
21 insert $\langle x, T\rangle$ with solution $x_{\min}$ to index;
22 return $x_{\min}$
```

5.2.1 Index on optimal solutions for subproblems

We call $\langle x, T\rangle$ a *subproblem* of another problem $\langle\tilde{x}, \tilde{T}\rangle$ if $T_i \subseteq \tilde{T}_i, |t_i - x_i| = |t_i - \tilde{x}_i|, \forall t_i \in T_i, T_i \in T$, i.e., sharing as a subset of candidates with the same repair cost.

Let $\tilde{x}'$ be an optimal solution for the problem $\langle\tilde{x}, \tilde{T}\rangle$.

Proposition 10 *For any subproblem $\langle x, T\rangle$ of $\langle\tilde{x}, \tilde{T}\rangle$, if the optimal solution $\tilde{x}'$ of $\langle\tilde{x}, \tilde{T}\rangle$ has $\tilde{x}'_i \in T_i, \forall T_i \in T$, then $\tilde{x}'$ is also an optimal solution of $\langle x, T\rangle$. (Proof can be found in Appendix A.10.)*

It enables us to share the solutions among problems. However, given the complex subset relationships between problems and their subproblems, it is difficult to efficiently materialize and re-access the subproblems.

Intuitively, we can relax the strict subproblem settings for efficient filtering. Proposition 10 implies an additional requirement on the ranges of candidates w.r.t. the optimal solution, i.e., $\min T_i \leq \tilde{x}'_i \leq \max T_i, \forall i$. We can employ this range requirement to filter subproblems.

Specifically, to support efficient access regarding the range requirements, we build a *k-d tree* [8] that supports both efficient range and nearest neighbor (NN) queries. Each leaf node is a n-dimensional optimal solution $\tilde{x}'$. We extend the tree index, by appending each leaf node $\tilde{x}'$ a list of problems $\langle\tilde{x}, \tilde{T}\rangle$ that yield the optimal solution $\tilde{x}'$. While we assume all tuples have the same dimensionality, as also introduced in Sect. 2.1, it is possible that some event i may not present in a trace, with different trace lengths. In this case, the corresponding assignment x_i is set to null. In order to index such tuples, we simply denote the null value by a large positive number that is greater than all the timestamps in the domain D.

Given a problem $\langle x, T\rangle$, a range query with [$\min T_i$, $\max T_i$] on each i is posed. It returns all the cached optimal solutions that are in the ranges specified by $\langle x, T\rangle$. For each returned optimal solution $\tilde{x}'$, we investigate whether $\langle x, T\rangle$ is a subproblem of any $\langle \tilde{x}, \tilde{T}\rangle$ that yields the solution $\tilde{x}'$. If yes, referring to Proposition 10, it is possible that $\tilde{x}'$ is also the optimal solution of $\langle x, T\rangle$.

5.2.2 Enhance reuseability

Intuitively, to increase the chance of an indexed solution being used, we show in the following that the strict condition on all dimensions i in the range query can be relaxed.

We call $T_i \in T$ a *safe* candidate set in $\langle x, T\rangle$ if each candidate $t_i \in T_i$ is not in violation with all the candidates in other nodes, i.e., $(t_i, t_j) \vDash M_{ij}, \forall t_j \in T_j, T_j \in T$.

Lemma 11 *For a $\langle x, T\rangle$ after candidate pruning, each $T_i \in T$ with $|T_i| = 1$ is safe. (Proof can be found in Appendix A.11.)*

We call $\langle x, T\rangle$ a *safe-subproblem* of $\langle \tilde{x}, \tilde{T}\rangle$ if (1) for all non-safe $T_i \in T$, it has $T_i \subseteq \tilde{T}_i$ and $|t_i - x_i| = |t_i - \tilde{x}_i|, \forall t_i \in T_i$; and (2) for all safe $T_j \in T$, $\tilde{T}_j$ is also safe. It is remarkable that $T_j \subseteq \tilde{T}_j$ is no longer necessary for safe T_j and $\tilde{T}_j$.

Proposition 12 *For any safe-subproblem $\langle x, T\rangle$ of $\langle \tilde{x}, \tilde{T}\rangle$, if the optimal solution $\tilde{x}'$ of $\langle \tilde{x}, \tilde{T}\rangle$ has $\tilde{x}'_i \in T_i$, for all non-safe $T_i \in T$, an optimal solution x' of $\langle x, T\rangle$ can be*

$$x'_i = \begin{cases} \arg\min_{t_i \in T_i} |t_i - x_i| & \textit{if } T_i \textit{ is safe} \\ \tilde{x}'_i & \textit{otherwise} \end{cases} \tag{8}$$

(Proof can be found in Appendix A.12.)

Consequently, referring to the safe-subproblem definition, we only need to consider the range query with intervals [$\min T_i$, $\max T_i$] on non-safe $T_i \in T$. With Proposition 12, an optimal solution x' could possibly be obtained.

5.2.3 Repairing with index and pruning

We extend the repair procedure by integrating with the optimal solution indexing in Algorithm 4. Lines 2–7 search index for possible optimal solutions, and return in an early stage if the target solution is already indexed. Otherwise, we adopt the same steps of repairing with pruning as described in Sect. 5.1.3. Finally, in Line 21, we maintain the index by inserting new optimal solutions. Indeed, the preliminary version of the Repair Algorithm 4 in [23] is indeed a special case of the new Algorithm 4 in this study, without the indexing mechanism in Lines 2–7 and 21.

It is notable that the indexing is mostly an implementation optimization, which gives a good performance improvement, but does not change the approach.

5.3 Putting techniques together

We now present the consolidated repair procedure with all the aforementioned techniques.

5.3.1 Consolidated repairing procedure

In summary, given temporal constraints M and a tuple x, the overall repairing procedure is:

1. To initialize, we transform x to a valid solution[1] via Algorithm 1, $x_{\min} := \mathsf{Transform}(M, x, \tilde{x}')$;
2. Generating candidates T according to M and x, by the generation Algorithm 2 (with pruning by $x_{\min}$), $T := \mathsf{Candidate}(M, x, x_{\min})$;
3. Solving $\langle x, T\rangle$ by $\mathsf{Repair}(M, T, x, x_{\min}, 1)$ in Algorithm 4.

5.3.2 Components of temporal constraints

In practice, one may observe that the temporal constraints in M are disjoint. That is, some events do not have any temporal relationships with each other either directly or indirectly through other events. For example, the examination step in designing a part has nothing to do with the part pricing step. In this sense, the events in N could be decomposed into several (c) components, $N_1, \ldots, N_c$, such that $N = N_1 \cup \cdots \cup N_c$. Moreover, the events in any N_i and N_j do not overlap, having $N_i \cap N_j = \emptyset$. Let $M_i \subseteq M$ be the set of temporal constraints in M over the events in N_i. We call $M_1, \ldots, M_c$ the components of temporal constraints in M, if $M = M_1 \cup \cdots \cup M_c$, and for any nodes i and j not belonging to the same component N_k, there is no constraint in M between events i and j, i.e., no M_{ij} exists in M. Since two events not in the same component are irrelevant w.r.t. the temporal constraints M, the repairing on one of them will not affect the other. Thereby, we may repair the timestamps in each component, respectively, and assemble them as the whole repair.

5.4 Running examples

We use running examples to explain the consolidated repair procedure of all the algorithms. The example tuple and tem-

[1] Referring to Proposition 13.

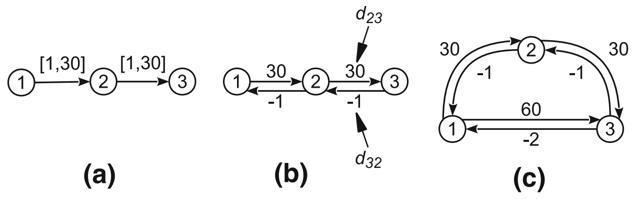


Fig. 2 **a** Example temporal constraint network, **b** the corresponding distance graph, and **c** the corresponding minimal network representation

poral constraint network are introduced in Sect. 5.4.1 (Fig. 2). We illustrate an example of solution transformation using the aforesaid tuple and temporal constraint in Sect. 5.4.2 (Fig. 3). The result after applying the transformation is used for pruning in the following candidate generation and timestamp repairing steps. In Sect. 5.4.3, we first show the examples of tight and provenance chains in Fig. 4. The candidate generation w.r.t the provenance chains is then illustrated in Fig. 5. Finally, with the generated candidates in the previous step, we discuss the repairing process with pruning and indexing in Sects. 5.4.4 and 5.4.5, respectively.

5.4.1 Temporal constraints

Consider an example temporal constraint network in Fig. 2a. Its corresponding distance graph is plotted in Fig. 2b. An edge, e.g., $2 \rightarrow 3$ in the distance graph, with weight $d_{23} = 30$, denotes that $X_3 - X_2 \leq d_{23} = 30$. Together with $X_2 - X_3 \leq d_{32} = -1$, it is equivalent to the constraint $[1, 30]$ in Fig. 2a, i.e., $1 \leq X_3 - X_2 \leq 30$. Figure 2c presents the equivalent minimal network representation M for S, by considering the shortest paths for each pair of nodes in Fig. 2b.

