Cleaning Timestamps with Temporal Constraints

Shaoxu Song Yue Cao Jianmin Wang
Tsinghua National Laboratory for Information Science and Technology
KLiss, MoE; School of Software, Tsinghua University, China
{sxsong, caoyue10, jimwang}@tsinghua.edu.cn

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. Experiments on real datasets demonstrate the superiority of our proposal compared to the state-of-the-art approaches.

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 [3]. Cleaning the imprecise timestamps is necessary for reliable applications, such as provenance analysis [15], identifying the sequence of steps leading to a data value, or complex event processing (CEP) [9], returning the occurrences of requested event patterns.

Constraints are essential in evaluating the correctness of data, such as integrity constraints for relational data [11]. Regarding temporal data, we employ *temporal networks* [8], specifying temporal constraints on timestamp differences between nodes/variables (see examples below). The aforesaid imprecise timestamps could be identified as violations of the temporal constraints.

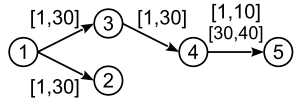
Inspired by data repairing w.r.t. integrity constraints [4], the timestamp repairing proposes to modify the timestamps towards satisfaction of the given temporal constraints.

Event	Timestamp
1 (Submit)	09:05
2 (Normalize)	**23:53**
3 (Proofread)	09:25
4 (Examine)	09:48
5 (Authorize)	09:54

(a) Trace σ_1

Event	Timestamp
3 (Proofread)	**14:__**
1 (Submit)	14:16
2 (Normalize)	14:35
4 (Examine)	14:52
5 (Authorize)	15:29

(b) Trace σ_2



(c) Temporal constraint network

Figure 1: Event logs with temporal constraints

Example 1. Figure 1 presents some example segments of real event logs in the ERP system of a train manufacturer (see Section 6 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 Figure 1(a) 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 Figure 1(b) is the trace of designing another part.

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:__.

The events are obviously not occurring in random, but constrained by certain workflow discipline. Figure 1(c) 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 Figure 1(c), indicates the minimum and maximum restrictions on the distance (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

Proceedings of the VLDB Endowment, Vol. 9, No. 10

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 Figure 1(c), [1,10][30,40] on edge $4 \to 5$ 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. Such temporal constraints can either be extracted from workflow/process specifications, or obtained from data [16].

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:__ − 14:16 = −16 are identified as inconsistent timestamps as well.

To address the imprecise timestamps, similar to other constraint-based repairing [7], the temporal constraint network is given in advance as inputs. Although only five events are presented in this example, 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 [4], 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.

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.

Example 2 (Example 1 continued)**.** 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 $1 \to 2$). 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. Indeed, such a minimum repair does help in applications as illustrated in Section 6.4. ■

Preliminary studies [10, 19] handle the imprecise timestamps by an uncertainty model on possible timestamps. The probabilistic-based repairing is thus performed via probabilistic inference in Bayesian Networks [13]. A key issue of this method (as analyzed in Section 7 of related work and evaluated in Section 6 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, we may model the temporal constraints as integrity constraints (e.g., denial constraints [6]), and employ existing constraint-based data repairing methods [7]. Unfortunately, according to our analysis (in Section 7 as well as in the experimental evaluation), the soundness w.r.t. satisfaction of temporal constraints is not guaranteed due to the greedy computation.

Contributions. Our main contributions are summarized as

(1) We propose a solution transformation paradigm, in Section 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 Section 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 exact, heuristic and randomized algorithms for repairing timestamps, in Section 5. Unlike the constraint-based greedy repairing [7], satisfaction of temporal constraints (soundness) is guaranteed in our results. Advanced *pruning* techniques are developed, in Section 5.

(4) We report an extensive experimental evaluation over a real dataset, in Section 6. The results demonstrate that our proposed methods show significantly higher repair accuracy compared to the state-of-the-art approaches, including probabilistic-based [13] and constraint-based [7], while time cost of our proposal is lower (see Section 6.3 for details). In particular, the higher repair accuracy compared to the probabilistic and randomized methods (that do not strictly follow the minimum modification principle) verifies the rationale of minimizing changes in timestamp/data repairing.

The remaining of this paper is organized as follows. In Section 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 Section 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 Section 4, and find the optimal solution from the candidates in Section 5.

2. PRELIMINARIES

Temporal Constraints. Consider a set of variables, $X_1, \ldots, X_n$. Each variable X_i represents a time point of an event i, taking values from a domain D of possible timestamps.

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 $i \to j$ $(i, j \in N)$ indicates that a constraint S_{ij} is specified. Each constraint S_{ij} specifies a single interval, $[a_{ij}, b_{ij}]$, to constraint 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 called a *solution* if the assignment $\{X_1 = x_1, \ldots, X_n = x_n\}$ satisfies all the constraints.

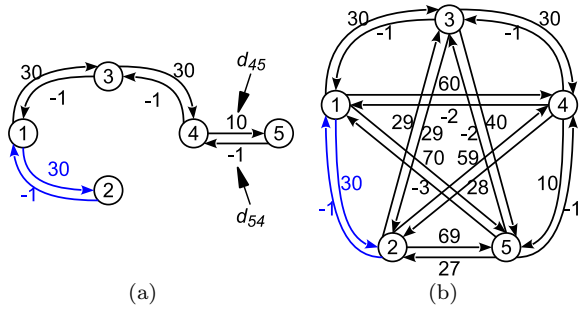


Figure 2: Distance graph for S with one interval [1,10] on edge $4 \to 5$ in Figure 1, and the corresponding minimal network representation

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 $x \vDash S$.

A *distance graph*, $S_d = (N, E_d)$, is a directed edge-weighted graph. It has the same node set as S, and each edge, $i \to j$, is labeled by a weight c_{ij}, representing $X_j - X_i \le c_{ij}$.

Let d_{ij} denote the length of the shortest path from i to j, w.r.t. the edge weights in the distance graph. An equivalent *minimal network representation*, R, of S is defined by $R_{ij} = [-d_{ji}, d_{ij}], \forall i, j \in N$. 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 $i \to j$, 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 R of all S [8].

