Efficiently Cleaning Structured Event Logs: A Graph Repair Approach

RUIHONG HUANG, Tsinghua University and Fujian Normal University
JIANMIN WANG and SHAOXU SONG, Tsinghua University
XUEMIN LIN, Shanghai Jiaotong University
XIAOCHEN ZHU, Tsinghua University
JIAN PEI, Simon Fraser University

Event data are often dirty owing to various recording conventions or simply system errors. These errors may cause serious damage to real applications, such as inaccurate provenance answers, poor profiling results, or concealing interesting patterns from event data. Cleaning dirty event data is strongly demanded. While existing event data cleaning techniques view event logs as sequences, structural information does exist among events, such as the task passing relationships between staffs in workflow or the invocation relationships among different micro-services in monitoring application performance. We argue that such structural information enhances not only the accuracy of repairing inconsistent events but also the computation efficiency. It is notable that both the structure and the names (labeling) of events could be inconsistent. In real applications, while an unsound structure is not repaired automatically (which requires manual effort from business actors to handle the structure error), it is highly desirable to repair the inconsistent event names introduced by recording mistakes. In this article, we first prove that the inconsistent label repairing problem is NP-complete. Then, we propose a graph repair approach for (1) detecting unsound structures, and (2) repairing inconsistent event names. Efficient pruning techniques together with two heuristic solutions are also presented. Extensive experiments over real and synthetic datasets demonstrate both the effectiveness and efficiency of our proposal.

CCS Concepts: • **Information systems** → **Data cleaning;**

Additional Key Words and Phrases: Event data, data cleaning, event label repairing

ACM Reference format:
Ruihong Huang, Jianmin Wang, Shaoxu Song, Xuemin Lin, Xiaochen Zhu, and Jian Pei. 2023. Efficiently Cleaning Structured Event Logs: A Graph Repair Approach. *ACM Trans. Datab. Syst.* 48, 1, Article 3 (March 2023), 44 pages.
https://doi.org/10.1145/3571281

This work is supported in part by National Natural Science Foundation of China (62021002, 62072265, 62232005), National Key Research and Development Plan (2021YFB3300500, 2019YFB1705301, 2019YFB1707001), Beijing National Research Center for Information Science and Technology (BNR2022RC01011), and Alibaba Group through **Alibaba Innovative Research** (**AIR**) Program.
Authors' addresses: R. Huang, Fujian Normal University, No. 8 Xuefu South Rd, Fuzhou, Fujian, China, 350117; email: ruihong@fjnu.edu.cn; J. Wang, S. Song (corresponding author), and X. Zhu, Tsinghua University, 30 Shuangqing Rd, Haidian District, Beijing, China, 100084; emails: {jimwang, sxsong}@tsinghua.edu.cn, zhu-xc10@mails.tsinghua.edu.cn; X. Lin, Shanghai Jiaotong University, 800 Dongchuan Rd, Minhang District, Shanghai, China, 200240; email: lxue@cse.unsw.edu.au; J. Pei, Simon Fraser University, 8888 University Drive, Burnaby, BC, Canada, V5A 1S6; email: jpei@cs.sfu.ca.

0362-5915/2023/03-ART3 $15.00
https://doi.org/10.1145/3571281

1 INTRODUCTION

Event data, logging the execution of business processes or workflows, often vary in precision, duration, and relevance [33]. In particular, the execution of a business process may be distributed in multiple companies or divisions, with various event recording conventions or even erroneous executions. The corresponding event data scattered over a heterogeneous environment involve inconsistencies and errors [38]. According to the statistics on a real dataset (in Section 7.1.3), about 82% of execution traces of processes are dirty.

The dirty event data lead to wild data provenance answers [42], mislead the aggregation profiling in process data warehousing [11], or obstruct finding interesting process patterns [19]. Indeed, event data quality is essential in process mining and is known as the first challenge in the Process Mining Manifesto by the IEEE Task Force on Process Mining [44].

Existing approaches [13, 48] on cleaning event data treat event logs as unstructured sequences. It is worth noting that structural information exists among events in many application scenarios. A very common example is the task passing relationships, e.g., the manager assigns the work to another staff for succeeding operations (see details in Example 1). To give another example, the **application performance management** (**APM**) monitors the performance of software applications. The invocation relationships among different micro-services are recorded in the event log [39] (see details in Example 2). We argue that such structural information is not only essential to obtaining more precise event repairs but also useful in improving the computation efficiency.

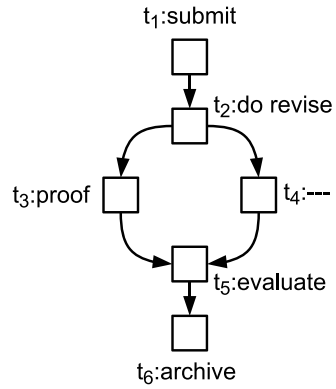
Example 1. We illustrate a real example of the part design process in a major bus manufacturer.[1] Figure 1(a) illustrates 6 steps (events t_1 to t_6) of accomplishing a part design, a.k.a., an *execution trace*. Each event includes a Name of being processed task, the Operator executing the task, and the Successors of the follow-up activities being assigned to. The links of Successor and Operator between events indicate the structural information. For example, the Successor of t_2 assigns the follow-up tasks to J.Zhe and O. Chu (corresponding to Operators in t_3 and t_4, respectively). It indicates the links from t_2 to t_3 and t_4 in the graph of Figure 1(a).

The execution of events should follow some process *specification*s, as shown in Figure 1(c). Following the convention of process management, we represent specifications by the notation of Petri Net [43], which is a graph with two types of nodes, as illustrated in Figure 1(c). The circles represent places, while the rectangles represent transitions. For each directed edge, the types of the adjacent vertexes are different. Each transition denotes an event type, e.g., the first event type submit in Figure 1(c). Arcs with dependent relationships between transitions and places represent the control flow. In particular, flows attached to transitions have AND semantics, such as AND-split after transition design. It indicates that both flows after design should be executed simultaneously. On the other hand, transition evaluate involving AND-join can be enabled when both the preceding flows are complete. Moreover, places specify XOR semantics, e.g., only one of the flows after place a (XOR-split) will be executed, i.e., either design a new part or revise an existing one. Consequently, the XOR-join, e.g., in place s, indicates the end of XOR choices, that is, the execution will proceed when one of the flows before place s is processed. Likewise, either insulation proof or electrician proof can appear in an execution trace after design.

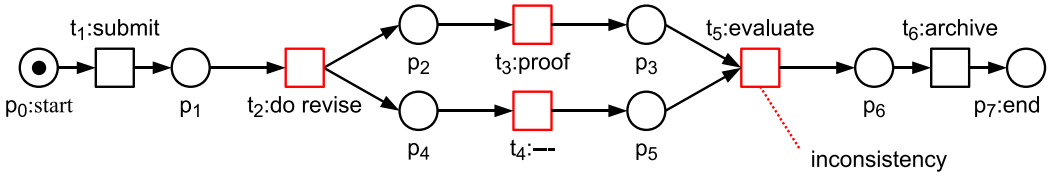
It is notable that the events in this part design process are executed by distinct departments and outsourcing companies. Due to various event recording conventions, a simple proof event name is reported in t_3. It is not clear whether it denotes insulation proof, electrician proof, or proof check. Such ambiguity leads to violations when checking the conformance between execution and

[1]It involves about 70 process specifications for product lifecycle management, which are manipulated by more than 100 outsourcing companies.