A tuple $x = (0, 30, 66)$, where x_1=0 denotes the timestamp of event 1 and so on, is considered in the following examples. We say that x_1, x_2 satisfy the temporal constraints, $(x_1, x_2) \vDash M_{12}$, since $x_2 - x_1 = 30 \leq 30$ and $x_1 - x_2 = -30 \leq -1$, where 30 and -1, corresponding to edges $1 \rightarrow 2$ and $2 \rightarrow 1$ in Fig. 2c, respectively, are the temporal constraints $[1, 30]$ in Fig. 2a. However, x_2, x_3 with distance $x_3 - x_2 = 36 > 30 = d_{23}$ are identified as violations of the temporal constraints. The inconsistent timestamps should be repaired to resolve the violations.

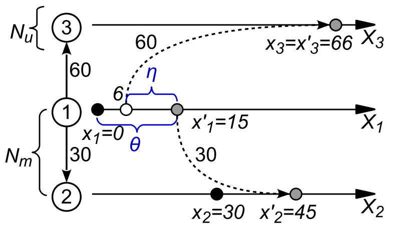


Fig. 3 Example of solution transformation under the temporal constraint in Fig. 2

5.4.2 Solution transformation

Consider a repair $x' = (15, 45, 66)$ of x, as illustrated in Fig. 3. We have node $3 \in N_u$, since $x_3 = x'_3 = 66$ is unchanged.

Nodes in $N_m = \{1, 2\}$ are proposed to move (decrease) together, given $x'_1 > x_1$ and the tight edge $1 \rightarrow 2$ with $x'_2 - x'_1 = 30 = d_{12}$. By solely decreasing x'_1 (e.g., to 6) without changing x'_2, it leads to violation to $x'_2 - x'_1 \leq d_{12} = 30$. Thereby, x'_2 should decrease together with x'_1.

$N_p = \{1, 2\}$ indicates that decreasing is preferred, since $x'_1 > x_1, x'_2 > x_2$.

$|N_p = \{1, 2\}| \geq |N_q = \emptyset|$ implies that by decreasing together the assignments of nodes in N_m, the repair cost will not increase.

Referring to Eq. (3), $\eta = d_{13} - (x'_3 - x'_1) = 60 - (66 - 15) = 9$ requires the amount of decreasing should not exceed 9, otherwise violation occurs between x'_1 and x'_3 (where $3 \notin N_m$). For instance, an assignment $x''_1 = 5$ with decreasing amount $15 - 5 = 10 > \eta = 9$ is not allowed, since $66 - 5 = 61 > d_{13} = 60$.

Referring to Eq. (4), $\theta = x'_1 - x_1 = 15 - 0 = 15$ means that a decreasing amount less than 15 will not change $|N_p = \{1, 2\}| \geq |N_q| = \emptyset$. It ensures the non-increasing repair cost.

After decreasing $\delta = 9$ (according to Case 1 since $\eta < \theta$), $x''_1 = 6$ becomes tightly connected with the unchanged node 3. Node 2 moving together with 1, having $x''_2 = 36$, is still tightly connected to node 1. Therefore, both nodes 1 and 2 in N_m are moved to N_u.

For the repair $x' = (15, 45, 66)$ with cost $\Delta(x, x') = 30$, after applying the transformation (decreasing $N_m = \{1, 2\}$), it forms another repair $x'' = (6, 36, 66)$, with lower cost $\Delta(x, x'') = 12$.

5.4.3 Candidate generation

For generating repair candidates for x, Fig. 4a illustrates a tight chain, where the number on each edge denotes the constraint from the minimal network M. For instance, -1 on 2→1 corresponds to d_{21}=-1 on 2→1 in Fig. 2c. All the nodes are connected via this tight chain to the unchanged node $x'_1 = x_1 = 0$. The number attached to each node i rep-

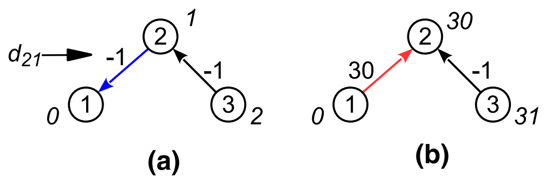


Fig. 4 Examples of **a** tight and **b** provenance chains, for repairing $x = (0, 30, 66)$ under the temporal constraint in Fig. 2

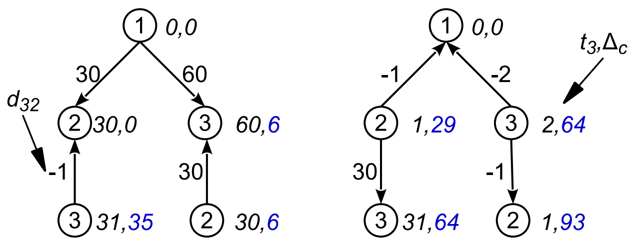


Fig. 5 Example of generating candidates w.r.t the provenance chains connected to the unchanged node 1 as in Fig. 4b

resents a repair candidate x_i', which is computed by Eq. (5). For instance, we have $x_3' = x_1 - d_{21} - d_{32} = 0+1+1 = 2$.

Figure 4b illustrates a provenance chain. As shown, the directions of edges appear alternatively in the chain. For example, the direction of edge $1 \rightarrow 2$ (in red) should be different from the following edge $2 \leftarrow 3$ in Fig. 4b. The tight chain in Fig. 4a has no such constraint, e.g., the edge $1 \leftarrow 2$ (in blue) is acceptable.

As a special tight chain, the repair candidates w.r.t. the provenance chain, $x' = (0, 30, 31)$, are computed by Eq. (5) as well. While candidate generation via tight chains has to consider both Fig. 4a and b, the generation over provenance chains considers Fig. 4b only. It is not surprising that, provenance chains lead to more concise candidate sets, and are more efficient.

Figure 5 illustrates the provenance chains connected to the unchanged x_1. It is indeed a tree rooted in node 1 with height $n-1$. The two numbers attached to each node i denote the candidate t_i and the partial cost $\Delta_c(x,t)$. For example, $(t_3, \Delta_c) = (2, 64)$ attached to node 3 denotes that the cost of generating the current chain $\langle 1 \leftarrow 3\rangle$ is $|0-0| + |66 - 2| = 64$. Suppose that the $x'' = (0, 30, 31)$ with $\Delta(x, x'') = 12$ calculated in Sect. 5.4.2 is used as $x_{\min}$ in pruning the subsequent chains. Consequently, the expansion on the chain $\langle 1 \leftarrow 3\rangle$, whose Δ_c cost is already 64, terminates.

5.4.4 Repairing with pruning

Figure 6a shows the candidate sets obtained for repairing the tuple $x = (0, 30, 66)$. For instance, the candidates for node 2 are $T_2 = \{1, 30, 36, 65\}$, where $x_2 \in T_2$ as well. The number attached to each candidate t_i denotes $|t_i - x_i|$, i.e., the cost needs for such a repair.

Suppose that $t_1 = 0 \in T_1$ of $\langle x, T\rangle$ in Fig. 6a is considered for branching. It yields a branch with new $T_1 = \{t_1 = 0\}$ as shown in Fig. 6b. The paid repair cost is $\Delta_p(x, T) = |t_1 - x_1| = 0$, where t_1 is 0.

Consider that a currently known feasible solution $x_{\min}$ has cost $\Delta(x, x_{\min}) = 12$, which is calculated in Sect. 5.4.2. For the problem $\langle x, T\rangle$ in Fig. 6b, the candidate $t_2 = 1 \in T_2$ with $|t_2 - x_2| = 29 > 12$ can be directly removed according to the pruning rule (1). Similar pruning applies to $65 \in T_2$, $2 \in T_3$and $31 \in T_3$.

Moreover, consider $T_1 = \{t_1 = 0\}$. According to the pruning rule (2), $t_2 = 36 \in T_2$ with $t_2 - t_1 = 36 > d_{12} = 30$ can be removed. Similar pruning applies to $66 \in T_3$. Figure 6c shows the problem $\langle x, T\rangle$ after pruning.

5.4.5 Indexing optimal solutions

Consider the problem $\langle x, T\rangle$ in Fig. 6d, where each red rectangle denotes a $t_i \in T_i$, i.e., $T_1 = \{0\}, T_2 = \{30\}, T_3 = \{31, 60\}$. Suppose that the range query with intervals $[0, 0], [30, 30], [31, 60]$ on nodes 1, 2, 3, respectively, returns a solution $\tilde{x}'$ with $\tilde{x}_1' = 0, \tilde{x}_2' = 30, \tilde{x}_3' = 60$. Let $\langle \tilde{x}, \tilde{T}\rangle$, where $\tilde{x} = (0, 30, 66)$, be the problem attached to $\tilde{x}'$, i.e., the problem yielding $\tilde{x}'$ as the optimal solution. Each blue triangle denotes a $\tilde{t}_i \in \tilde{T}_i$, i.e., $\tilde{T}_1 = \{0\}, \tilde{T}_2 = \{30, 36\}, \tilde{T}_3 = \{31, 60, 66\}$. Since $T_i \subseteq \tilde{T}_i$ and $|t_i - x_i| = |t_i - \tilde{x}_i|$ for all $t_i \in T_i$, $\langle x, T\rangle$ is a subproblem of $\langle \tilde{x}, \tilde{T}\rangle$. Referring to $\tilde{x}_i' \in T_i, \forall i$, we directly return $\tilde{x}'$ as the optimal solution of $\langle x, T\rangle$.