Example 3 (Example 1 continued)**.** Consider a simple temporal network S with one interval [1,10] on edge $4 \to 5$ in Figure 1. Its corresponding distance graph is plotted in Figure 2(a). An edge, e.g., $4 \to 5$ in the distance graph, with weight $d_{45} = 10$, denotes that $X_5 - X_4 \le d_{45} = 10$. Together with $X_4 - X_5 \le d_{54} = -1$, it is equivalent to the constraint $[1, 10]$ in Figure 1, i.e., $1 \le X_5 - X_4 \le 10$. Figure 2(b) presents the equivalent minimal network representation R for S, by considering the shortest paths for each pair of nodes in Figure 2(a).

Timestamps in trace σ_1 in Figure 1(a) are represented by a tuple x=(09:05, 23:53, 09:25, 09:48, 09:54), where x_1=09:05 denotes the timestamp of event 1, and so on. We say that x_1, x_3 satisfy the temporal constraints, $(x_1, x_3) \vDash R_{13}$, since $x_3 - x_1 = 20 < 30$ and $x_1 - x_3 = -20 < -1$, where 30 and -1, corresponding to edges $1 \to 3$ and $3 \to 1$ in Figure 2(b), respectively, are the temporal constraints $[1, 30]$ in Figure 1.

Consider the simple temporal network S' with another interval [30,40] on edge $4 \to 5$ in Figure 1. Referring to [8], by combining the minimal network representation for S' with Figure 2(b) for S, the minimal network M equivalent to the temporal network G in Figure 1 is obtained, as shown in Table 1. For instance, [1,10][30,40] on edge $4 \to 5$ in Figure 1 is represented by [1,10][30,40] in the cell at row 4 column 5, in the minimal network M in Table 1; or equivalently by [-10,-1][-40,-30] in the cell at row 5 column 4. ■

As studied in [8], by applying all-pair-shortest-paths algorithm to the distance graph, the minimal network representation R can be constructed from the simple temporal network S, in $O(n^3)$ time. Since such 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.

Table 1: Minimal network M equivalent to the temporal constraints G in Figure 1

	1	2	3	4	5
1	[0]	**[1,30]**	[1,30]	[2,60]	[3,100]
2	[-30,-1]	[0]	[-29,29]	[-28,59]	[-27,69]
3	[-30,-1]	[-29,29]	[0]	[1,30]	[2,70]
4	[-60,-2]	[-59,28]	[-30,-1]	[0]	[1,10] [30,40]
5	[-100,-3]	[-69,27]	[-70,-2]	[-10,-1] [-40,-30]	[0]

Repair Model. For a tuple x that does not satisfy the temporal constraints M, denoted by $x \nvDash M$, the repairing is to find another tuple x' by modifying the assignment in x such that $x' \vDash M$. Along the same line of minimum change principle in data repairing [4, 5] (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 $x' \vDash M$ and $\Delta(x, x')$ is minimized.

Theorem 1. *The timestamp repairing problem is* NP*-hard.*

Proof sketch. To prove NP-hardness, we build a reduction from the 3-coloring problem. We show that a tuple x has a repair x' with cost $\Delta(x, x') = k$ iff the graph in reduction is 3-colorable (see proof details in technical report [1]). ■

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 Section 4).

3.1 Tightly Connected Nodes

Consider any repair $x' \vDash M$. We call $i \to j$ a *tight edge* if $x'_j - x'_i = d_{ij}$. 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*

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

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

For example, Figure 4(a) shows a tight chain, with 4 tight edges, between nodes 1 and 5. The numbers attached to nodes denote timestamps, e.g., $x'_1 = 10$, while edges are

associated with weights of temporal constraints (from M in Table 1/Figure 2), such as $d_{12} = 30$ for $1 \to 2$. Since the edges in the chain are tight, we have $x'_2 - x'_1 = 40 - 10 = d_{12}$.

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.

Example 4. Consider a tuple $x = (10, 35, 0, \dots)$, and a repair $x' = (15, 35, 45, \dots)$ of x, as illustrated in Figure 3. We have node $2 \in N_u$, since $x_2 = x'_2 = 35$ is unchanged.

Nodes in $N_m = \{1, 3\}$ are proposed to move (decrease) together, given $x'_1 > x_1$ and the tight edge $1 \to 3$ with $x'_3 - x'_1 = 30 = d_{13}$. By solely decreasing x'_1 (e.g., to 5) without changing x'_3, it leads to violation to $x'_3 - x'_1 \leq d_{13} = 30$. Thereby, x'_3 should decrease together with x'_1.

$N_p = \{1, 3\}$ indicates that decreasing is preferred, since $x'_1 > x_1, x'_3 > x_3$ (see details soon). ■

3.2 Transformation without Increasing Cost

Intuitively, if $|N_p| \geq |N_q|$, by decreasing a very small $\delta, \delta > 0$, 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

$$\begin{aligned}\Delta(x, x') - \Delta(x, x'') &= \sum_j |x_j - x'_j| - |x_j - x''_j| \\ &= |N_p|\delta - |N_q|\delta \geq 0 \end{aligned} \quad (2)$$

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

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

It denotes the maximum amount of allowed variation such that no violation will be introduced to any $k, k \notin N_m$. Equation 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,

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

The min operator ensures that any variation less than θ will not change the relationship between x'_j and x_j for all $j \in N_p$.

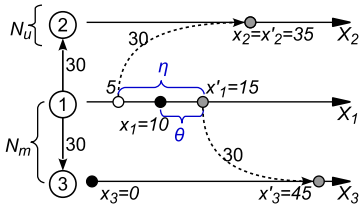


Figure 3: Example of transformation

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.

Example 5 (Example 4 continued)**.** $|N_p = \{1, 3\}| \geq |N_q| = \emptyset$ implies that by decreasing together the assignments of nodes in N_m, the repair cost will not increase.