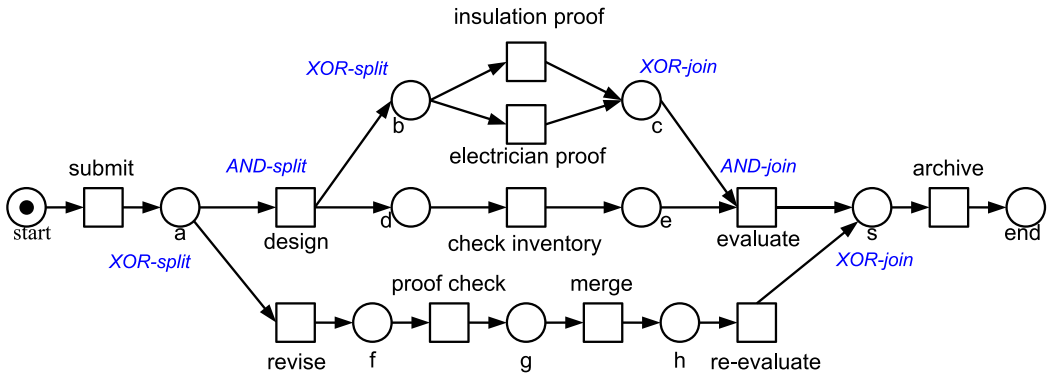
Event	Name	Operator	Successor
t_1	submit	M. Liu	F. Kang
t_2	do revise	F. Kang	J. Zhe & O. Chu
t_3	proof	J. Zhe	X. Feng
t_4	—	O. Chu	X. Feng
t_5	evaluate	X. Feng	System2
t_6	archive	System2	—



(a) An execution trace with 6 events



(b) Representing execution as causal net



(c) Specification for part design process

Fig. 1. Example of structured event data.

specification. Even worse, the event name could be missing in some cases (t_4) for various reasons, such as forgetting to submit when manually recording event logs, or suffering from system failures.

To resolve the inconsistencies, existing repairing techniques [13] may repair the sequence of events $t_1, \ldots, t_6$ to either $\sigma_1\langle$submit, design, insulation proof, check inventory, evaluate, archive$\rangle$, $\sigma_2\langle$submit, design, electrician proof, check inventory, evaluate, archive$\rangle$ for designing a new part, or $\sigma_3\langle$submit, revise, proof check, merge, re-evaluate, archive$\rangle$ for revising an existing part. Referring to the structural information, i.e., t_2 evoking two parallel events t_3 and t_4 (by J. Zhe & O. Chu, respectively), the latter one σ_3 is an invalid repair where no parallel tasks exist and contradicts the structure in Figure 1(b). Instead, the parallel insulation/electrician proof and check inventory after design in σ_1/σ_2 exactly match the structure. With the structural information, we are able to

identify the more likely repair σ_1/σ_2 and discard the irrational σ_3, which cannot be distinguished by existing [13] with sequential information only. (σ_1 and σ_2 can further be distinguished via the cost model in Section 2.4.)

To capture structural information and conformance to specification, we also use Petri Net to represent execution traces, called *Causal Net*. Figure 1(b) shows the net of the execution trace in Figure 1(a). It can be simply transformed from the graph in Figure 1(a) by replacing each edge with a place. Every place in the net is attached with at most one flow (since only one of the alternatives can be executed in XOR-split). The work is accomplished when the flow successfully executes from start to end exactly following the AND/XOR constraints on event (type) names specified by the specification.

With structural information, we can directly ignore the repair candidate σ_3 which corresponds to the revise division without parallel tasks. Repairing efficiency is thus improved compared with the simple sequence-based approaches.

In general, both the Name labeling and the structural Operator/Successor may involve dirty information, known as (1) *inconsistent labeling* and (2) *unsound structure*. (A structure is said unsound if it cannot find any labeling conforming to the specification, see Example 4 for instance). The unsound structure may be raised due to ad-hoc re-assignment of operators, e.g., a task is assigned to a successor J. Zhe but actually executed by the operator O. Chu. Such structural inconsistency needs business actors to manually handle. Inconsistent labeling of event names, however, typically occurs owing to mistakenly recording. Therefore, it is highly desirable to efficiently detect unsound structures, and repair the inconsistent labeling with sound structure. According to our statistics in a real dataset (see details in Table 3 in the experiments), among traces with detectable inconsistencies,[2] about 5.42% are raised by unsound structure, while the others (about 94.58%) are structurally sound but with inconsistent labeling.

In this article, we study two problems of cleaning event data, (1) detecting unsound structure; or (2) returning a repair of event names if the structure is sound. That is, while reporting all detectable inconsistencies, we also try to remedy the majority of inconsistencies as accurate as possible.

1.1 Challenges

The major challenges of detecting and repairing dirty event data originate from coupling of data and logic. The repairing problem is proved to be NP-complete (in Section 3). The hardness of repairing workflow execution logs comes from the spread of inconsistencies, i.e., modifying the task of one transition often introduces new inconsistencies in other transitions with dependent, AND or XOR constraints. In particular, in the workflow execution, repairing one transition by mistake may lead to complete modification of the following executions.

Existing database repairing techniques [22, 41] cannot handle the complex structural relationships, e.g., t_2[Successor]= J. Zhe & O. Chu denoting the follow-up relationships among t_2, t_3, t_4. Moreover, the constraints specified by process specifications are very different from integrity constraints in relational data. In particular, data dependencies declare relationships in tuple pairs, while process specifications indicate constraints on events with flow directions, AND/XOR semantics.

Adapting the existing graph relabeling technique [40], by treating execution and specification as simple graphs, falls short in two aspects: (1) the AND/XOR semantics are not considered; (2) the vertex contraction technique in [40] modifies the structure of execution and thus cannot detect unsound structure. Recent works utilize **graph functional dependencies** (**GFDs**) to capture and

[2] Other errors, that are consistent w.r.t. the specification, are unlikely to be detected without further knowledge and are not in the scope of this study.

repair inconsistencies in graph data [28, 29]. However, the graph patterns defined in GFDs cannot support the AND/XOR semantics on events very well, since the graph structures of execution and specification could be different due to the existence of XOR branches.

The conformance checking [13, 34, 37] studied in the process mining field also assesses the deviations of event data with respect to the expected behavior of the process. However, they work on a different problem from ours. The commonly used alignment algorithm [13, 34] returns an alignment between the execution trace and process specification, which consists of a sequence of *moves*. Each *move* relates an event in the trace to the one in the specification. When the event observed in the trace is allowed according to the specification, it is a *synchronous move*. Otherwise, the move is unsynchronized, and is indicated by either a *log move* (the observed event in the trace is not allowed by the specification), or a *model move* (an event should be observed according to the specification but missing in the trace). For example, the repair candidate σ_3 in Example 1 can be returned by the alignment with two synchronous moves (submit, archive), four model moves (revise, proof check, merge, re-evaluate), and ignoring the four log moves (do revise, proof, -, evaluate). As a result, the structure of execution could be modified when the unsynchronized model/log move occurs, and the unsound structure cannot be detected. Since most alignment algorithms simply regard the execution as a sequence of events [20], we compare the p-alignment [34] in the experiment part, which also considers the structural information (partial order) among events. Unfortunately, p-alignment still suffers the aforesaid problems, and its performance is not as good as our proposal.

1.2 Contributions

While using structure event logs based on structural information has been validated in the conference version [47], we further enhance the study by analyzing the hardness of the problem, leading to more effective approximation, and showing its optimal solutions in certain datasets. For the hardness analysis, the NP-completeness of the studied problem indicates that the high time complexity of the exact repairing algorithm is not surprising. Therefore, we turn to more effective approximate solutions. For a new heuristic method, we show that it indeed returns the exact solution in a special case of simple path structure. Moreover, the heuristic method can achieve comparable accuracy to the exact algorithm while keeping relatively lower time costs in practice. For two more datasets, the newly conducted experiments on the Bank dataset with more nested AND/XOR structures in the specification illustrate the significantly higher time cost of the exact repairing algorithm, and thus the needs for heuristic algorithms. In addition, the experiments on the Log dataset verify that the new heuristic method can give an exact solution under the special case of a simple path.

Our major contributions in this article are summarized as follows.

(1) We analyze the complexity of the studied problem in Section 3. We prove that the inconsistent labeling repair problem is NP-complete by a reduction from the set cover problem (Theorem 2).
(2) We propose an exact repairing algorithm to either provide the optimal repair of an execution trace or conclude an unsound structure in Section 4. Branch and bound algorithms are also devised together with several efficient pruning techniques.
(3) We develop an efficient PTIME transition-oriented heuristic algorithm, by only one pass through the transitions in the execution trace in Section 5. Although it may generate false negatives regarding the detection of unsound structure and may not be able to guarantee the optimal repairing, the performance studies show that it can achieve good accuracy while keeping time cost extremely low.