Consider the problems $\langle x, T\rangle$ and $\langle \tilde{x}, \tilde{T}\rangle$ denoted by red rectangles and blue triangles, respectively, in Fig. 6e. Since $T_1 = \{0\}$ and $\tilde{T}_1 = \{6\}$, i.e., $T_1 \nsubseteq \tilde{T}_1$, $\langle x, T\rangle$ is not a subproblem of $\langle \tilde{x}, \tilde{T}\rangle$. However, $T_1 = \{0\}$ is a safe candidate set, with $(t_1 = 0, t_2 = 30) \vDash M_{12}$ for all candidates in $T_2 = \{30\}$, and similarly for T_3. Moreover, $\tilde{T}_1$ is also safe in $\tilde{T}$. That is, $\langle x, T\rangle$ is a safe-subproblem of $\langle \tilde{x}, \tilde{T}\rangle$.

For the optimal solution $\tilde{x}' = (6, 30, 60)$ of $\langle \tilde{x}, \tilde{T}\rangle$, it has $\tilde{x}_2', \tilde{x}_3'$ in the intervals of $[30, 30]$ and $[36, 60]$ on non-safe nodes 2 and 3, respectively, specified by the range query of $\langle x, T\rangle$. Indeed, we have $\tilde{x}_2' \in T_2, \tilde{x}_3' \in T_3$. Referring to Proposition 12, an optimal solution x' of $\langle x, T\rangle$ could be obtained by Eq. (8) with $x_1' = 0, x_2' = \tilde{x}_2' = 30, x_3' = \tilde{x}_3' = 60$.

6 Approximate algorithm

Although pruning and indexing techniques are developed, referring to the hardness in Theorem 1, the exact Algorithm 4 is still costly. In this section, we present two approximate solutions, including a simple heuristic method adapted from the exact algorithm (in Sect. 6.1) and another even more efficient method without candidate generation (in Sect. 6.2).

6.1 Simple heuristic repair

While the exact repairs (Algorithm 4) costly consider all possible branches, a simple heuristic approximation is to greedily consider only one branch (e.g., eliminating violations most) in each iteration. If it forms a feasible solution, as presented in Lines 19–20 in Algorithm 4, the program stops branching and directly returns this solution as the repair result.

As illustrated in Algorithm 4, the heuristic method still needs to take the candidates T as the input. Referring to the analysis at the end of Sect. 4.2, the maximum size of candidates is $2c^{n-1}(n-1)!$, where $n-1$ is the tight chain length and c is the maximum number of intervals labeling an edge in M. It motivates us to devise another approximation without the costly candidate generation step.

6.2 Linear programming approximation

Another intuitive idea of approximation is to relax the problem by ignoring the constraints that the repair takes only the values from the domain D of possible timestamps. Without such a constraint, we show how the repairing problem can be formulated as linear programming (LP) in Sect. 6.2.1.

6.2.1 Linear programming relaxation

Referring to the repair model introduced in Sect. 2.2, the timestamp repairing problem is to find a repair x' of x such that $x'_i \in D$ the repair takes values from the timestamp domain D, $x' \vDash M$ the temporal constraints M are satisfied, and $\Delta(x, x')$ the repair cost in Eq. (1) is minimized. To relax the problem, we propose to remove first the constraints $x'_i \in D$ that the repair must take values only from the timestamp domain D. According to the requirement of temporal constraints M, i.e., $x' \vDash M$, the relaxed repairing problem is given by

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} |x_i - x'_i| \\ \text{s.t.} \quad & x'_i - x'_j \le d_{ji}, && d_{ji} \in M \\ & x'_j - x'_i \le d_{ij}, && d_{ij} \in M \end{aligned} \tag{9}$$

where the variables x'_i in problem solving no longer require $x'_i \in D$.

To formulate the relaxed problem as linear programming (LP), we have to eliminate $|x_i - x'_i|$ the absolute difference between the original timestamp x_i and the repaired timestamp x'_i. Let

$$u_i = \frac{|x'_i - x_i| + (x'_i - x_i)}{2},$$
$$v_i = \frac{|x'_i - x_i| - (x'_i - x_i)}{2}.$$

We have $|x'_i - x_i| = u_i + v_i$ and $x'_i - x_i = u_i - v_i$. It follows the LP relaxation

$$\begin{aligned} \min \; & \textstyle\sum_{i=1}^{n} u_i + v_i \\ \text{s.t.} \; & v_j - u_j + u_i - v_i - x_j + x_i \le d_{ji}, \; d_{ji} \in M \\ & u_j - v_j + v_i - u_i - x_i + x_j \le d_{ij}, \; d_{ij} \in M \\ & u_i \ge 0, v_i \ge 0, \qquad\qquad 1 \le i \le n \end{aligned} \tag{10}$$

where u_i, v_i are variables in problem solving. The solution by LP relaxation is thus $x'_i = x_i + u_i - v_i$.

6.2.2 Rounding by transformation

Let $\hat{x}'$ denote the solution of LP in Eq. (10). To obtain a valid repair, we need to round $\hat{x}'$ into the domain D of possible timestamps.

A simple rounding is $\tilde{x}'_i = \arg\min_{t_i \in D} |\hat{x}'_i - t_i|$, i.e., simply assigning the timestamps in D that are closest to the LP solution as the repair. It is not surprising that violations to the temporal constraints M may be introduced again in this simple rounding. The generated $\tilde{x}'$ might not satisfy the temporal constraints M.

Fortunately, by calling Algorithm 1 of solution transformation again, $x' = \mathsf{Transform}(M, x, \tilde{x}')$, we can transform the infeasible $\tilde{x}'$ into a solution x'. Referring to Lines 5–10 and 14 in the transformation Algorithm 1, the returned repair x' always satisfies the temporal constraints M.

Proposition 13 *Given any x' that may violate the temporal constraints M, Algorithm 1 always outputs a repair x'' that satisfies M, $x'' \vDash M$. (Proof can be found in Appendix A.13.)*

The LP+transform approximation without candidate generation is more efficient than the heuristic approach. However, the Transform Algorithm 1 used for rounding the LP solution does not aim to minimize the repair cost. Therefore,

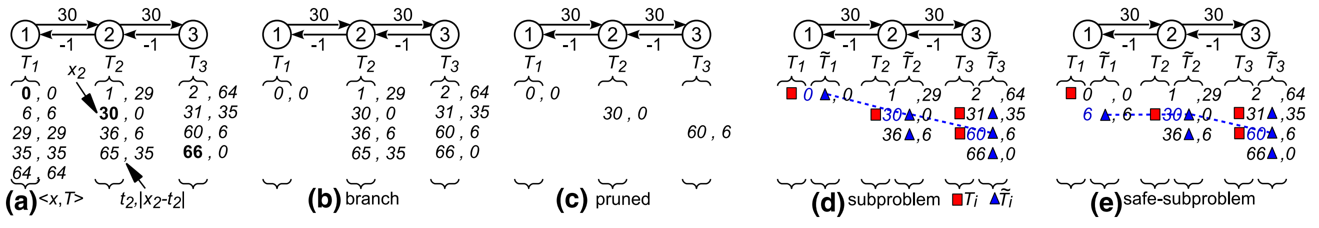


Fig. 6 Example process for repairing a tuple $x = (0, 30, 66)$ with candidates generated in the same way as in Fig. 5

as illustrated in Figs. 16 and 17, the LP+transform method may show lower accuracy than Heuristic in some experiments.

7 Streaming algorithm

The data in practice may have thousands of nodes, such as the smart home datasets introduced in Sect. 8 of experiments, which arrive in a stream. The repairing is often expected to be performed instantly, e.g., before storing the data in a database. Fortunately, the proposed technique can be naturally adapted to stream computing so that the repairing occurs in a window rather than the whole space of all nodes.

We extend our algorithms to support incremental computation in a streaming scenario, where events (data items) arrive in real time, as follows. Algorithm 5 provides the pseudo code for the stream processing. The sliding windows are for each node, where N_w denotes the nodes in the current window w, and N_{w+1} includes the nodes in the next window $w + 1$.