Referring to Equation 3, $\eta = d_{12} - (x'_2 - x'_1) = 30 - (35 - 15) = 10$ requires the amount of decreasing should not exceed 10, otherwise violation occurs between x'_1 and x'_2 (where $2 \notin N_m$). For instance, an assignment $x''_1 = 4$ with decreasing amount $15 - 4 = 11 > \eta = 10$ is not allowed, since $35 - 4 = 31 > d_{12} = 30$.

Referring to Equation 4, $\theta = x'_1 - x_1 = 15 - 10 = 5$ means that a decreasing amount less than 5 will not change $|N_p = \{1, 3\}| \geq |N_q| = \emptyset$. It ensures the non-increasing repair cost.

After decreasing $\delta = 5$ (according to Case 2 since $\theta < \eta$), $x''_1 = x_1 = 10$ becomes unchanged. Node 3 moving together with 1, having $x''_3 = 40$, is still tightly connected to node 1. Therefore, both nodes 1 and 3 in N_m are moved 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 sketch. Each transformation step ensures no cost increase. By moving changed nodes to N_u, the conclusion is proved. ■

3.3 Transformation Algorithm

Algorithm 1 shows the procedure of the aforesaid transformation from x' to x''. Lines 6 to 8 assemble N_m w.r.t. tight edges. For $|N_p| \geq |N_q|$, N_m proposes to decrease as presented in Section 3.2. Otherwise, Lines 21 to 24 increase the assignment for nodes in N_m.

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

Input: a repair x' of x
Output: a repair x'' with repair cost no greater than that of x', and each changed node in x'' is connected to some unchanged node by a tight chain.

```
1  N_v ← the set of n (unvisited) nodes;
2  N_u ← ∅; N_m ← ∅;
3  while N_v is not empty do
4      move one node i from N_v to N_m;
5      while N_m is not empty do
6          for each j ∈ N_m, i ∈ N_v, d_ji, d_ij ∈ M do
7              if x'_i − x'_j = d_ji or x'_j − x'_i = d_ij then
8                  move node i from N_v to N_m;
9          for each j ∈ N_m, k ∈ N_u, d_jk, d_kj ∈ M do
10             if x'_k − x'_j = d_jk or x'_j − x'_k = d_kj then
11                 move all nodes j from N_m to N_u;
12         for each j ∈ N_m do
13             if x'_j = x_j then                    // unchanged repair
14                 move all nodes j from N_m to N_u;
15         N_p ← {j ∈ N_m | x'_j > x_j};
16         N_q ← {j ∈ N_m | x'_j < x_j};
17         if |N_p| ≥ |N_q| then                     // decrease N_m
18             η ← min_{j∈N_m, k∈N_v∪N_u, d_jk∈M} d_jk − (x'_k − x'_j);
19             θ ← min_{j∈N_p} x'_j − x_j;
20             x'_j ← x'_j − min(η, θ), ∀j ∈ N_m;
21         else                                      // increase N_m
22             η ← min_{j∈N_m, k∈N_v∪N_u, d_kj∈M} d_kj − (x'_j − x'_k);
23             θ ← min_{j∈N_q} x_j − x'_j;
24             x'_j ← x'_j + min(η, θ), ∀j ∈ N_m;
25 return x' as a new repair x''
```

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 sketch. The correctness of Algorithm 1 could be illustrated similar to the proof of Proposition 2. A node will be moved to N_u or N_m only once. For each node, checking its tight connections costs $O(n)$. The algorithm runs in $O(n^2)$ time. ■

Example 6 (Example 5 continued)**.** For a given tuple $x = (10, 35, 0, 52, 60)$ and its repair $x' = (15, 35, 45, 52, 60)$, with cost $\Delta(x, x') = 50$. After applying the transformation in Example 5 (decreasing $N_m = \{1, 3\}$), it forms another repair $x'' = (10, 35, 40, 52, 60)$, with lower cost $\Delta(x, x'') = 40$.

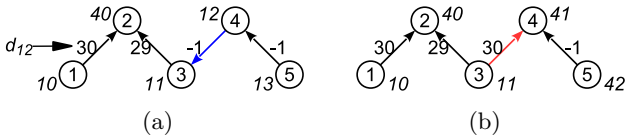
For the remaining nodes, they are unchanged, such as $x'_4 = x_4 = 52$, and will be moved to N_u directly in Line 13. Algorithm 1 terminates. ■

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 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 Section 5).

4.1 Candidates from Tight Chains

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



(a) (b)

Figure 4: Example of (a) tight and (b) provenance chains, for repairing $x = (10, 40, 11, 41, 13)$

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 $x^*_j - x^*_i = d_{ij}$.*

Proof sketch. Assuming reducing x^*_i without modifying the corresponding x^*_j, we prove by contradiction. ■

Similarly, for $x^*_i < x_i$, there must exist an tight edge in the form of $j \to i$ 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 $x^* = (x^*_1, \ldots, x^*_n)$ 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 sketch. The conclusion can be proved by conducting Transform(M, x, x') for any optimal solution x'. ■

We now generate the repair candidate for node j w.r.t. unchanged node i and tight chain $\langle k_0 = i, k_1, \ldots, k_\ell = j\rangle$. 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

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

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.

Example 7. Consider a tuple $x = (10, 40, 11, 41, 13)$. Figure 4(a) illustrates a tight chain for generating repair candidates for x. The number on each edge denotes the constraint from the minimal network M. For instance, 29 on 3→2 corresponds to [-29,29] in row 3 column 2 in Table 1, or more specifically, d_{32}=29 on 3→2 in Figure 2(b). Similarly, -1 on 5→4 corresponds to [-10,-1][-40,-30] in row 5 column 4 in Table 1, or more specifically, d_{54}=-1 on 5→4 in Figure 2(b). All the nodes are connected via this tight chain to the unchanged node $x'_1 = x_1 = 10$. The number attached to each node i represents a repair candidate x'_i, which is computed by Equation 5. For instance, we have $x'_4 = x_1 + d_{12} - d_{32} - d_{43} = 10 + 30 - 29 + 1 = 12$.