(4) We present another place-oriented heuristic algorithm, by enumerating all the possible labelings for each place in the execution in Section 6. Besides, we introduce a special case of simple path structure, which is prevalent in real-world scenarios, and show that the place-oriented heuristic algorithm can give an exact solution in this special case (Proposition 7).
(5) We report an extensive experimental evaluation to demonstrate the performance of the proposed methods in Section 7. Repairing accuracies of both our exact and heuristic approaches (greater than 90% in most tests) are significantly higher than the state-of-the-art sequence-based [34] and graph-based [28, 40] methods. The transition-oriented heuristic algorithm achieves significant improvement in time performance compared with the exact approach, while the place-oriented heuristic method shows better approximation results in both real and synthetic datasets.

The rest of the article is organized as follows. We introduce preliminaries in Section 2. We show the NP-completeness of the studied problems in Section 3. Major results of the exact detecting/repairing algorithms are presented in Section 4. Two heuristic algorithms are presented in Sections 5 and 6, respectively. Section 7 provides an experimental evaluation. Finally, we discuss related work in Section 8 and conclude the article in Section 9.

2 PROBLEM STATEMENT

We first formalize syntax and definitions for process specifications and executions. The conformance between specifications and executions is then introduced, which raises the detecting and repairing problems.

2.1 Preliminary

For a function f and a set A, let $f(A)$ denote $\{f(x) \mid x \in A\}$. In this article, we follow the notations of Petri net [36], also known as a **place/transition (PT)** net. A transition denotes a task to execute in a workflow. A token enables the execution of transition. A place holds tokens before a transition is executed.

Definition 1. A *net* is a triplet $N = (P, T, F)$, where **(i)** P is a finite set of places, **(ii)** T is a finite set of transitions, $P \cap T = \emptyset$, **(iii)** $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, namely flow relation. While the network structure is static, tokens denoted by black dots can flow through the network. A place, denoted by circle, is used to hold tokens that enable the execution of transition actions. A transition is called enabled if each of its input places contains a token.

It is notable that the concept of markings and tokens of Petri nets [36] are omitted for brevity. A net is a bipartite directed graph, with set F of edges between nodes in P and T. Each $(x, y) \in F$ is a directed arc from node x to node y. For any $x \in P \cup T$, let

$$\text{pre}_F(x) = \{y \mid (y, x) \in F\},$$

be the set of all input nodes of x and

$$\text{post}_F(x) = \{y \mid (x, y) \in F\},$$

denote the set of all output nodes of x.

Definition 2. A *process specification* is a Petri net $N(P, T, F)$ such that **(i)** P contains a source place having $\text{pre}_F(\text{start}) = \emptyset$, **(ii)** P contains a sink place having $\text{post}_F(\text{end}) = \emptyset$.

The AND-split denotes the multiple outgoing flows of a transition. It means that all the flows after the AND-split transition should be executed in parallel. Likewise, the AND-join, corresponding to the multiple flows before a transition, indicates all the preceding parallel flows are complete. In contrast, the XOR-split denotes the multiple flows after a place. It means the choice execution

that only one of the flows after the XOR-split place will be executed. Similarly, the XOR-join, corresponding to the multiple flows before a place, indicates that the execution can proceed when one of the flows before the XOR-join place is processed. It is notable that the *process specification* is also referred to as *process model* [34] or *system net* [37] in the work of *conformance checking*. In the specification, the AND-split is represented by the transition with multiple outward flows, such as design with two outward flows in Figure 1(c). To clearly mark the structure, we also put text AND-split next to the transition. Likewise, the AND-join is denoted as the transition with multiple inward flows. And similarly, the XOR-split (or XOR-join) is marked by the place with multiple outward (or inward) flows. It is not surprising that the AND-split/join and XOR-split/join structures in the specification complicate the studied problems (as analyzed in Section 2.4). The AND-split leads to various possible correspondences of an event. For example, t_4 in Figure 1(b) may correspond to check inventory, electrician proof, or insulation proof after the AND-split in the specification in Figure 1(c). Likewise, the XOR-split also leads to multiple choices, e.g., t_2 in Figure 1(b) may either be design or revise after the XOR-split in the specification in Figure 1(c).

Definition 3. A *causal net* is a Petri net $N = (P, T, F)$ such that for every $p \in P$, $|\text{pre}_F(p)| \leq 1$ and $|\text{post}_F(p)| \leq 1$.

It is easy to see that there will be no XOR-split or XOR-join in a causal net, since only one of the flows will be executed after a place, i.e., the maximum in/out degree 1 of places. Instead, AND-split and AND-join are allowed, since multiple flows could be executed in parallel after a transition, i.e., no restriction for the number of the degree of transitions. In addition, cycles are excluded in the causal net. If we interpret places as edges connecting two transitions, the net is indeed a directed acyclic graph of transitions [21].

Definition 4. An *execution* of a process specification $N_s(P_s, T_s, F_s)$ is denoted by (N_σ, π), where $N_\sigma(P_\sigma, T_\sigma, F_\sigma)$ is a causal net and π is a labeling

$$\pi : P_\sigma \cup T_\sigma \rightarrow P_s \cup T_s,$$

such that $\pi(P_\sigma) \subseteq P_s$, and $\pi(T_\sigma) \subseteq T_s$.

We use $y : Y$ to denote $\pi(y) = Y$ for short, where y is a transition/place in N_σ mapping to a transition/place Y in N_s via π, e.g., $\pi(t_1)$=submit denoted by t_1:submit in Figure 1.

Definition 5. We say an execution (N_σ, π) *conforms* to a process specification N_s, denoted by

$$(N_\sigma, \pi) \vDash N_s,$$

if and only if **(i)** $\pi(P_\sigma) \subseteq P_s$ and $\pi(T_\sigma) \subseteq T_s$; **(ii)** for any $t \in T_\sigma$, $\pi(\text{pre}_{F_\sigma}(t)) = \text{pre}_{F_s}(\pi(t))$ and $\pi(\text{post}_{F_\sigma}(t)) = \text{post}_{F_s}(\pi(t))$; **(iii)** for any $p \in P_\sigma$, $\text{pre}_{F_\sigma}(p) = \emptyset$ implies $\pi(p) = \text{start}$ and $\text{post}_{F_\sigma}(p) = \emptyset$ indicates $\pi(p) = \text{end}$.

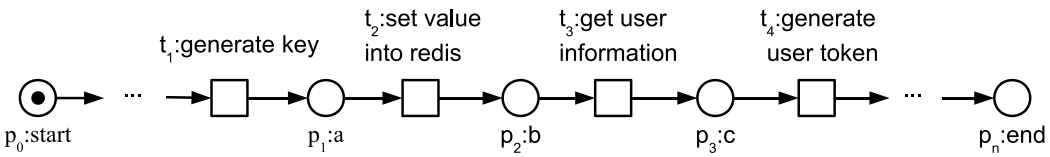
That is, there is a bijection between pre_{F_σ} and pre_{F_s} for each transition t in the execution (N_σ, π), and similarly, for post_{F_σ} and post_{F_s}.

Example 2. In addition to the real example in Figure 1 about the part design process in a bus manufacturer, in this example, we consider another real APM scenario from our industrial partner Cloudwise, for monitoring the performance of software applications.

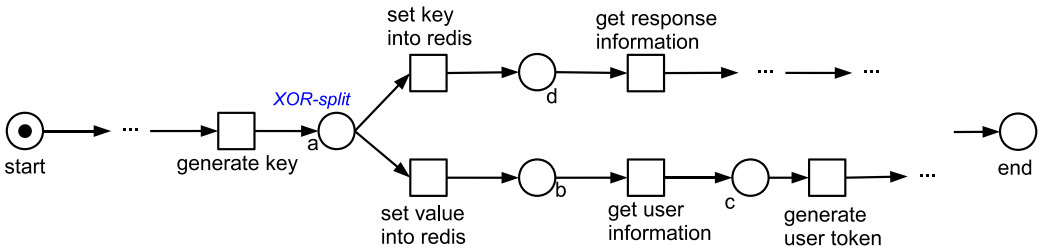
Figure 2(a) illustrates parts of the example execution trace, which relates to the process of user login with QR code. The process starts with the web service generating a key related with the QR code. Then the redis service records the correspondence between the QR code and the user who scan the code using a cell phone. After that, the db service queries the user information from the database, and finally the login service generates a user tokens according to the obtained information. The structural information is derived from the invocation relationships among different

Event	Name	Operator	Successor
...	...	...	...
t_1	generate key	10.1.1.114 (web service)	10.1.1.115 (redis service)
t_2	set value into redis	10.1.1.115 (redis service)	10.1.1.116 (db service)
t_3	get user information	10.1.1.116 (db service)	10.1.1.117 (login service)
t_4	generate user token	10.1.1.117 (login service)	...
...	...	...	...