For candidate generation, the incremental computation is as follows. Given candidate sets T_w for nodes in N_w in the current window w, Algorithm 5 computes the new candidate sets T_{w+1} for nodes in N_{w+1} in the next sliding window $w+1$, denoted by $T_{w+1} := \mathsf{CandidateUpdate}(M, x, T_w, N_{w+1})$. It is an adaption in Algorithm 2 to support streaming setting. That is, for each candidate t_i recorded in T, we also keep its corresponding N_c the currently processed nodes, t the tuple of candidates over N_c, i the current ending (latest expanded) point of the chain, and "direction" the direction of the last edge (on i). First, all the candidates t_i can be removed whose $N_c \not\subseteq N_{w+1}$, i.e., no longer belonging to the current window $w + 1$. Second, rather than expanding the provenance paths from scratch, the expansion carries on from each previously generated candidate t_i recorded in T_w, by calling the Generate procedure in Algorithm 2.

For timestamp repairing, the adaption, e.g., in Algorithm 4, is simply to replace $k+1$ in Line 16 by $k-1$. Consequently, the branch and bound computation starts from the last node (data item) in the current window w, instead of the first one in the previous version of the algorithm. Similar to candidate generation, when calling $\mathsf{Repair}(M, T, x, x_{min}, k)$ which is previously processed, we may directly reuse the result without further branching. Pruning of candidates in Sect. 5.1 and indexing of optimal solutions in Sect. 5.2 can be naturally applied without any adaption needed.

Algorithm 5: $\mathsf{CandidateUpdate}(M, x, T_w, N_{w+1})$

```
Input: a minimal network $M$, a tuple $x$, the candidate sets $T_w$
       for nodes in $N_w$ in the window $w$, the set $N_{w+1}$ of nodes
       in the next sliding window $w + 1$ ,
Output: $T$ the new candidate sets for nodes in $N_{w+1}$ in the next
        sliding window $w + 1$ where each $T_i \in T$ is a set of
        candidate timestamps for variable $X_i$
1 $N = N_{w+1}$;
2 remove candidates $t_i$ from $T_w$ whose $N_c \not\subseteq N_{w+1}$;
3 $T := T_w$;
4 for each $t_i$ recorded in $T_w$ do
5     Generate ($N_c, t, i$, out);
6     Generate ($N_c, t, i$, in);
7 return $T$;
```

8 Experiment

In this section, we present the experimental evaluation, with particular focus on comparing our proposed methods to the existing approaches. All the implementation codes and data are available online [5].

Data Set We use a real dataset of event logs collected from the ERP systems of a train manufacturer . Temporal constraints are abstracted from the workflow specifications in the company. In total, there are 38 different workflow specifications, with the number of nodes (variables, analogous to number of attributes in a relation) ranging from 5 to 37, and 8612 event traces (tuples).

Moreover, we consider a series of large real-world sensor data sets. The climatology data [1] consists of 13.8 thousand tuples with up to 38 nodes. The biomedical science data [6] has 1 million tuples over up to 17 thousand nodes. The smart cities data [3] contains 27 thousand tuples over up to 37 nodes. The smart home data [4] has 15 thousand tuples over up to 10 thousand nodes. The IoT dataset [2] is with 1 million reading over up to 10 thousand nodes for streaming evaluation.

In addition, we also use a semi-synthetic big data set to control the experiment and the injection of faults. A log generation toolkit [20] is employed to generate up to 1 million tuples over 1 thousand nodes, using the real workflow specifications from the ERP systems of a train manufacturer (as introduced above).

Criteria Following the same line of evaluating data repairing [9], we inject faults in timestamps to evaluate the repairing methods. Let x_{truth} be the original correct timestamps of a tuple, x_{fault} be the error timestamps with injected faults, and x_{repair} be the repaired timestamps. We observe the accuracy measure of repairing [24], accuracy $= 1 - \frac{\Delta_{\text{error}}(x_{\text{repair}}, x_{\text{truth}})}{\Delta_{\text{cost}}(x_{\text{repair}}, x_{\text{fault}}) + \Delta_{\text{inject}}(x_{\text{truth}}, x_{\text{fault}})}$, where $\Delta_{\text{error}}(x_{\text{repair}}, x_{\text{truth}})$ is the error distance between true timestamps and repair results, $\Delta_{\text{cost}}(x_{\text{repair}}, x_{\text{fault}})$ is the distance cost paid in repair, and $\Delta_{\text{inject}}(x_{\text{truth}}, x_{\text{fault}})$ is the distance injected between true and fault timestamps.

All the distances are defined on absolute differences, i.e., the Δ distance function defined in Eq. (1). The accuracy measure takes $\Delta_{\text{cost}}(x_{\text{repair}}, x_{\text{fault}})$ into consideration, in order to normalize the measure, following the same line in [24]. That is, according to triangle inequality on distances, in the worst case, we have $\Delta_{\text{error}}(x_{\text{repair}}, x_{\text{truth}}) = \Delta_{\text{cost}}(x_{\text{repair}}, x_{\text{fault}}) + \Delta_{\text{inject}}(x_{\text{truth}}, x_{\text{fault}})$ with accuracy=0. For the best repair results, $\Delta_{\text{error}}(x_{\text{repair}}, x_{\text{truth}}) = 0$, we have accuracy=1.

8.1 Evaluation on candidate generation

This experiment evaluates the generation of candidates in Sect. 4. The experiment is performed on 10 workflow specifications which have 5 nodes/variables. The results are averages over 1750 tuples/traces. Figure 7 reports the average size of candidate timestamps generated for each node, the corresponding generation time cost, and the accuracy of repairing with such candidates. The x-axis considers various limits on the maximum lengths of provenance chains in generation.

As shown in Fig. 7a, by considering longer lengths of provenance chains, more candidates could be generated. The number of candidates does not increase fast, which illustrates the effectiveness of avoiding unnecessary candidates by provenance chains and pruning techniques in Sect. 4.

The time cost of candidate generation, in Fig. 7b, however, increases significantly. It is not surprising that, with more candidates, the corresponding time cost of repairing will be significantly higher as well (see more details in the following experiments).

Nevertheless, Fig. 7c illustrates that by considering provenance chains with length 1 or 2, the repair accuracy is already high, while further increasing the chain length leads to only a slight improvement in accuracy. The corresponding generation (as well as repairing) time cost for longer chains will be much higher as aforesaid.

Motivated by the result that a longer provenance chain has significantly higher time cost but little contribution in improving repair accuracy, we consider below the candidate generation with a provenance chain length 4.

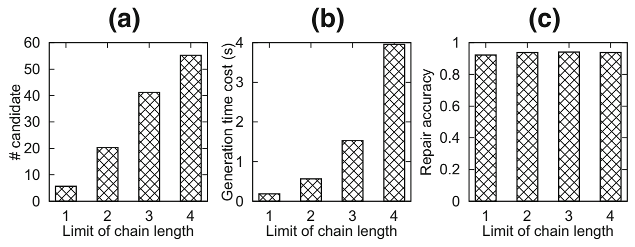


Fig. 7 Candidate generation with various lengths of provenance chains over the workflow dataset

8.2 Efficiency of proposed techniques

Next, we evaluate the performance of prune techniques in Sect. 5.1.2 for repairing. The experiment considers various numbers of tuples under the same temporal constraints. Therefore, only 50 tuples w.r.t. the same temporal constraint network (among 8000 tuples w.r.t. different temporal constraint networks) are considered. The size of each tuple is 5. A fault rate 0.3 is considered in the experiments, i.e., 30% events (nodes/variables) are injected with fault timestamps.

Figure 8 reports the time performance of our Baseline repair method in Sect. 5.1.1, Prune (without index) and Index+Prune (with both prune and index) in Algorithm 4. Results in different numbers of nodes (analogous to schema sizes in relational settings) and traces (tuples) are presented. It is clear to see the significantly reduced repair time cost by prune and index. In particular, Index+Prune shows about one order of magnitude improvement in time cost

8.3 Evaluation on indexing solutions

To compare the time costs for repairing with and without the index proposed in Sect. 5.2, we report the results under various budgets of index sizes, ranging from 0 (no index) to 1000 solutions, in Figs. 9 and 10. In particular, a budget 0 of index size means repairing without index.