The correctness of computing candidates by Equation 5 is verified by showing that each edge in Figure 4(a) w.r.t. x'_i is tight, e.g., for $1 \to 2$, having $x'_2 - x'_1 = d_{12} = 30$. Indeed, this $x' = (10, 40, 11, 12, 13)$ forms an optimal repair with the minimum cost $\Delta(x, x') = 29$. ■

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, $\langle k_0 = i, k_1, \ldots, k_\ell = j\rangle$, such that the tight edges are in the form of either*

$$k_0 \to k_1, k_1 \leftarrow k_2, k_2 \to k_3, k_3 \leftarrow k_4, k_4 \to k_5, \ldots \quad \text{or}$$
$$k_0 \leftarrow k_1, k_1 \to k_2, k_2 \leftarrow k_3, k_3 \to k_4, k_4 \leftarrow k_5, \ldots$$

That is, the directions of consecutive tight edges are always flipped in the chain (see Figure 4(b) for example).

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

Proof sketch. Since edges $i \to j$ and $j \to k$ are tight, we can infer the relationship of $i \to k$ referring to the shortest paths in the minimal network. ■

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

Proposition 7. *An optimal solution $x^* = (x_1^*, \ldots, x_n^*)$ 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 sketch. Referring to Corollary 5, the conclusion is proved by applying the transitivity in Lemma 6. ■

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.

Example 8 (Example 7 continued)**.** Figure 4(b) illustrates a provenance chain. As shown, the directions of edges appear alternatively in the chain. For example, the direction of edge $3 \to 4$ (in red) should be different from the previous $2 \leftarrow 3$ in Figure 4(b). The tight chain in Figure 4(a) has no such constraint, e.g., the edge $3 \leftarrow 4$ (in blue) is acceptable.

As a special tight chain, the repair candidates w.r.t. the provenance chain, $x' = (10, 40, 11, 41, 42)$, are computed by Equation 5 as well. While candidate generation via tight chains has to consider both Figures 4(a) and (b), the generation over provenance chains considers Figure 4(b) only. It is not surprising that, provenance chains lead to more concise candidate sets, and are more efficient. ■

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. Line 2 initializes the start point of all possible provenance chains, whose timestamps are not changed, i.e., the original x_i. Procedure Generate$(N_c, t, i, \text{direction})$ 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 Section 5.2.2 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 $x_{\min}$, 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 $j \in N \setminus N_c$.

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, ..., n};
2  initialize T := {T_i | i ∈ N} where each T_i := {x_i};
3  visited := ∅;
4  for each i ∈ 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 Δ_c(x,t) < Δ(x, x_min) and t ⊨_c M and
       (N_c, t, j, direction) ∉ visited then
3          visited := visited ∪ {(N_c, t, i, direction)};
4          if N_c = N then
5              x_min := t;
6              return;
7          for each j ∈ N \ N_c, d_ij, d_ji ∈ 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 ∪ {t_j} for T_j ∈ T;
15             Generate(N_c ∪ {j}, t, j, flipped);
```

Example 9 (Example 8 continued)**.** Figure 5 illustrate 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 denotes the candidate t_i and the partial cost $\Delta_c(x,t)$. For example, $(t_2, \Delta_c) = (41, 98)$ attached to node 2 denotes that the cost of generating the current chain $\langle 1, 5, 2\rangle$ is $|10-10| + |110-13| + |41-40| = 98$.

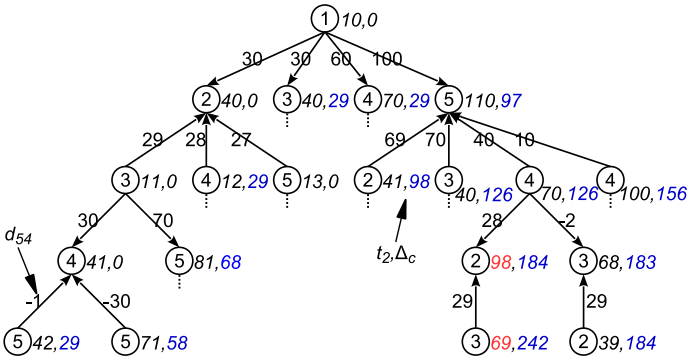


Figure 5: Example of generating candidates for repairing $x = (10, 40, 11, 41, 13)$

It is notable that not all the provenance chains generate feasible candidate sets. For example, for the chain $\langle 1, 5, 4, 2\rangle$ with $t_1 = 10, t_5 = 110, t_4 = 70, t_2 = 98$ (in red), we have $t_2 - t_1 = 88 > d_{12} = 30$, i.e., $t \not\models_c M$. Such chains with violations to temporal constraints are not necessary to consider.

Suppose that Algorithm 2 first expands the left most chain $\langle 1, 2, 3, 4, 5\rangle$, with $t = (10, 40, 11, 41, 42)$ and $\Delta(x, t) = 29$, which will be used as $x_{\min}$ in pruning the subsequent chains. Consequently, the expansion on chain $\langle 1, 2, 4\rangle$, whose Δ_c cost is 29 (in blue), terminates. ■

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 $n - 1$ in Figure 5.)

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. TIMESTAMP REPAIRING

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 Section 5.1).

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.

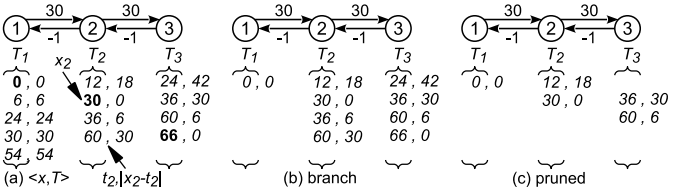


Figure 6: Process for repairing a tuple $x = (0, 30, 66)$**, with temporal constraints over three nodes**

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 9, 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.

Example 10. For simplicity, let us consider a small temporal constraints network over three nodes in Figure 6. Suppose that the candidate sets are obtained for repairing a tuple $x = (0, 30, 66)$, as shown in Figure 6(a). For instance, the candidates for node 2 are $T_2 = \{12, 30, 36, 60\}$, 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 Figure 6(a) is considered for branching. It yields a branch with new $T_1 = \{t_1 = 0\}$ as shown in Figure 6(b). The paid repair cost is $\Delta_p(x, T) = |t_1 - x_1| = 0$, where t_1 is 0. ■

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 $(t_i, t_j) \not\models M_{ij}$ 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$(M, T, x, x_{\min})$