(a) Parts of execution trace



(b) Representing execution as causal net



(c) Specification for user login process

Fig. 2. Example structured event data from a distributed system.

micro-services. For example, the event t_1 in Figure 2(a) is executed by web service whose IP address is 10.1.1.114. In addition, the executor of the following event of t_1 is also recorded, which is the redis service. The corresponding causal net of the execution is demonstrated in Figure 2(b), while the specification for user login process is shown in Figure 2(c). As illustrated, the process specification is defined with the XOR structure, providing two branches of setting keys or values into redis. In contrast, the execution model is constrained to acyclic causal nets that can only choose one of the XOR branches, e.g., set value into redis in t_2 in Figure 2(b).

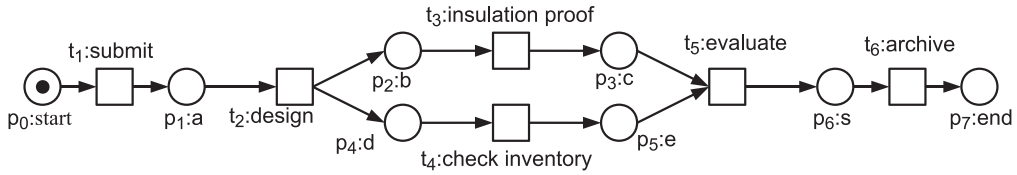
Due to the failure of packet parsing, the evolution of log events across versions [51] or problems with network transmission, the event data generated by the software applications could be inconsistent with the specification. The corresponding experiment results of the aforesaid APM scenario are reported in Figure 26 in Section 7.5.

2.2 Execution Trace

In practice, execution is stored as execution trace σ, with schema (Event, Name, Operator, Successor,...). Each tuple in σ, a.k.a. an event, denotes a transition in execution $t_i \in T_\sigma$, ordered by execution timestamp, e.g., the ith executed event/transition $\sigma(i) = t_i$. By the labeling π, each event t_i in T_σ is associated with a name $\pi(t_i)$, which usually corresponds to a type in the specification N_s, i.e., $\pi(t_i) \in T_s$.

Event	Name	Operator	Successor	Conf
t_1	submit	A	B	1.0
t_2	design	B	C & D	0.6
t_3	insulation proof	C	E	0.7
t_4	check inventory	D	E	0.1
t_5	evaluate	E	F	0.9
t_6	archive	F	—	0.8

(a) Example of an execution trace



(b) Representing execution as causal net

Fig. 3. Example of conformance.

Execution trace also records the net structure of execution. As there is no XOR-split or XOR-join in the causal net of an execution, each place p_j in $\text{pre}_{F_\sigma}(t_i)$ corresponds to exactly one transition, say $\text{pre}_{F_\sigma}(p_j) = \{t_j\}$. Combining t_j of all p_j forms $\text{pre}_{F_\sigma}(\text{pre}_{F_\sigma}(t_i))$, i.e., the prerequisite of t_i.

Proposition 1. *For any $\sigma(j) = t_j, \sigma(i) = t_i, j < i$ in a trace σ, it always has $t_i \notin \text{pre}_{F_\sigma}(\text{pre}_{F_\sigma}(t_j))$ and $t_j \notin \text{post}_{F_\sigma}(\text{post}_{F_\sigma}(t_i))$.*

Proof. According to the execution of transition in the causal net N_σ, event t_i is executed after event t_j and cannot be the input node of any nodes in $\text{pre}_{F_\sigma}(t_j)$. Thus, we have $t_i \notin \text{pre}_{F_\sigma}(\text{pre}_{F_\sigma}(t_j))$. Similarly, event t_j cannot be the output node of any nodes in $\text{post}_{F_\sigma}(t_i)$, i.e., $t_j \notin \text{post}_{F_\sigma}(\text{post}_{F_\sigma}(t_i))$. □

Thus, no t_i can appear before its prerequisite t_j in a trace σ.

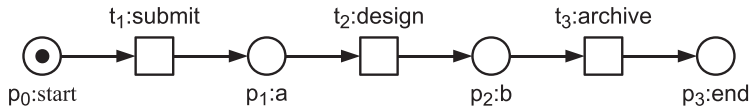
Conformance of the execution trace can be checked by recovering its corresponding causal net, i.e., recovering places (and labeling) between a transition and its prerequisite (as places are not recorded in the execution trace).

Example 3 (Example 1 Continued). Consider another execution trace in Figure 3(a) over the specification in Figure 1(a). We represent the corresponding causal net in Figure 3(b) as follows. For the first t_1 without any prerequisite, we put a place p_0 with $\pi(p_0) = \text{start}$ as the pre set. The second event $\sigma(2) = t_2$ has prerequisite $\text{pre}_{F_\sigma}(\text{pre}_{F_\sigma}(t_2)) = \{t_1\}$. We recover the labeling of the place p_1 between t_2 and its prerequisite t_1 to the place between $\pi(t_2)$ and $\pi(t_1)$ in the specification, i.e., $\pi(p_1) = \text{a}$. Similarly, considering the prerequisites of t_5, $\text{pre}_{F_\sigma}(\text{pre}_{F_\sigma}(t_5)) = \{t_3, t_4\}$, we obtain $\pi(p_3) = \text{c}, \pi(p_5) = \text{e}$. For the last t_6, which is not a prerequisite of any others, a place $p_7 : \text{end}$ is appended as $\text{post}_{F_\sigma}(t_6)$.

Referring to the conformance definition, for any transition, e.g., t_1, we have $\pi(\text{pre}_{F_\sigma}(t_1)) = \pi(p_0) = \{\text{start}\} = \text{pre}_{F_s}(\text{submit}) = \text{pre}_{F_s}(\pi(t_1))$ and $\pi(\text{post}_{F_\sigma}(t_2)) = \pi(\{p_2, p_4\}) = \{\text{b,d}\} = \text{post}_{F_s}(\text{design}) = \text{post}_{F_s}(\pi(t_2))$.

Event	Name	Operator	Successor
t_1	submit	A	B
t_2	design	B	C
t_3	archive	C	—

(a) Example of an execution trace



(b) Representing execution as causal net

Fig. 4. Example of unsound structure.

We consider two types of inconsistencies, unsound structure and inconsistent labeling, which harm the conformance.

2.3 Unsound Structure Detection Problem

We say that a causal net N_σ is *unsound* w.r.t. the specification N_s, if there does not exist any labeling π such that (N_σ, π) forms an execution conforming to N_s.

Problem 1. Given an execution (N_σ, π) over the specification N_s, the *unsound structure detection problem* is to determine whether there exists a labeling π' such that $(N_\sigma, \pi') \vDash N_s$.

In other words, the structure is sound if there exists at least one labeling π' to make the conformance. It is worth noting that the unsound structure detection is different from the conformance checking problem [13, 34, 37] in the process mining field, which does not consider the repairing of inconsistent labeling of events. In conformance checking, usually a sequential alignment between the execution and specification is returned. When the event observed in the execution is not allowed in the specification, an unsynchronized log or model move is recorded in the alignment. An optimal alignment is the one with minimum deviation (unsynchronized moves). Since the structure of the execution may be modified when the log/model occurs in the alignment, unsound structure cannot be detected in this way.

Example 4 (Example 1 Continued). Consider another execution trace in Figure 4 over the specification in Figure 1(a). As shown in the recovered causal net, the second t_2 involves inconsistency that $\pi(\text{post}_{F_\sigma}(t_2)) = \pi(p_2) = \{\text{b}\} \neq \{\text{b,d}\} = \text{post}_{F_s}(\text{design}) = \text{post}_{F_s}(\pi(t_2))$.

To accomplish the work specified in Figure 1(a), at least two transitions should be processed which take t_2 as prerequisites. However, only one transition t_3 in the causal net in Figure 4 has prerequisite t_2. It is impossible to find any labeling π' that can make conformance to the specification.