Figure 9 evaluates the utility of index by observing the hit rate, i.e., the times of a solution being returned by index divided by the total number of the procedure Repair being called. It is not surprising that the larger the budget of index is,

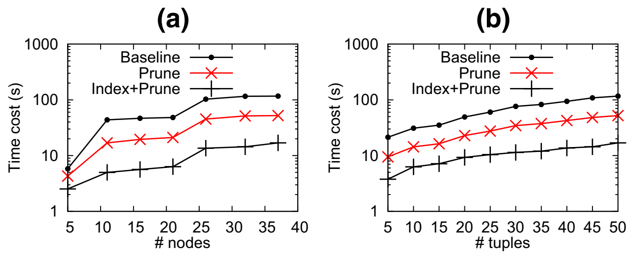


Fig. 8 Repairing with pruning over the workflow dataset

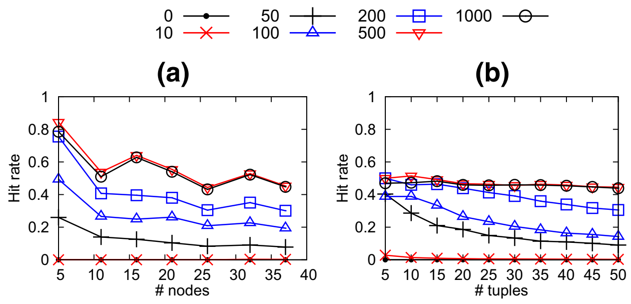


Fig. 9 Hit rate of index in repairing over the workflow dataset

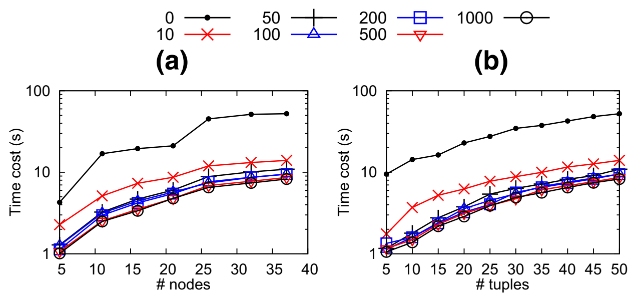


Fig. 10 Repairing with various index size budgets over the workflow dataset

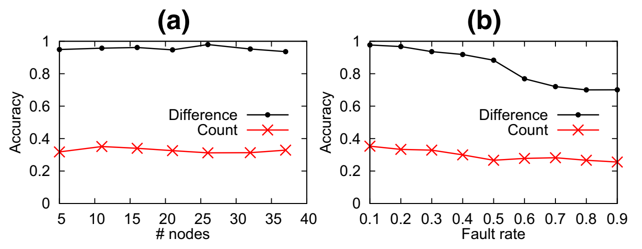


Fig. 11 Evaluation on repair cost functions over the workflow dataset

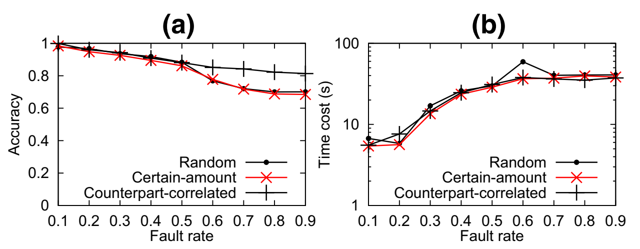


Fig. 12 Evaluation on various error cases over the workflow dataset

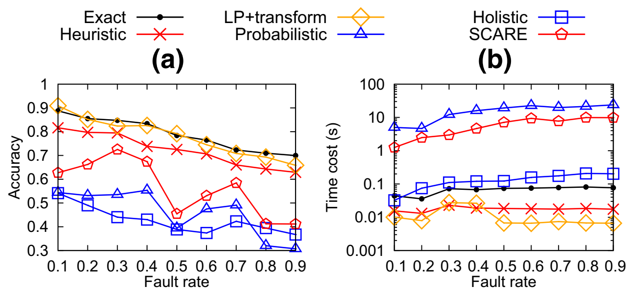


Fig. 13 Comparison on various fault rates over the semi-synthetic dataset

the higher the hit rate will be. However, by further increasing the budget, e.g., from 500 to 1000, the improvement of hit rate slows down. On the other hand, as shown in Fig. 9b, given a small budget (say 50), since the index is already full after processing the first 5 tuples, no more solutions could be indexed when processing the following tuples and the hit rate drops. Nevertheless, a large budget like 500 can still keep the hit rate high.

Consequently, in Fig. 10, the improvement of time cost is not significant by further increasing the index budget from 500 to 1000. Indeed, owing to the cost of accessing large size index, the improvement from 50 to 100 is already not as significant as that from 10 to 50. Nevertheless, the indexing technique shows about one order of magnitude improvement in time cost compared to that of no index (budget 0).

8.4 Evaluation on cost functions

Besides the absolute difference-based repair cost metric in Eq. (1), other metrics, such as counting the number of changed timestamps, could also be applied. Figure 11 compares the results by using the absolute difference-based and count-based repair cost metrics. As shown, the repair accuracy with the absolute difference-based cost function shows higher repair accuracy than the count-based. The reason is that, compared to the count-based metric, the absolute difference-based cost can capture more precisely the "amount" information of the data deviations, and thus achieve better the minimum change goal.

8.5 Evaluation on various error cases

This experiment considers several representative cases of timestamp errors that often occur in practice: (1) Random errors, which take random values from the timestamp domain. (2) Certain amount errors, such that all timestamps being off by a certain amount in some sources. (3) Counterpart correlated errors, where faulty timestamp values are partially correlated with their correct counterparts through a normal distribution-based fault model, $\mathcal{N}(\mu, \sigma^2)$. μ denotes the correct counterpart (true timestamp) of an event, and σ^2 is variance. That is, faulty timestamp values are partially correlated with their correct counterparts μ.

Figure 12 reports the results of Exact repairing on various error cases. Generally, the accuracy drops with the increase of fault rate. Random errors and certain amount errors show very similar performance, which illustrates the robustness of proposed methods. The accuracy of counterpart correlated errors is a bit higher, especially when the fault rate is large, as illustrated in Fig. 12a. The result is not surprising given that the faulty timestamp values are partially correlated with their correct counterparts. Time costs under various error cases are generally similar.

8.6 Comparison to existing methods

This experiment compares our proposal with the repairing methods.

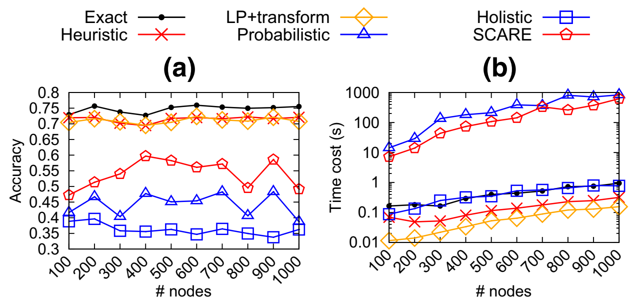


Fig. 14 Comparison on various numbers of nodes over the semi-synthetic dataset

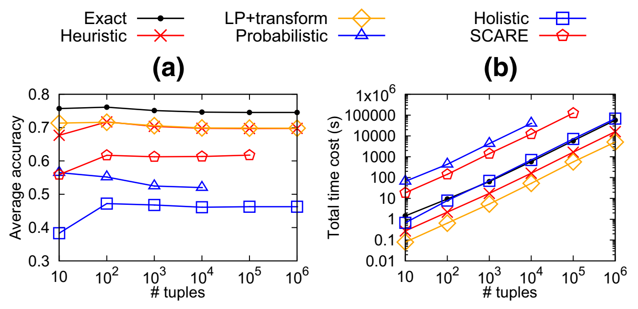


Fig. 15 Comparison on various numbers of tuples over the semi-synthetic dataset

1. For our proposed Exact repairing, we use the most advanced Algorithm 4 with prune and index techniques.
2. Heuristic approximation (in Sect. 6.1, with termination in Line 20 in Algorithm 4) is also evaluated.
3. LP+transform is another approximation method with LP relaxation and rounding by solution transformation in Algorithm 1 as presented in Sect. 6.2.
4. The Probabilistic approach [22] studies the distribution of timestamps and uses Bayesian Network to determine repair values.
5. The Holistic method [13] greedily repairs data (timestamps) in violations to the given denial constraints [12]. By representing temporal constraints as denial constraints, this repairing method is applicable to timestamp repairing.
6. The ML-based repairing method SCARE [27] considers the repair likelihood defined on values w.r.t. functional dependencies. To adapt the maximum likelihood principle in timestamp repairing, we study the likelihood over timestamps. Repairs are then generated upon the maximum likelihood instead of the minimum change.