```
1  for each T_i ∈ T, |T_i| > 1 do
2  |  w_min := +∞;
3  |  for each t_i ∈ T_i do
4  |  |  if |t_i − x_i| > Δ(x, x_min) then
5  |  |  |  T_i := T_i \ {t_i};
6  |  |  if (t_i, t_j) ⊨ M_ij, ∀t_j ∈ T_j, T_j ∈ T then
7  |  |  |  T_i := T_i \ {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 ∪ {t_min};
11 for each t_i ∈ T_i, |T_i| = 1, T_i ∈ T do
12 |  for each t_j ∈ T_j, |T_j| > 1, T_j ∈ T do
13 |  |  if (t_i, t_j) ⊭ M_ij then
14 |  |  |  T_j := T_j \ {t_j} for T_j ∈ 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 sketch. By comparing all the pairs of candidates across two nodes, it needs $O(a^2n^2)$ comparisons. ■

Example 11 (Example 10 continued)**.** Suppose that a currently known feasible solution $x_{\min}$ has cost $\Delta(x, x_{\min}) = 36$. For the problem $\langle x, T\rangle$ in Figure 6(b), the candidate $t_3 = 24 \in T_3$ with $|t_3 - x_3| = 42 > 36$ can be directly removed according to the pruning rule (1).

Moreover, consider $T_1 = \{t_1 = 0\}$. According to pruning rule (2), $t_2 = 60 \in T_2$ with $t_2 - t_1 = 60 > d_{12} = 30$ can be removed. Similar pruning applies to $36 \in T_2$ and $66 \in T_3$. Figure 6(c) shows the problem $\langle x, T\rangle$ after pruning. ■

5.2 Putting Techniques Together

We now present the consolidated repair procedure. As interesting by-products, we also devise a simple randomized repairing via transformation, and a heuristic repairing by greedily considering only one branch (instead of all).

5.2.1 Repairing with Pruning

In Algorithm 4, Line 1 employs candidate pruning by $\mathsf{Prune}(M, T, x, x_{\min})$ in Algorithm 3. In each iteration, Line 7 chooses a branch. By removing Lines 13-14 (which are used for heuristic approximation, see details in Section 5.2.2), the branching will continue to compute other solutions. Finally, the program outputs $x_{\min}$ as the optimal solution.