Unsound structures mainly result from business fraud. Such structural inconsistency needs further manual handling before repairing the inconsistent labels, such as discarding the execution or re-executing the workflow. While we can detect unsound structure, as mentioned, handling unsound structure is not the focus of this study. As illustrated in Theorem 2 below, the repairing for inconsistent labeling is already hard. While handling inconsistent structures need to further consider all possible execution structures of the process specification, which makes the problem even more complex. Therefore, we leave the challenging problem of unsound structure repairing as the future study.

2.4 Inconsistent Labeling Repair Problem

For an execution trace with sound structure, we can repair the *inconsistent labeling* of events. Repairing the execution can be viewed as the relabeling of transitions (and places) from N_σ to the specification N_s. The new labeling, say π', should meet the conformance requirement.

As discussed in the introduction, along the same line of database repairing [9], a typical principle is to find a repair that minimally differs from the original data.

Let (N_σ, π') be a repaired execution of the original (N_σ, π) by changing the labeling function from π to π' such that $(N_\sigma, \pi') \vDash N_s$. The event repairing cost is given by

$$\Delta(\pi, \pi') = \sum_{t \in T_\sigma} \delta(\pi(t), \pi'(t)), \tag{1}$$

where $\pi'(t)$ is the new (type) name of the transition (event) t in the repair (N_σ, π'), and $\delta(\pi(t), \pi'(t))$ denotes the cost of replacing $\pi(t)$ by $\pi'(t)$.

Let $conf(t)$ denote the confidence associated with transition t, as the example illustrated in Figure 3(a). It is defined as the possibility of the specific event being correctly recorded by the executor in the historical data, e.g., $conf(t) = \frac{correct(t)}{processed(t)}$ where $processed(t)$ is the total number of event t processed by the executor in the historical data, while $correct(t)$ represents the number of the correctly recorded ones. The confidence field is optional and analogous to the confidence of each tuple in database repairing [9]. The definition of $conf(t)$ can be regarded as the precision of the event being correctly recorded by the executor in the historical data. It is used to infer whether t in the current execution should be repaired or not. According to the definition, higher confidence means that most processing of t in the history is correct. For example, for those events t processed by reliable executors, e.g., a senior skilled worker, the confidence $conf(t)$ is higher. In this sense, we tend to not modify such highly confident event t that is probably correct according to the history. It is achieved by assigning a larger repair cost of t, i.e., more tending to not repair t.

The frequency $freq(\pi(t))$ is defined as the ratio of the specific event name $\pi(t)$ appearing in the historical data compared to all the event names, i.e., $freq(\pi(t)) = \frac{num(\pi(t))}{all}$ where all is the total number of all the event names appearing in the historical data, while $num(\pi(t))$ denotes the number of the specific name $\pi(t)$. Frequency is an observation of "user behaviors" and may help in repairing. For instance, in Figure 1(a), if insulation proof appears much more *frequently* than electrician proof in the database of all execution traces, we may repair t_3 by insulation proof. The cost of repairing a high-frequency $freq(\pi(t))$ to a low frequency $freq(\pi'(t))$ is large.

Therefore, the cost δ of replacing $\pi(t)$ by $\pi'(t)$ can be defined as

$$\delta(\pi(t), \pi'(t)) = conf(t) \cdot dis(\pi(t), \pi'(t)) \cdot \frac{freq(\pi(t))}{freq(\pi'(t))}, \tag{2}$$

where $dis(\pi(t), \pi'(t))$ denotes the metric distance between two names $\pi(t)$ and $\pi'(t)$, e.g., edit distance.

The confidence $conf(t)$ and the frequency $freq(\pi(t))$ could be estimated from the historical data in advance. For the unseen labelings in the repairing, we set a fixed value for confidence and frequency, respectively. The edit distance $dis(\pi(t), \pi'(t))$ between two strings of event names can be computed online in an ad-hoc way, i.e., we do not need to pre-define the cost for each pair of $\pi(t)$ and $\pi'(t)$.

Example 5 (Example 1 Continued). Consider the cost δ of repairing t_3 in Figure 1(a) by <insulation proof>. As illustrated in Figure 3(a), the confidence of the event t_3 being correctly recorded by the executor is $conf(t_3) = 0.7$. That is, according to the historical data, there are 70 percent of event t_3 being correctly recorded by the executor. Moreover, suppose that

<insulation proof> appears 300 times over all the 6,000 events in the historical data, the frequency of <insulation proof> is given by $freq(\text{insulation proof}) = \frac{300}{6,000} = 0.05$. In contrast, the original erroneous name <proof> of event t_3 in Figure 3(a) rarely occurs in the historical data, having $freq(\text{proof}) = 0.001$. Furthermore, we use edit distance for the metric distance between two names $\pi(t)$ and $\pi'(t)$, i.e., $dis(\text{proof}, \text{insulation proof}) = 11$ for at least 11 character editing operations including blank. According to Equation (2), the cost of repairing t_3 by <insulation proof> is

$$\delta(\pi(t_3), \pi'(t_3)) = conf(t_3) \cdot dis(\text{proof}, \text{insulation proof}) \cdot \frac{freq(\text{proof})}{freq(\text{insulation proof})}$$
$$= 0.7 \cdot 11 \cdot \frac{0.001}{0.05} = 0.154.$$

Consider the cost δ of repairing t_3 with an incomplete name <proof> by another irrelevant <check inventory>. The confidence $conf(t_3)$ is still 0.7, while the frequency of <check inventory> is given by $freq(\text{check inventory}) = \frac{360}{6,000} = 0.06$. Furthermore, the edit distance between two names is $dis(\text{proof}, \text{check inventory}) = 15$. According to Equation (2), the cost of repairing t_3 by <check inventory> is

$$\delta(\pi(t_3), \pi'(t_3)) = conf(t_3) \cdot dis(\text{proof}, \text{check inventory}) \cdot \frac{freq(\text{proof})}{freq(\text{check inventory})}$$
$$= 0.7 \cdot 15 \cdot \frac{0.001}{0.06} = 0.175.$$

Even though <check inventory> has a higher frequency than <insulation proof> in the historical data, the cost 0.154 of repairing the inconsistent event name <proof> to a related value <insulation proof> is lower.

Let π^* be the ground truth of the input labeling π. That is, $\pi^*(t)$ denotes the true name of the incorrect $\pi(t)$ in the real-world for event t in the input causal net N_σ. It is true that for constraint-based data cleaning, a consistent dataset does not mean that it is the ground truth π^*. Among many possible repairs that can satisfy the consistency, rather than choosing a random value, we may use some hints to return the one most likely to be the ground truth. The repair cost function in Formula 2 thus proposes to consider several aspects of possible hints. (1) The distance metric *dis* indicates how distant the repaired name is to the original name. Following the intuition that the repair should avoid losing information of the original data [23], the modification is expected to be minimized. For example, in Figure 1(b), it is more reasonable to repair the inconsistent event name <proof> to a related value such as <insulation proof>, than another arbitrary value such as <check inventory>, which differs greatly from the original name. (2) The confidence *conf* provides the ability to manually adjust the repair cost. It is possible that the ground truth may not be the one with the minimum modification. Nevertheless, for those ones believed to be true values, we can set a very large confidence (e.g., infinite value). This large confidence leads to a very large repair cost, and thus will not be considered and returned as the minimum repair. (3) The frequency *freq* denotes the frequency of the event names appearing in the historical data. A repaired name with higher frequency is preferred, rather than a random value that may rarely occur.

Problem 2. Consider the input of a causal net N_σ and a labeling π as execution, and the specification N_s. The *inconsistent labeling repairing the problem* is to find a relabeling π' of π, if exists, such that (i) the execution conforms to the specification $(N_\sigma, \pi') \vDash N_s$ and (ii) the repairing cost $\Delta(\pi, \pi')$ is minimized.

The input causal net N_σ, as defined in Definition 3, provides both a set of finite places and a set of finite transitions. The output is a relabeling π' of π that conforms to the specification, while the structure of causal net N_σ is unchanged. Moreover, the repairing cost $\Delta(\pi, \pi')$ is minimized.

3 HARDNESS ANALYSIS

Before introducing technical details, let us first investigate the hardness of the studied problems.