We first report the results over the semi-synthetic big data set, under various settings. Figure 13 reports the results with various fault rates, e.g., a fault rate 0.3 denotes that 30% events (nodes/variables) are injected with fault timestamps. It is not surprising that the accuracy drops with the increase

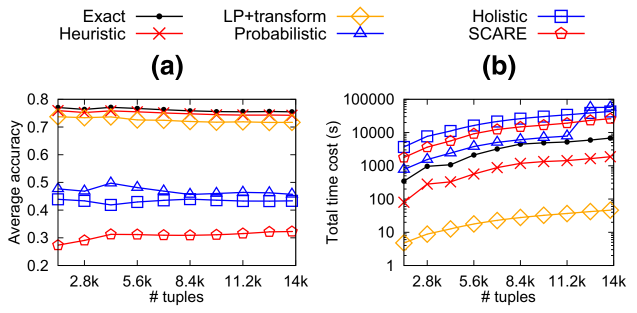


Fig. 16 Comparison on various numbers of tuples over the climatology dataset

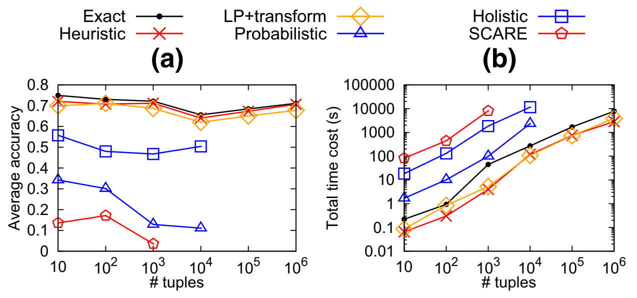


Fig. 17 Comparison on various numbers of tuples over the biomedical science dataset

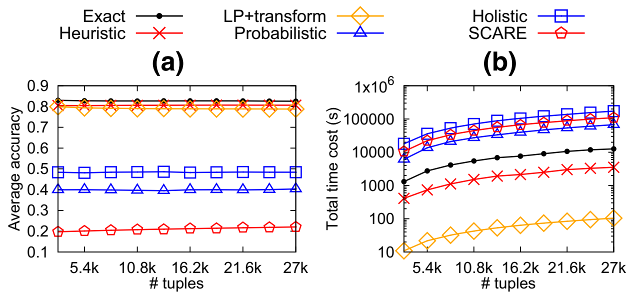


Fig. 18 Comparison on various numbers of tuples over the smart cities dataset

of fault rate. In Fig. 14, the time cost increases heavily with the number of nodes, while the accuracy is relatively stable. Figure 15 shows again that all the methods have stable accuracy but linear time cost w.r.t. the number of tuples.

Moreover, we conduct the experiments over large real-world sensor data sets. Figure 16 evaluates on the climatology data [1]. Figure 17 considers the biomedical science data [6]. Figure 18 is for the smart cities data [3]. Figure 19 presents the smart home data [4]. The performances are generally similar to that over the semi-synthetic data in Fig. 15. Some results of the existing methods are not reported in Figs. 15 and 17, owing to the extremely high costs over the large semi-synthetic and biomedical science datasets.

Our Exact repair always shows the highest repair accuracy in all the tests. Remarkably, its corresponding time cost is surprisingly lower than that of Probabilistic or Holistic. The

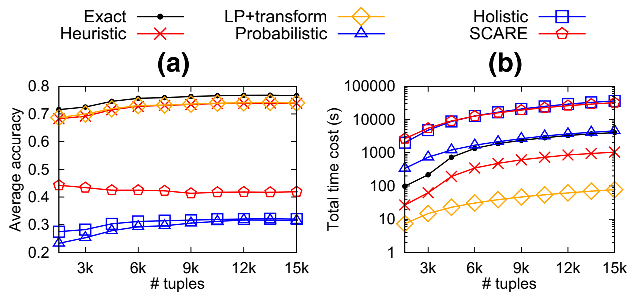


Fig. 19 Comparison on various numbers of tuples over the smart home dataset

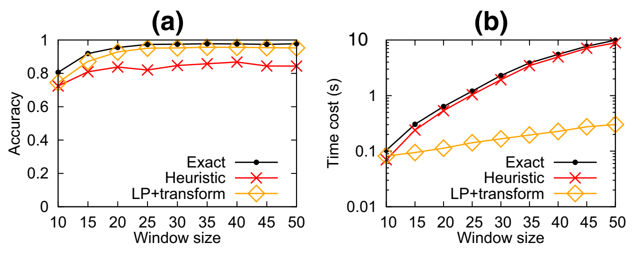


Fig. 20 Repairing timestamps with various sliding windows sizes over the streaming semi-synthetic dataset

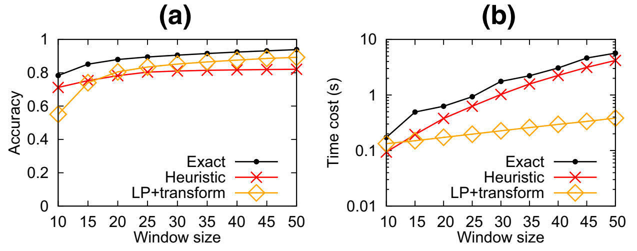


Fig. 21 Repairing timestamps with various sliding windows sizes over the streaming smart home dataset

reason is that Probabilistic employs the high cost Bayesian Network inference, while the greedy repair in Holistic could be trapped in local optima and evokes multiple rounds of repairing.

The accuracy of Heuristic approach is not as high as Exact, whereas its time cost is significantly lower than Exact. The LP+transform method is comparable in accuracy with Heuristic, and has even lower time cost. The reason is that Heuristic still needs to generate a large number of repair candidates, which is not necessary in LP+transform.

The Probabilistic approach has lower accuracy and much higher time cost, compared to our proposal (Exact and Heuristic). The reason is, as discussed in Sect. 9, the Probabilistic repairing heavily relies on obtaining a right order of events (nodes) in the first step, and the second step of inference over Bayesian Network is very costly.

The Holistic method also shows lower accuracy but higher time cost, compared to our Exact approach. As discussed (in Sect. 9 as well), the greedy repair may be trapped in local optima and cannot guarantee to eliminate all the violations. A large number of repair iterations need to be performed.

The SCARE approach has various performances in different datasets. The reason is that it needs to specify a set of reliable nodes where no errors present. In practice, however, errors could occur in any node. The statistical-based SCARE (as well as Probabilistic) shows unstable repair accuracy in Figs. 13 and 14, since the statistical distributions could vary under different fault rates and numbers of nodes.

The higher repair accuracy of our proposed methods compared to the statistical-based approaches (that do not follow the minimum modification principle) verifies the rationale of minimizing changes in timestamp repairing. Repairing should be conducted by avoiding losing information of the original data.

8.7 Evaluation in streaming setting

To evaluate the incremental computation in a streaming setting, similar to Sect. 8.6, we use again the semi-synthetic dataset and another real dataset smart home [4]. The experiment is performed by sliding windows from node 1 to n in tuples.

Figure 20 reports the repair accuracy and the corresponding time cost under various window sizes. By varying the window size, it is not surprising that a larger window size achieves higher accuracy in Fig. 20a, since more nodes are considered in the context. However, more time cost is required to deal with the nodes in a larger sliding window as well, which is indicated in Fig. 20b.

Similar results are also observed in the real dataset smart home [4] in Fig. 21. Again, the time cost of LP+transform increases slower than those of Exact and Heuristic, since they have to generate a large number of candidates (though in an incremental manner as presented in Algorithm 5). The LP+transform approximation shows better performance than the simple Heuristic, especially given a larger window size.

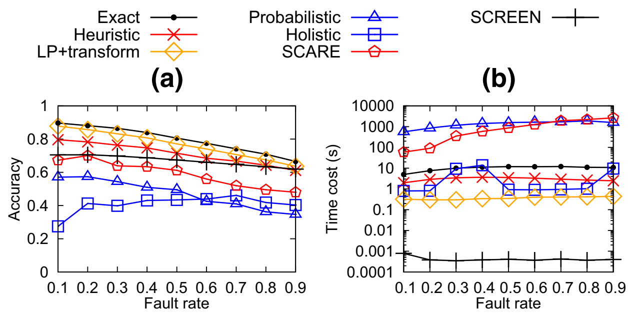


Fig. 22 Repairing values (rather than timestamps) with various fault rates over the walking dataset

8.8 Applicability beyond timestamps

It is remarkable that the proposed repairing could also be applied to other finite, partially ordered sequences of data, as long as the corresponding constraints can be represented in the form of minimal networks. To demonstrate the general applicability and the practical value of our proposal, we manually collect a dataset of GPS readings, by carrying a GPS device and walking around the campus. The maximum walking speed of a person thus serves as the hard limits.

Figure 22 reports the results over various fault rates. The results are generally similar to Fig. 13 on repairing timestamps. That is, with the increase of fault rate, the repair accuracy drops. Our proposed Exact algorithm can achieve higher accuracy and lower time costs, compared to the existing Probabilistic and SCARE approaches. The performance of Holistic is not stable, since it may be trapped in local optima and evokes multiple rounds of repairing as aforesaid. The results demonstrate the general applicability and practical value of our proposal in the fields beyond timestamps.

Moreover, we conduct another experiment on a larger IoT data with 1 million readings over up to 10 thousand nodes by comparing with the existing method SCREEN [24], which is dedicated to cleaning big sensor data streams. As shown in Fig. 23, while the SCREEN method is efficient, our proposal achieves a higher repairing accuracy. The reason is that SCREEN uses coarse grained speed constraints on all data points, while we employ fine grained constraints over different data pairs. With more precise constraint, our proposal achieves a higher accuracy. The corresponding time cost of SCREEN is lower, which is not surprising given the coarse grained yet simple speed constraints.

9 Previous work

Owing to the distinct difference between temporal constraints and integrity constraints, most existing data repairing techniques (such as [9] based on functional dependencies) are not directly applicable to repairing timestamps.