Algorithm 4: $\mathsf{Repair}(M, T, x, x_{\min}, k)$

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  if k > n then
3  |  x' := solution where x'_i = t_i, T_i = {t_i}, ∀T_i ∈ T;
4  |  return x'
5  BC := T_k;
6  while BC ≠ ∅ do
7  |  remove a t_k from BC;
8  |  T' := a branch of T on t_k;
9  |  if T' is feasible and Δ_p(x, T') < Δ(x, x_min) then
10 |  |  x' := Repair(M, T', x, x_min, k + 1);
11 |  |  if Δ(x, x') < Δ(x, x_min) then
12 |  |  |  x_min := x';
13 |  if x_min is feasible/not null then
14 |  |  break;            // for heuristic approximation
15 return x_min
```

Proposition 9. *Algorithm 4 (without Lines 13-14 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 sketch. Referring to the branch and bound computation, it is not surprising to see the $O(a^n)$ complexity. ■

5.2.2 Simple Randomized/Heuristic Repair

Random Assignment Transformation. Besides exact computation, a simple randomized algorithm can be devised by solution transformation. It is worth noting that the input x' of $\mathsf{Transform}(M, x, x')$ in Algorithm 1 is not necessary to be a feasible repair solution. Indeed, given some $x' \not\models M$, the transformation can still output a feasible solution x'' towards a smaller distance (lower repair cost) to x. Thereby, we can randomly draw an assignment x', and transform it to a feasible repair solution x'' by the Transform algorithm.

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 13-14 in Algorithm 4, the program stops branching and directly returns this solution as the repair result.

5.2.3 Consolidated Repairing Procedure

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

(1) Refining the random solution $\tilde{x}'$ via the transformation Algorithm 1, $x_{\min} := \mathsf{Transform}(M, x, \tilde{x}')$;

(2) Generating candidates T according to M and x, by 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.

6. EXPERIMENT

In this section, we present the experimental evaluation, with particular focus on comparing our proposed methods to the existing approaches, Probabilistic [13] and Holistic [7] (see Section 6.3 for details of these compared methods).

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).

Criteria. Following the same line of evaluating data repairing [4], we inject faults in timestamps. Let x_{truth} be the original correct timestamps of a trace, x_{fault} be the error timestamps with injected faults, and x_{repair} be the repaired timestamps. We observe the accuracy measure of repairing [14], 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 Equation 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 [14]. 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.

6.1 Evaluation on Candidate Generation

This experiment evaluates the generation of candidates in Section 4. The experiment is performed on 10 workflow specifications which have 5 nodes/variables. The results are averages over 1750 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.

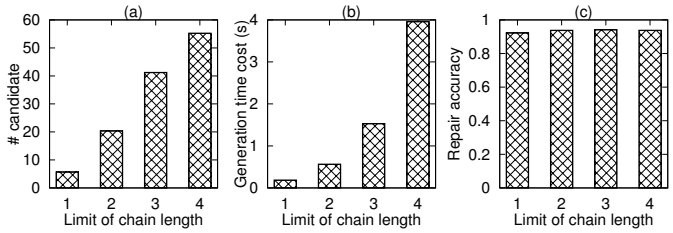


Figure 7: Candidate generation with various lengths of provenance chains

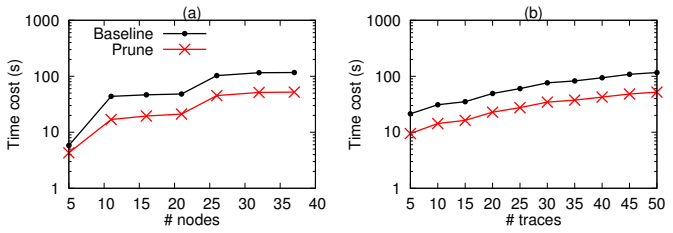


Figure 8: Repairing with pruning

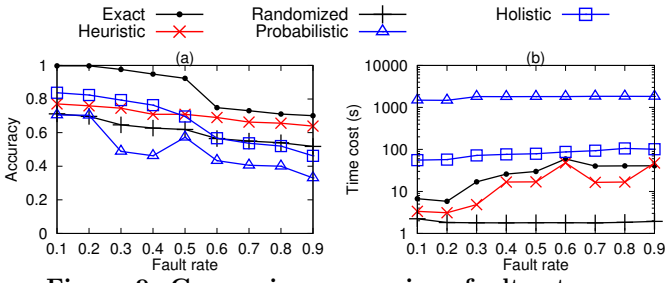


Figure 9: Comparison on various fault rates

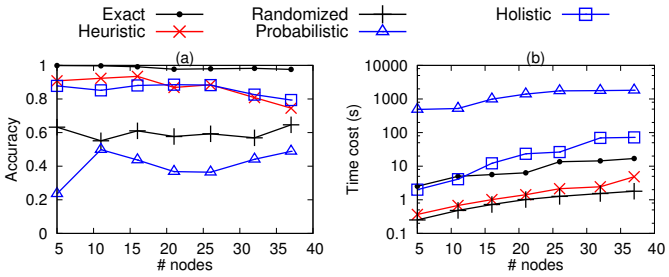


Figure 10: Comparison on various event type sizes

As shown in Figure 7(a), 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 Section 4.

The time cost of candidate generation, in Figure 7(b), 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, Figure 7(c) 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 this result that longer provenance chains have significantly higher time cost but little contribution in improving repair accuracy, we consider below the candidate generation with provenance chain length 4.

6.2 Efficiency of Proposed Techniques

Next, we evaluate the performance of prune techniques in Section 5.1.2 for repairing. The experiment in Figure 8(b) considers various numbers of traces under the same temporal constraints. Therefore, only 50 traces w.r.t. the same temporal constraint network (among 8000 traces w.r.t. different temporal constraint networks) are considered. The size of each trace 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 Section 5.1.1, and Prune in Algorithms 3 and 4. Results in different numbers of nodes (analogous to schema sizes in relational settings) and traces (number of tuples) are presented. It is clear to see the significantly reduced repair time cost by prune.