THEOREM 2. *Given an execution (N_σ, π) over the specification N_s, and an integer K, the problem of determining whether there is a relabeling π' such that $(N_\sigma, \pi') \vDash N_s$ with the cost $\Delta(\pi, \pi') \leq K$ is NP-complete.*

The proof is as follows. The problem is clearly in NP. Given any repairing π', the conformance can be verified by checking the pre and post sets of each transition t in N_σ. Meanwhile, the repairing cost is computed by comparing the difference between $\pi'(t)$ and $\pi(t)$ of each t in $O(n)$ time.

To prove the NP-hardness of the repairing problem, we show a reduction from the set cover problem, which is one of Karp's 21 NP-complete problems [32]. By constructing a workflow specification and an inconsistent execution, we will show that there is a set cover C of size k if and only if the corresponding execution N_σ has a repairing π' with cost $\Delta(\pi, \pi') = K$, where $K = k + m$ and m is the size of the element set $\mathcal{U}$ in the set cover instance. To ensure the cost $\Delta(\pi, \pi') = m + k$ for the constructed specification and execution, for any two different transitions a, b, we have the confidence $conf(a) = 1$ and the frequency ratio $\frac{freq(a)}{freq(b)} = 1$, leading to the repairing cost $\delta(a, b) = 1$ in Equation (2). A total cost $\Delta(\pi, \pi') = K = m + k$ means that there are $m + k$ transitions repaired.

3.1 Transformation

Consider a set of m elements $\mathcal{U} = \{u_1, \ldots, u_m\}$, and n sets $\mathcal{S} = \{s_1, \ldots, s_n\}$, such that $s_i \subseteq \mathcal{U}$ and $\cup_i s_i = \mathcal{U}$. A set cover is a $C \subseteq \mathcal{S}$ of sets whose union is still $\mathcal{U}$. The set cover problem is to determine whether there exists a set cover C with a size no greater than k.

The transformation is conducted as follows.

(1) As illustrated in Figure 5(a), we construct a workflow specification $N_s(P_s, T_s, F_s)$. Let R be the first transition in T_s having $(\texttt{start}, R) \in F_s$. For each element $u_j \in \mathcal{U}, j = 1, \ldots, m$, we add a place $U_j \in P_s$ such that $(R, U_j) \in F_s$. Let

$$\mathcal{B} = \{B_{ji} \mid u_j \in s_i, j = 1, \ldots, m, i = 1, \ldots, n\},$$

be a set of intermediate transitions, whose cardinality is exactly $\sum_{i=1}^{n} |s_i|$. For each element u_j belonging to set s_i, we put an arc $(U_j, B_{ji}) \in F_s$.

Moreover, for each $s_i \in \mathcal{S}, i = 1, \ldots, n$, we add two corresponding transitions S_i, S_i' that point to the sink place, i.e., $(S_i, \texttt{end}) \in F_s, (S_i', \texttt{end}) \in F_s, i = 1, \ldots, n$. As explained later, S_i' is utilized to denote whether an S_i is repaired in the repairing problem. In addition, we introduce a new transition S_0 that connects to the sink place such that $(S_0, \texttt{end}) \in F_s$. Let

$$\mathcal{D} = \{D_i \mid i = 0, \ldots, n\},$$

be a set of intermediate places. The first intermediate place D_0 in the specification connects to S_0 and all the S_i', having $(D_0, S_0), (D_0, S_1'), \ldots, (D_0, S_n') \in F_s$. This D_0 is utilized to denote whether S_i is repaired in the repairing problem. Since D_0 only connects to S_0 and S_i', if the transition with labeling S_i has a place with labeling D_0 in its pre set, the labeling S_i of the transition should be repaired. For each remaining $D_i, i = 1, \ldots, n$, we add two arcs (D_i, S_i) and (D_i, S_i') to F_s.

The places and transitions of N_s are then given as

$$T_s = \{R, S_0, S_1, S_1', \ldots, S_n, S_n'\} \cup \mathcal{B},$$

$$P_s = \{\texttt{start}, U_1, \ldots, U_m, \texttt{end}\} \cup \mathcal{D}.$$

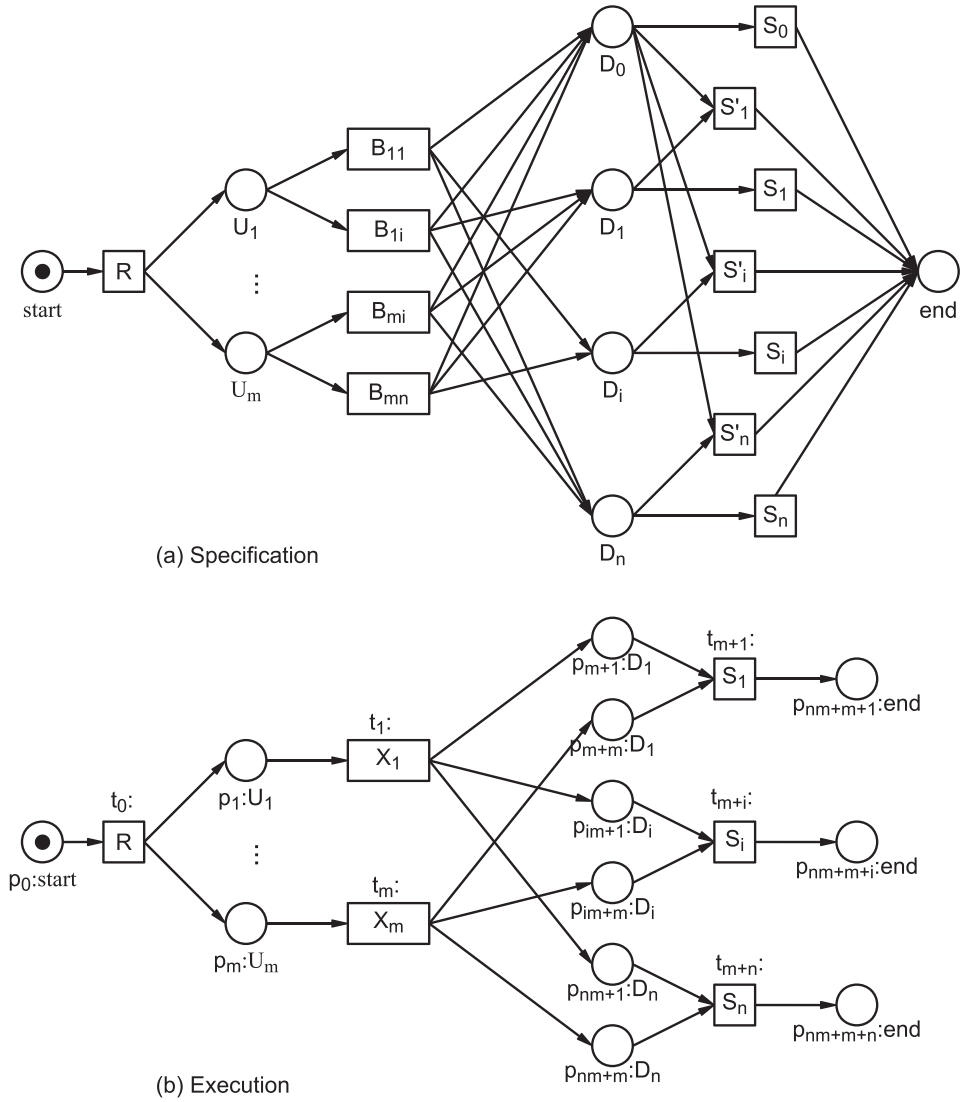


Fig. 5. Reduction from SET COVER problem.

Finally, for each intermediate transition $B_{ji} \in \mathcal{B}$, we insert at most n arcs to certain intermediate places, i.e., (B_{ji}, D_k) such that $k = 0, \ldots, n$ and $k \neq i$. In other words, for any $j = 1, \ldots, m$, it always has $(B_{ji}, D_i) \notin F_s$.

(2) We build a causal net of execution $N_\sigma(P_\sigma, T_\sigma, F_\sigma)$, as shown in Figure 5(b). Let $\pi(p_0) =$ `start`. Similarly, we put the first transition t_0 with arc (p_0, t_0) corresponding to $(\texttt{start}, R)$ in the specification N_s.