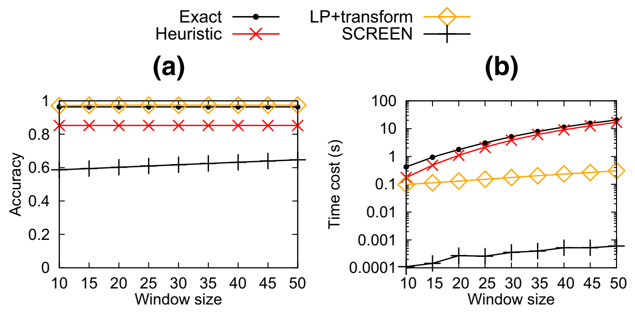


Fig. 23 Repairing values (rather than timestamps) with various sliding windows sizes over the streaming IoT dataset

Holistic repair [13] can support repairing w.r.t. temporal constraints, by expressing them as denial constraints [12]. It greedily modifies values (timestamps) to eliminate the currently observed violations. This greedy modification may introduce new violations to other data points, and thus evokes multiple rounds of repairing. Moreover, the greedy repair could be trapped in local optima, and cannot eliminate all the violations. It is worth noting that assigning fresh variables outside the currently known timestamp domain does not help in eliminating violations of temporal constraints.

To the best of our knowledge, the only existing work dedicated to repairing timestamps is [22]. Unlike the holistic cleaning in a constraint-based approach, the repairing in [22] consists of two steps: (1) repairing the order of data points (since the imprecise timestamps may lead to out-of-order arrival), and (2) then adapting the timestamps. It is worth noting that if an erroneous order of data points is returned in the first step, the timestamps would never be repaired correctly.

Instead of repairing the imprecise timestamps, Zhang et al. [28] handle the imprecise timestamps in a different setting. A range of possible timestamps is assumed to be given for each event, together with a probabilistic distribution of the possible timestamps. The study [28] thus focuses on performing analyses directly over the uncertain timestamps. In our scenario, we do not have such a given range and distribution of possible timestamps. In this sense, our proposal of timestamp repairs is not directly comparable to [28].

10 Conclusion

This study proposes to repair timestamps that do not conform to *temporal constraints*. The timestamp repairing is manipulated under the minimum change principle, widely considered in data repairing [9]. To find the optimal minimum repair over the various combinations of possible timestamps, we notice that any optimal repair solution can be transformed to a special form, such that each changed node (in repairing) is connected to some unchanged one via a tight/provenance chain (Corollary 5). A finite set of promising candidates are thus generated upon the chains and unchanged timestamps, where an optimal repair can always be found (Proposition 7). We devise (1) an exact algorithm for computing the optimal repair from the generated candidates, (2) a heuristic approximation by greedily selecting repairs from the candidates, (3) a solution indexing scheme for reusing the solutions of problems, (4) the extension to streaming repairing, and (5) a LP relaxation with rounding by solution transformation. Extensive experiments over real datasets demonstrate that our exact method has the highest repair accuracy compared to the state-of-the-art approaches as well as the proposed approximation methods. The simple heuristic approximation shows lower time costs than the exact approach, but still needs the costly candidate generation. The LP+transform approximation without candidate generation is thus more efficient than the heuristic approach while still having comparable repair accuracy.

While the temporal constraints in the form of minimal networks are prevalent and can capture many realistic scenarios, additional (more complicated) temporal constraints could also be declared. For example, event patterns in complex event processing [15] may be used as temporal constraints as well. The sequential semantics in event patterns, i.e., an event occurs after another with time difference at least a but within b, is exactly the temporal constraints in the form of minimal networks considered in this paper. In addition to the pairwise constraints on two events, the event patterns could further declare the constraints on multiple events, i.e., several events could occur in any order but their time differences should be at least a but no greater than b. Note that with minimal networks on event pairs, the timestamp repairing problem is already hard (Theorem 1). Given the more complicated temporal constraints on multiple events, the cleaning is highly nontrivial. We leave this problem as future work.

Acknowledgements This work is supported in part by the National Key Research and Development Plan (2019YFB1705301), the National Natural Science Foundation of China (62072265, 61572272, 71690231), and the MIIT High Quality Development Program 2020.

A Proofs

A.1 Proof of Theorem 1

To prove the NP-hardness of the repairing problem, we build a reduction from the 3-coloring problem, which is known to be NP-complete [21]. Given a connected graph $G = (V, E)$, the 3-coloring problem is to determine whether there is a way of coloring the vertices in graph G such that no two adjacent vertices are of the same color, using at most 3 different colors.

Each vertex $v_i \in V$ corresponds to a variable X_i. Its assigned color $C(v_i)$ maps to the assignment of X_i. Let $D = \{1, 2, 3, 6\}$ be the timestamp domain, where the values 1, 2 and 3 stand for the three admissible colors in the coloring problem, and the value 6 is for the initial assignment. That is, we set $x_i = 6$ for all i at the beginning. For each edge $(v_i, v_j) \in E$, we associate a constraint S_{ij} with multiple intervals {[-2,-2], [-1,-1], [1,1], [2,2]}. It restricts X_i and X_j to have different values, i.e., restricting the two adjacent vertices v_i and v_j to have different colors.

We show in the following that the tuple x of assignment has a repair x' with cost $\Delta(x, x') \geq 3n$ that satisfies all the constraints iff the graph G is 3-colorable.

First, let C be a feasible 3-coloring solution. For each edge $(v_i, v_j) \in E$, recall that v_i and v_j should not be the same color, and $C(v_i), C(v_j) \in \{1, 2, 3\}$, which stand for the three admissible colors. We consider a repair $x'_i = C(v_i)$ for all i. The difference between $C(v_i)$ and $C(v_j)$, i.e., $x'_i - x'_j$ (or $x'_j - x'_i$) will always satisfy the constraint {[-2,-2], [-1,-1], [1,1], [2,2]}. That is, we get a repair x' with cost $\Delta(x, x') \geq 3n$ that satisfies all the temporary constraints.

Conversely, suppose that there exists a feasible repair x' with cost $\Delta(x, x') \geq 3n$. Apparently, we have $x'_i \neq 6$ for all i, since any $x'_i = 6$ will definitely violate the constraints. For each edge $(v_i, v_j) \in E$, since $x'_i - x'_j \neq 0$ and x'_i, x'_j can only take values from $\{1, 2, 3\}$, the two adjacent vertices v_i and v_j do not share the same color. Thereby, we have a proper 3-coloring solution $C(v_i) = x'_i$ for graph G.

A.2 Proof of Proposition 2

To prove this proposition, we consider two aspects:

First, $\Delta(x, x'') \leq \Delta(x, x')$ is ensured. By decreasing the assignment for $|N_p| \geq |N_q|$ (and similarly, increasing for $|N_p| < |N_q|$), the repairing cost is non-increasing in each step, as illustrated in Eq. (2).

Second, by moving changed nodes to N_u, the conclusion is proved. In particular, a node i is added into N_m if there is a tight edge $i \to j$ or $j \to i$ for some $j \in N_m$. Moreover, nodes in N_m are moved to N_u, if either they are connected to some node in N_u or some node in N_m itself become unchanged.

A.3 Proof of Proposition 3

To illustrate the correctness of Algorithm 1, we consider the following aspects.

First, x'' is a feasible solution. The bound η ensures that each being modified assignment will not exceed the constraints specified by d_{jk} in Eq. (3) or equivalently Lines 24 and 28 in Algorithm 1.

Second, $\Delta(x, x'') \leq \Delta(x, x')$ is ensured. By decreasing the assignment for $|N_p| \geq |N_q|$ (and similarly, increasing for $|N_p| < |N_q|$), the repairing cost is non-increasing in each step, as illustrated in Eq. (2).

Third, the connectivity w.r.t. tight chain is obvious by seeing that nodes in N_m are connected by tight edges. In particular, a node i is added into N_m if there is a tight edge $i \to j$ or $j \to i$ for some $j \in N_m$, according to Line 13. Moreover, nodes in N_m are moved to N_u, if either they are connected to some node in N_u (Line 16) or some node in N_m itself become unchanged (Line 19).

Finally, to show the termination of the algorithm, we can see that after each step of modification (Lines 23–30), either some node become unchanged by variation θ (with at least one node moved to N_u) or some node reaches the bound by variation η. For the latter case, at least one node is moved from N_v to N_m or from N_m to N_u.

A.4 Proof of Lemma 4

Solely reducing x_i^* without modifying the corresponding x_j^* is forbidden. Otherwise, it leads to another solution with lower repairing cost, which is contradictory to the optimality of x^* with the minimum repairing cost. In other words, there must exist some j such that $x_j^* - x_i^* = d_{ij}$.

A.5 Proof of Corollary 5

Referring to Proposition 3, the conclusion is obvious by conducting $\mathsf{Transform}(M, x, x')$ for any optimal solution x'. It returns another optimal solution x^* with the same optimal repairing cost and connecting changed nodes to unchanged ones via tight edges (chains).

A.6 Proof of Lemma 6

Since edges $i \rightarrow j$ and $j \rightarrow k$ are tight, we have $x'_k - x'_i = d_{ij} + d_{jk}$. Referring to the temporal constraints, it follows $d_{ij} + d_{jk} = x'_k - x'_i \le d_{ik}$. According to the shortest paths in defining the minimal network M, we have $d_{ik} \le d_{ij} + d_{jk}$. The conclusion is a direct consequence.