6.3 Comparison to Existing Methods

This experiment compares our proposal with two other repairing methods, Probabilistic [13] and Holistic [7].

(1) For our proposed Exact repairing, we use the most advanced Algorithm 4 with prune techniques.

(2) Heuristic approximation (in Section 5.2.2, with termination in Line 14 in Algorithm 4) is also evaluated.

(3) To illustrate the rationale of the minimum cost repairing (widely considered in data repairing [4]), we compare a Randomized repair as well, which is not strictly guided by repair cost as presented in Section 5.2.2).

(4) Rather than random selection, the Probabilistic approach [13] studies the distribution of timestamps and uses Bayesian Network to determine repair values.

(5) The Holistic approach [7] greedily repairs data (timestamps) in violations to the given denial constraints [6]. By representing temporal constraints as denial constraints, this repairing method is applicable to timestamp repairing.

Figure 9 reports the results under 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 of fault rate. In Figure 10, while the accuracy is relatively stable, the time cost increases heavily with the number of nodes.

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 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 (comparable to Holistic), whereas its time cost is significantly lower than both Exact and Holistic.

The Probabilistic approach has lower accuracy and much higher time cost, compared to our proposal (Exact and Heuristic). The reason is, as discussed in Section 7, 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.

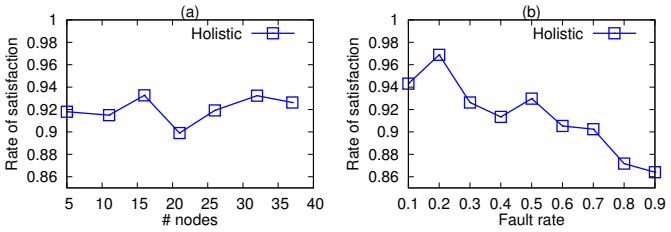


Figure 11: Soundness of results by Holistic

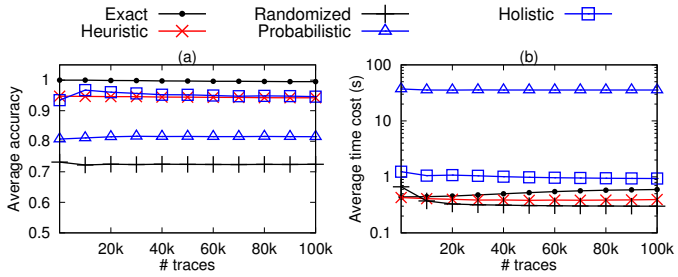


Figure 12: Performance on various traces

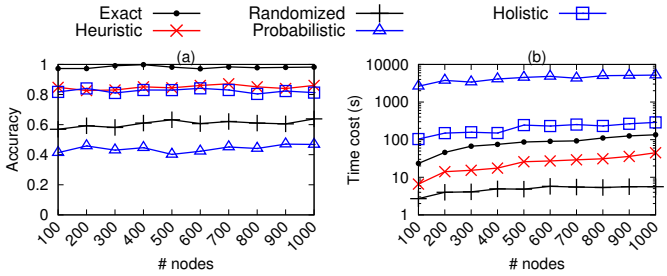


Figure 13: Scalability over a large number of nodes

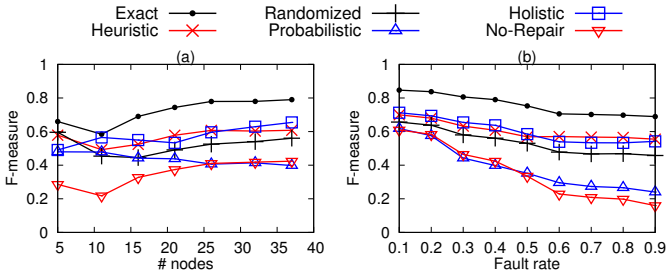


Figure 14: Pattern matching application

The Holistic method also shows lower accuracy but higher time cost, compared to our Exact approach. As discussed (in Section 7 as well), the greedy repair may be trapped in local optima and cannot guarantee to eliminate all the violations. Figure 11 reports the proportion of traces that satisfy the temporal constraints after repairing. As shown, only about 90% traces (tuples) can be entirely repaired without retaining any inconsistencies. In contrast, all our proposed algorithms guarantee to entirely repair the inconsistent timestamps. That is, the rate of satisfaction is always 100%.

The minimum repair cost aware methods, Exact, Heuristic and Holistic, show higher repair accuracy than Probabilistic and Randomized, which do not strictly follow the minimum change principle. The results verify again the intuition that systems or human try to minimize their mistakes in practice, and the rationale of minimizing changes in repairing.

To evaluate the scalability over a larger number of event traces, we employ a log generation toolkit [12] to generate up to 100k traces over the real workflow specifications (introduced at the beginning of this section). Figure 12 illustrates the average accuracy and the average time cost over m traces, with m ranging from 1k to 100k. As shown, both the accuracy and average time cost are stable in various traces. Similar results are observed about the superiority of our proposals compared to the simple Randomized and existing Probabilistic and Holistic approaches.

To evaluate the scalability over a large number of nodes, we employ the generation toolkit [12] again. Workflow specifications with up to 1000 nodes are generated. The total number of traces (tuples) is 1000. A fault rate 0.4 is introduced in the data. As shown in Figure 13, the results are generally similar to the previously reported Figure 10 (with at most 37 nodes). That is, while the accuracy is stable, the time cost increases with the number of nodes. Our proposed Exact algorithm shows higher repair accuracy, and remarkably, lower time costs, compared to the existing Probabilistic and Holistic approaches.

6.4 Application in Pattern Matching

To demonstrate the effectiveness of repairing, we consider a real application of event pattern matching query [2]. A pattern query is expressed with `SEQ` and `AND` predicates. For instance, a real pattern query over the traces in Figure 1 is