For each place $p_j : U_j, j = 1, \ldots, m$, there is an arc connecting from R, i.e., (t_0, p_j) corresponding to (R, U_j) in N_s. Let $\{X_1, \ldots, X_m\}$ be a set of intermediate transitions with no overlap to $\mathcal{B}$, i.e., $X_j \notin \mathcal{B}$. We add m arcs (p_j, t_j) mapping to (U_j, X_j) in N_s, $j = 1, \ldots, m$. It is easy to see that $t_j : X_j$ with $\text{pre}_{F_\sigma}(t_j) = \{U_j\}$ are violations, since the pre sets of $\pi(t_j) = X_j$ are not defined in the specification.

Next, for each transition $t_{m+i} : S_i, i = 1, \ldots, n$, we insert m places $p_{im+j} : D_i, j = 1, \ldots, m$ with corresponding arcs (p_{im+j}, t_{m+i}). Moreover, each $t_{m+i} : S_i$ points to an individual end place, $p_{nm+m+i} :$ end. Note that the additional transition S_0 is not included in the current execution.

Finally, for each intermediate transition $t_j : X_j, j = 1, \ldots, m$, we add n arcs to the corresponding intermediate places, i.e., $(t_j, p_{im+j}), i = 1, \ldots, n$. The transformation completes in polynomial time with $(n+1)(m+1)$ places and $(n+m+1)$ transitions in the causal net N_σ of execution trace, and $(m+n+3)$ places and at most $(nm+2n+2)$ transitions in the specification net N_s.

3.2 Proof

Without loss of generality, let the repairing cost be evaluated by the modification cost, having $\delta(a, b) = 1$ for any two different transitions a, b. With regard to the aforesaid transformation, we will show that there is a set cover C of size k if and only if N_σ has a repairing π' with cost $\Delta(\pi, \pi') = m + k$, where $k = K - m$.

First, let C be a set cover of size k. According to the set cover definition, for each element $u_j \in \mathcal{U}$, there must exist a set $s_i \in C$ such that $u_j \in s_i$. In the causal net N_σ, the corresponding transition $t_j : X_j$ with $\pi(\text{pre}_{F_\sigma}(t_j)) = \{U_j\}$ is then modified to $\pi'(t_j) = B_{ji}$ with cost 1, which has $\text{pre}_{F_s}(B_{ji}) = \{U_j\}$ according to the specification N_s. Note that for the n intermediate places connecting to t_j, it has $(t_j, p_{im+j}) \in F_\sigma$ but $(B_{ji}, D_i) \notin F_s$ in the currently modified execution (N_σ, π'). To eliminate such inconsistency, we can modify the place to $\pi'(p_{im+j}) = D_0$, such that $(B_{ji}, D_0) \in F_s$. Considering all the m elements in $\mathcal{U}$, the total relabeling cost in the first step is m. By this relabeling, i.e., changing the place p_{im+j} from D_i to D_0, a new inconsistency over $(p_{im+j}, t_{m+i}) \in F_\sigma$ is introduced, i.e., $D_0 \notin \text{pre}_{F_s}(S_i)$. Therefore, we further change the transition S_i to S_0 if $\pi'(\text{pre}_{F_\sigma}(t_{m+i})) = \{D_0\}$; otherwise, $\pi'(t_{m+i}) = S_i'$. The repairing cost is 1. For all the k sets in the set cover C, the total cost of the second relabeling step is k. Note that for all the $s_i \in \mathcal{S} \setminus C$, the corresponding places p_{im+j} will leave $p_{im+j} : D_i$ mapping by the original π unchanged in the repairing π'. Thus, the transition $t_{m+i} : S_i$ has no violation introduced. Consequently, we produce a π' of π such that $(N_\sigma, \pi') \vDash N_s$ and $\Delta(\pi, \pi') = m + k, k = K - m$.

Conversely, assume that we have a repairing π' with cost $\Delta(\pi, \pi') = m + k, k = K - m$. For each transition $t_j, j = 1, \ldots, m$, which currently maps to X_j, it must be modified from X_j to B_{ji} with cost 1, referring to the arcs between U_j and B_{ji} in F_s. Besides the above m transitions modified in π', the remaining repairing with cost k must come from transitions $t_{m+i}, i = 1, \ldots, n$, with $t_{m+i} : S_i$ in the original mapping π. Obviously, a transition t_{m+i} can only be modified from S_i to S_i' or S_0 with the cost 1. Let C be the set of transitions t_{m+i} that are modified by π'. We have $|C| = k$. By collecting all s_i that correspond to t_{m+i} in C, since all the elements u_j are covered, it forms a set cover C with size k.

4 EXACT ALGORITHM

Both detecting and repairing problems can be solved by an algorithm of attempting to find the minimum repair. If no valid repair is found, the input execution trace is detected as unsound structure. In this section, a practical branch and bound algorithm is developed for computing exact solutions. We also propose advanced bounding functions and pruning techniques to further improve the efficiency.

4.1 Branch and Bound

We first briefly describe the idea of computing repairs. For each transition in a given execution trace, there may be multiple candidates for repairing. In order to generate the optimal repair, we should theoretically consider all the repairing alternatives, each of which leads to a branch of generating possible repairs. The repairing must roll back to attempt the other branches in order to

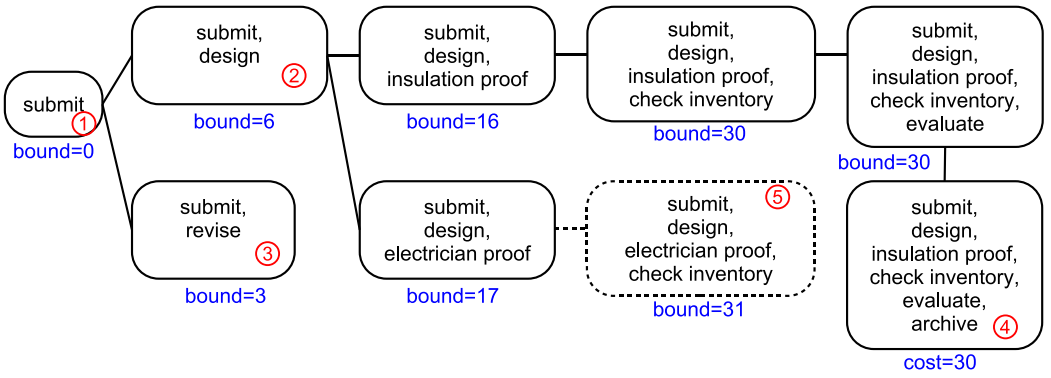


Fig. 6. Idea of branch and bound.

find the minimum cost one. Intuitively, by trying all the possible branches, we can find the exact solution.

4.1.1 Overview. Starting from the first transition in the execution trace σ, in each step, we will consider all the possible repairs for a transition t_k, each of which leads to a repairing branch.

The branching graph $BG = (V, E)$ consists of different σ_k, the first k transitions in σ that have been repaired by π'. Each node $v \in V$ in the graph represents a specific σ_k, while each edge $(v_1, v_2) \in E$ indicates that the repairing σ_k of the node v_2 is based on that of the node v_1. As we will present soon, a lower bound of least cost for repairing the remaining transitions in $\sigma \setminus \sigma_k$ can be computed, to form a valid repair. That is, we can compute a bound of repairing cost $LB(\sigma_k, \pi')$ for all the possible repairs generated in the branches w.r.t σ_k. A simple bounding function can be $LB(\sigma_k, \pi') = \Delta(\pi, \pi')$, i.e., the cost that has already been paid in the repairing π' for the first k transitions in the trace. It is clear that any repair over the entire trace generated in the branch of σ_k must have cost higher than $LB(\sigma_k, \pi')$.

Consequently, if we have found a valid repair in some other branches whose repairing cost is less than the lower bound $LB(\sigma_k, \pi')$, all the branches on σ_k can be safely pruned.