A.7 Proof of Proposition 7

For any tight edges, $k_{y-1} \rightarrow k_y, k_y \rightarrow k_{y+1}$, in a tight chain that makes it not a provenance chain, according to the transitivity in Lemma 6, there must exist a tight edge $k_{y-1} \rightarrow k_{y+1}$. In other words, the node k_y can be removed from the tight chain.

Similar conclusion applies to the case of $k_{y-1} \leftarrow k_y, k_y \leftarrow k_{y+1}$. By removing all the aforesaid k_y, the chain becomes a provenance chain.

A.8 Proof of Proposition 8

We prune the unused temporal constraints by comparing all the pairs of candidates across two nodes, where the maximum size of candidates of a node is a. Comparing all the pairs of candidates across two nodes needs $O(a^2)$ comparisons, and we have n^2 node pairs, so the whole time complexity is $O(a^2 n^2)$.

A.9 Proof of Proposition 9

Referring to the branch and bound computation, the repairing procedure at most try all the combinations of the candidates of n nodes, where the maximum size of candidates of a node is a. The time complexity of Algorithm 4 is $O(a^n)$.

A.10 Proof of Proposition 10

First, given $T_i \subseteq \tilde{T}_i, \forall i$, it is obvious to see that all the solutions of subproblem $\langle x, T \rangle$ are also the solutions of $\langle \tilde{x}, \tilde{T} \rangle$. On the other hand, $\tilde{x}'_i \in T_i, \forall i$, indicates that $\tilde{x}'$ is a solution of $\langle x, T \rangle$ as well. If there exists another solution x^* with lower cost for $\langle x, T \rangle$, it contradicts the optimality of $\tilde{x}'$ to $\langle \tilde{x}, \tilde{T} \rangle$.

A.11 Proof of Lemma 11

The correctness is easy to see according to the candidate prune rule (2) in Sect. 5.1.2. It eliminates all the candidates in violation to $T_i = \{t_i\}$, i.e., all the $t_j \in T_j$ such that $(t_i, t_j) \nvDash M_{ij}$.

A.12 Proof of Proposition 12

Assume that x' is not an optimal solution of $\langle x, T \rangle$, i.e., exists a x^* with $\Delta(x, x^*) < \Delta(x, x')$. According to Eq. (8), for any safe $T_i \in T$, x'_i is the one with the lowest cost, i.e., $|x_i^* - x_i| \ge |x'_i - x_i|$. Since x' is not optimal, there must exists a non-safe T_j such that $|x_j^* - x_j| < |x'_j - x_j|$.

We construct a $\tilde{x}''$ for $\langle \tilde{x}, \tilde{T} \rangle$, where $\tilde{x}''_i = \tilde{x}'_i$ if T_i is safe; otherwise, $\tilde{x}''_j = x_j^*$ for non-safe T_j. The safe-subproblem definition requires $\tilde{T}_i$ to be safe for each safe T_i, i.e., $\tilde{x}''$ forms a feasible solution. It follows $\Delta(\tilde{x}, \tilde{x}'') < \Delta(\tilde{x}, \tilde{x}')$, referring to $|\tilde{x}''_j - \tilde{x}_j| = |x_j^* - x_j| < |x'_j - x_j| = |\tilde{x}'_j - \tilde{x}_j|$ for some non-safe T_j. In other words, $\tilde{x}''$ is a solution of $\langle \tilde{x}, \tilde{T} \rangle$ with cost lower than $\tilde{x}'$, which is a contradiction.

A.13 Proof of Proposition 13

First, as illustrated in Lines 4–10 in Algorithm 1, when moving a node i from N_v to N_m, we check whether it will introduce violations to the existing nodes in $N_v \cup N_u$ w.r.t. temporal constraints M. That is, a modification is made on x'_i to ensure its $\alpha_i \le 0$ and $\beta_i \le 0$, where $\alpha_i = \max_{k \in N_v \cup N_u, d_{ik} \in M} x'_k - x'_i - d_{ik}$ and $\beta_i = \max_{k \in N_v \cup N_u, d_{ki} \in M} x'_i - x'_k - d_{ki}$. Similarly, Line 14 guarantees no violation when moving node i from N_v to N_m.

Moreover, as presented in the proof of Proposition 3, with the bound η, the modification in Lines 24 and 28 will not introduce violations to the temporal constraints M either. To sum up, Algorithm 1 always returns a feasible solution that satisfies the temporal constraints M, no matter whether the input x' has violation or not.

References

1. http://ampds.org/
2. http://db.csail.mit.edu/labdata/labdata.html
3. http://iot.ee.surrey.ac.uk:8080/datasets.html
4. https://archive.ics.uci.edu/ml/datasets/gas+sensors+for+home+activity+monitoring
5. https://github.com/rui-hrh/timestamp
6. https://physionet.org/data/
7. Barga, R.S., Goldstein, J., Ali, M.H., Hong, M.: Consistent streaming through time: a vision for event stream processing. In: CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, 7–10 Jan 2007, Online Proceedings, pp. 363–374 (2007)
8. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (1975)
9. Bohannon, P., Flaster, M., Fan, W., Rastogi, R.: A cost-based model and effective heuristic for repairing constraints by value modification. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, 14–16 June 2005, pp. 143–154 (2005)
10. Cheng, D., Bahadori, M.T., Liu, Y.: FBLG: a simple and effective approach for temporal dependence discovery from time series data. In: Macskassy, S.A., Perlich, C., Leskovec, J., Wang, W., Ghani, R. (eds.) The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA, 24–27 Aug 2014, pp. 382–391. ACM (2014)
11. Chomicki, J., Marcinkowski, J.: On the computational complexity of minimal-change integrity maintenance in relational databases. In: Inconsistency Tolerance [Result from a Dagstuhl Seminar], pp. 119–150 (2005)
12. Chu, X., Ilyas, I.F., Papotti, P.: Discovering denial constraints. PVLDB **6**(13), 1498–1509 (2013)
13. Chu, X., Ilyas, I.F., Papotti, P.: Holistic data cleaning: putting violations into context. In: 29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, 8–12 April 2013, pp. 458–469 (2013)
14. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artif. Intell. **49**(1–3), 61–95 (1991)
15. Ding, L., Chen, S., Rundensteiner, E.A., Tatemura, J., Hsiung, W., Candan, K.S.: Runtime semantic query optimization for event stream processing. In: Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, 7–12 April 2008, Cancún, Mexico, pp. 676–685 (2008)
16. Duan, L., Pang, T., Nummenmaa, J., Zuo, J., Zhang, P., Tang, C.: Bus-OLAP: a data management model for non-on-time events query over bus journey data. Data Sci. Eng. **3**(1), 52–67 (2018)
17. Dyreson, C.E., Snodgrass, R.T.: Supporting valid-time indeterminacy. ACM Trans. Database Syst. **23**(1), 1–57 (1998)
18. Fan, W.: Dependencies revisited for improving data quality. In: Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, 9–11 June 2008, Vancouver, BC, Canada, pp. 159–170 (2008)
19. Fan, W.: Constraint-driven database repair, 2nd edn. In: Encyclopedia of Database Systems (2018)
20. Jin, T., Wang, J., Wen, L.: Efficiently querying business process models with beehivez. In: Proceedings of the Demo Track of the Ninth Conference on Business Process Management 2011, Clermont-Ferrand, France, August 31st, 2011 (2011)
21. Karp, R.M.: Reducibility among combinatorial problems. In: Proceedings of a symposium on the Complexity of Computer Computations, Held 20–22 March 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA, pp. 85–103 (1972)
22. Rogge-Solti, A., Mans, R., van der Aalst, W.M.P., Weske, M.: Improving documentation by repairing event logs. In: The Practice of Enterprise Modeling—6th IFIP WG 8.1 Working Conference, PoEM 2013, Riga, Latvia, 6–7 Nov 2013, Proceedings, pp. 129–144 (2013)
23. Song, S., Cao, Y., Wang, J.: Cleaning timestamps with temporal constraints. PVLDB **9**(10), 708–719 (2016)
24. Song, S., Zhang, A., Wang, J., Yu, P.S.: SCREEN: stream data cleaning under speed constraints. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31–June 4, 2015, pp. 827–841 (2015)
25. Sun, P., Liu, Z., Davidson, S.B., Chen, Y.: Detecting and resolving unsound workflow views for correct provenance analysis. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29–July 2, 2009, pp. 549–562 (2009)
26. Tang, L., Li, T., Shwartz, L.: Discovering lag intervals for temporal dependencies. In: The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, 12–16 Aug 2012, pp. 633–641 (2012)
27. Yakout, M., Berti-Équille, L., Elmagarmid, A.K.: Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, 22–27 June 2013, pp. 553–564 (2013)
28. Zhang, H., Diao, Y., Immerman, N.: Recognizing patterns in streams with imprecise timestamps. PVLDB **3**(1), 244–255 (2010)