```
SEQ(Submit, AND(Normalize, Proofread), Examine)
```

It returns all the traces which first processes a Submit task, and then performs Normalize and Proofread in parallel, followed by an Examine step. Owing to imprecise timestamps, both the order of events and their timestamp distances may vary. The query results could be dramatically distracted (as also observed in the experiments below). For instance, trace σ_1 in Figure 1(a) will not be returned as a result, since event 2 (Normalize) appears after 4 (Examine) owing to the imprecise timestamp 23:53. By repairing the timestamp of event 2 to 09:35 as in Example 2, trace σ_1 can be successfully identified as a result of the aforesaid pattern. In addition to the example results in Figure 1, Figure 14 presents the corresponding pattern query results over the entire dataset.

Let truth be the set of clean traces that match the query pattern, and found be the set of results that are returned by evaluating the pattern query over the fault/repaired traces. To evaluate the pattern matching accuracy, we employ the widely used f-measure [17], given by $precision = \frac{|\text{truth} \cap \text{found}|}{|\text{found}|}$, $recall = \frac{|\text{truth} \cap \text{found}|}{|\text{truth}|}$, and $f\text{-}measure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$.

Figure 14 reports the result accuracy of pattern matching queries over the data repaired by Exact, Heuristic, Randomized, Probabilistic, Holistic approaches and the data without timestamp repairing (No-Repair). It is not surprising that the query result accuracy is low if no repair is performed. The query result accuracy of various repairing approaches is generally similar to the repair accuracy as presented in Figures 9(a) and 10(a). That is, (1) the Exact algorithm can always achieve the highest accuracy, in all the tests; (2) our Heuristic approach also shows better performance than the Randomized and Probabilistic methods, and comparable to Holistic (constraint-based, minimum change guided).

Figure 15 evaluates pattern matching queries involving much longer event traces (up to 1000 nodes). We use again the dataset for evaluating the repair scalability in Figure 13 in Section 6.3. As shown in Figure 15(a), the accuracy is

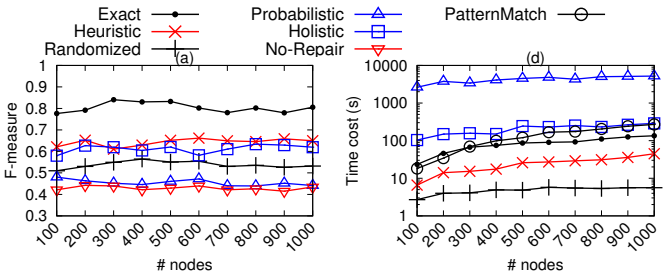


Figure 15: Scalability for pattern matching

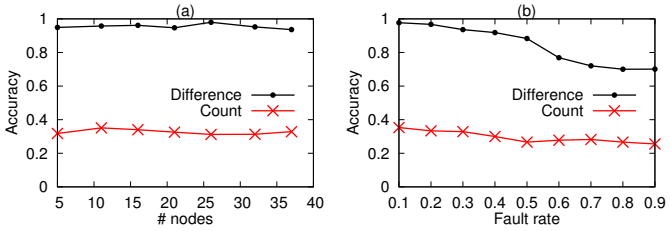


Figure 16: Comparison on repair cost functions

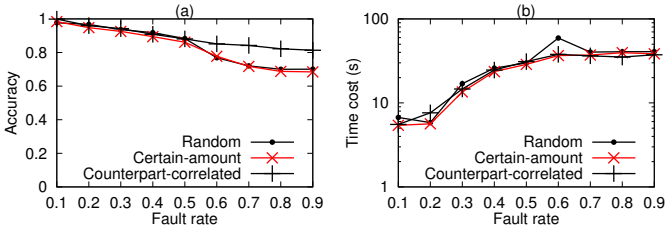


Figure 17: Comparison on various error cases

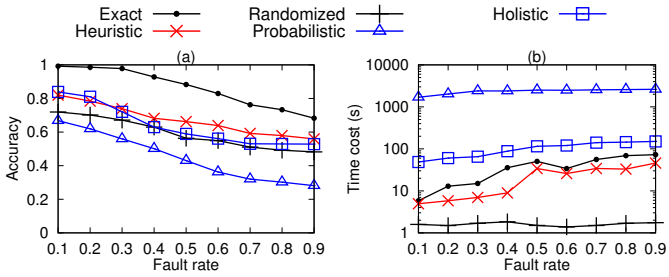


Figure 18: Repairing temperature data

similar to the results over the real data in Figure 14(a). That is, our Exact approach still shows the best performance.

Figure 15(b) reports the time cost of pattern matching together with the time costs of repairing by various approaches. Since the time costs of pattern matching over the data repaired by various approaches are almost the same, we present the average. As shown, the time cost of our Exact repairing is very close to the time cost of pattern matching application. In this sense, the approach could meet the real-time/low-latency requirements of pattern matching queries.

6.5 Evaluation on Cost Functions

Besides the absolute difference-based repair cost metric in Equation 1, other metrics, such as counting the number of changed timestamps, could also be applied. Figure 16 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.

6.6 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 17 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 Figure 17(a). 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.

6.7 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 consider a temperature dataset[1], where each trace recorded the temperatures of 24 hours in a day, i.e., with 24 nodes. There are 15,190 traces collected from 2,168 observation stations. The constraint between two nodes in the minimal network, e.g., [5, 20] from nodes 6 to 12 denotes that the temperature difference between 6 o'clock in the morning and 12 o'clock at noon is at least 5 degree but at most 20 degree. Similar to injecting errors in timestamps, in this experiment, we inject errors in the temperatures, and apply our proposed methods to repair the injected temperature errors.

Figure 18 reports the results over various fault rates. Generally, the results are similar to Figure 9 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 Holistic approaches. The results demonstrate the general applicability and practical value of our proposal in the fields beyond timestamps.

7. PREVIOUS WORK

Owing to the distinct difference between temporal constraints and integrity constraints, most existing data repairing techniques (such as [4] based on functional dependencies) are not directly applicable to repairing timestamps.

Holistic repair [7] can support repairing w.r.t. temporal constraints, by expressing them as denial constraints [6]. It

[1]http://www.cma.gov.cn/

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 [13]. Unlike the holistic cleaning in a constraint-based approach, the repairing in [13] 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. [19] 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 [19] 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 [19].

8. 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 [4]. 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, and (3) a simple randomized method by applying the aforesaid solution transformation. Experiments over a real dataset demonstrate that our proposed method has better performance than the state-of-the-art probabilistic-based and constraint-based repairing approaches, in both repair accuracy (Section 6.3) and application accuracy (Section 6.4).

Besides the widely considered minimum change principle [4, 5], a novel maximum likelihood principle [18] is recently proposed for repairing relational data. The repair likelihood is defined w.r.t. functional dependencies. To adapt the maximum likelihood principle in timestamp repairing, we first need to define the likelihood over timestamps, e.g., w.r.t. temporal constraints. Repair candidates are then generated upon the maximum likelihood instead of the minimum change. We leave this interesting topic as future studies.

Acknowledgement

This work is supported in part by the Tsinghua University Initiative Scientific Research Program; Tsinghua National Laboratory Special Fund for Big Data Science and Technology; China NSFC under Grants 61572272, 61325008, 61370055 and 61202008.

9. REFERENCES

[1] Full version. *http://ise.thss.tsinghua.edu.cn/sxsong/doc/timestamp.pdf.*

[2] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD Conference*, pages 147–160, 2008.

[3] R. S. Barga, J. Goldstein, M. H. Ali, and M. Hong. Consistent streaming through time: A vision for event stream processing. In *CIDR*, pages 363–374, 2007.

[4] P. Bohannon, M. Flaster, W. Fan, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD Conference*, pages 143–154, 2005.

[5] J. Chomicki and J. Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In *Inconsistency Tolerance*, pages 119–150, 2005.

[6] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

[7] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.

[8] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, 1991.

[9] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W.-P. Hsiung, and K. S. Candan. Runtime semantic query optimization for event stream processing. In *ICDE*, pages 676–685, 2008.

[10] C. E. Dyreson and R. T. Snodgrass. Supporting valid-time indeterminacy. *ACM Trans. Database Syst.*, 23(1):1–57, 1998.

[11] W. Fan. Dependencies revisited for improving data quality. In *PODS*, pages 159–170, 2008.

[12] T. Jin, J. Wang, and L. Wen. Efficiently querying business process models with beehivez. In *BPM (demo)*, 2011.

[13] A. Rogge-Solti, R. Mans, W. M. P. van der Aalst, and M. Weske. Improving documentation by repairing event logs. In *PoEM*, pages 129–144, 2013.

[14] S. Song, A. Zhang, J. Wang, and P. S. Yu. SCREEN: stream data cleaning under speed constraints. In *SIGMOD Conference*, pages 827–841, 2015.

[15] P. Sun, Z. Liu, S. B. Davidson, and Y. Chen. Detecting and resolving unsound workflow views for correct provenance analysis. In *SIGMOD Conference*, pages 549–562, 2009.

[16] L. Tang, T. Li, and L. Shwartz. Discovering lag intervals for temporal dependencies. In *KDD*, pages 633–641, 2012.

[17] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[18] M. Yakout, L. Berti-Equille, and A. K. Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *SIGMOD Conference*, pages 553–564, 2013.

[19] H. Zhang, Y. Diao, and N. Immerman. Recognizing patterns in streams with imprecise timestamps. *PVLDB*, 3(1):244–255, 2010.