Example 6 (Example 1 Continued). Consider the execution trace in Figure 1. Each node in Figure 6 denotes a state of repairing the trace, i.e., $\pi'(\sigma_k)$. Initially, the first transition does not need to change, having $\pi(\sigma_1) = [\text{submit}]$ in node ①. For the next t_2, there are two possible repairs that lead to two branches $\pi(\sigma_2) = [\text{submit, design}]$ in node ② or [submit, revise] in node ③. The branching continues in the remaining transitions of σ until it forms a valid repair (e.g., node ④ for all 6 transitions in σ) or no further repairing can be applied such as node ③. Suppose that the repairing cost $\Delta(\pi, \pi')$ of node ④ is 30 (computed by string edit distance on event names). Then, all the branches on node ⑤ with bound 31 can be safely pruned.

4.1.2 Algorithm. Algorithm 1 presents the procedure of branch and bound repairing. We maintain a priority queue Q, where each element (σ_k, π') denotes a node or state of branching. As shown in Line 3, each step fetches an element from Q, say σ_{k-1} together with its repair π', which has the minimum $LB(\sigma_{k-1}, \pi')$. If the current π' has already formed a valid repair, in Line 6, we directly return it as the result. As the remaining nodes in Q must have a lower bound no less than the current solution, the result is the first valid solution with the minimum cost.

Otherwise, we keep on branching to the next transition t_k. According to Proposition 1, it ensures that all the prerequisites of t_k are in σ_{k-1}.

ALGORITHM 1: ExactBB(N_σ, π, N_s)

Input: An execution (N_σ, π) and a specification N_s
Output: An optimal repair π' with the minimum repairing cost such that $(N_\sigma, \pi') \vDash N_s$

1: $Q := \{(\emptyset, \pi)\}$
2: **while** $Q \neq \emptyset$ **do**
3: $(\sigma_{k-1}, \pi') := \arg\min_{(\sigma_i, \pi') \in Q} LB(\sigma_i, \pi')$
4: $Q := Q \setminus \{(\sigma_{k-1}, \pi')\}$
5: **if** $(N_\sigma, \pi') \vDash N_s$ **then**
6: **return** π'
7: **else**
8: $t_k := \sigma(k)$ the kth transition in the execution trace {branch t_k to generate $\pi'(t_k)$}
9: $\sigma_k := \sigma_{k-1} \cup \{t_k\}$
10: **for** each $p_i \in \text{pre}_{F_\sigma}(t_k)$ **do**
11: $P_i^c :=$ all valid labeling $\pi'(p_i)$ of p_i.
12: $\Lambda := P_1^c \times \cdots \times P_{|\text{pre}_{F_\sigma}(t_k)|}^c$
13: **for** each labeling π' in Λ on places $\text{pre}_{F_\sigma}(t_k)$ **do**
14: $T_c := \cap_{p_i \in \text{pre}_{F_\sigma}(t_k)} \text{post}_{F_s}(\pi'(p_i))$
15: **for** each $X \in T_c$ **do**
16: **if** $\pi'(\text{pre}_{F_\sigma}(t_k)) = \text{pre}_{F_s}(X)$ **then**
17: $\pi'(t_k) := X$
18: $Q := Q \cup \{(\sigma_k, \pi')\}$
19: **return** unsound structure

Lemma 3. *For the current branching for each t_k, it always satisfies* $\text{pre}_{F_\sigma}(\text{pre}_{F_\sigma}(t_k)) \subseteq \sigma_{k-1}$.

Proof. Suppose that there is an event $t_i \in \text{pre}_{F_\sigma}(\text{pre}_{F_\sigma}(t_k))$ and $t_i \notin \sigma_{k-1}$. Since σ_{k-1} denotes the first $k-1$ events in σ that have been repaired, and $t_i \notin \sigma_{k-1}, i \neq k$, we have $i > k$. According to Proposition 1, we have $t_i \notin \text{pre}_{F_\sigma}(\text{pre}_{F_\sigma}(t_k))$ with $k < i$, which is a contradiction. □

That is, the transitions in σ_{k-1} have already been repaired and will not be modified in the current branching. As illustrated in Figure 7, the prerequisites of t_k determine the possible assignments of places in $\text{pre}_{F_\sigma}(t_k)$, i.e., Lines 10–12 in Algorithm 1. The determination of $\pi'(p_i)$ for each $p_i \in \text{pre}_{F_\sigma}(t_k)$ will be presented below. Consequently, for each labeling π' on places in $\text{pre}_{F_\sigma}(t_k)$, we can enumerate the corresponding possible repairs (Line 14) of t_k for branching (Line 18).

Finally, the **while** iteration terminates when there is no element left in Q. The returned results can be either the optimal repair or the identification of an unsound structures. The correctness of the conformance of the returned repair is guaranteed by Line 5 in Algorithm 1.

4.2 Generating Branches

Recall that each branch w.r.t. the current transition t_k corresponds to a possible repairing $\pi'(t_k)$. As illustrated in Figure 7, to determine $\pi'(t_k)$, we need to first identify the labeling on the places in the pre set of t_k.

Let us consider any $p_i \in \text{pre}_{F_\sigma}(t_k)$. Referring to the definition of the causal net, we have a unique transition, say t_i, in the pre set of p_i, denoted as $\text{pre}_{F_\sigma}(p_i) = \{t_i\}$. This t_i must belong to σ_{k-1} according to Lemma 3, where the repair $\pi'(t_i)$ has been given. As illustrated in Line 11 in Algorithm 1, we can find a set P_i^c of all valid labeling $\pi'(p_i)$ of p_i that are consistent with $\pi'(t_i)$. There are several scenarios to consider for determining P_i^c:

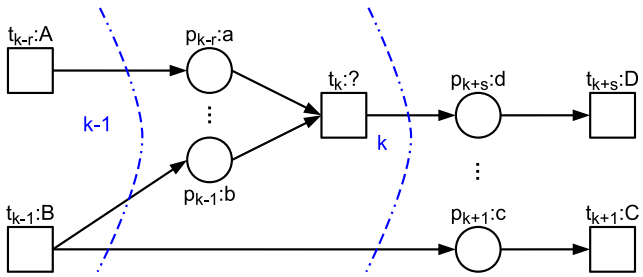


Fig. 7. Generating a branch.

Case 1. If $\text{post}_{F_\sigma}(\text{post}_{F_\sigma}(t_i)) \not\subseteq \sigma_k$, then we have $P_i^c := \text{post}_{F_s}(\pi'(t_i))$. That is, there exists at least one transition, whose prerequisite is t_i, but not belonging to σ_k, e.g., t_{k+1} following t_{k-1} in Figure 7 that has not been repaired in (σ_{k-1}, π'). We can assign any $\pi'(p_i)$ in $\text{post}_{F_s}(\pi'(t_i))$ without introducing inconsistencies to t_i in the current stage.

Example 7. Consider a part of execution in Figure 8(b) and its related specification in Figure 8(a). Let t_1 in Figure 8(b) be the currently considered kth transition, we have $\text{pre}_{F_\sigma}(t_1) = \{p_1\}$, and $\text{pre}_{F_\sigma}(p_1) = \{t_0\}$. Since there exist two transitions (t_2 and t_3), whose prerequisite is t_0, but not belong to σ_k, i.e., $\text{post}_{F_\sigma}(\text{post}_{F_\sigma}(t_0)) = \{t_1, t_2, t_3\} \not\subseteq \sigma_k$, we can assign any $\pi'(p_1)$ in $P_1^c = \text{post}_{F_s}(\pi'(t_0)) = \{a, b, c\}$ without introducing inconsistencies to t_0 in the current stage.

Case 2.1. If $\text{post}_{F_\sigma}(\text{post}_{F_\sigma}(t_i)) \subseteq \sigma_k$, and

$$\pi'(\text{post}_{F_\sigma}(t_i) \setminus \{p_i\}) = \text{post}_{F_s}(\pi'(t_i)),$$

then we have $P_i^c := \text{post}_{F_s}(\pi'(t_i))$. In this (and following 2.x) case, we have all the transitions, whose prerequisite is t_i, belonging to σ_k. In other words, all the transitions, e.g., $\text{post}_{F_\sigma}(\text{post}_{F_\sigma}(t_{k-r}))$ of t_{k-r} in Figure 7, are repaired in (σ_{k-1}, π') except t_k. Moreover, the condition $\pi'(\text{post}_{F_\sigma}(t_i) \setminus \{p_i\}) = \text{post}_{F_s}(\pi'(t_i))$ ensures the conformance on t_i if we ignore p_i. Consequently, any assignment $\pi'(p_i)$ in $\text{post}_{F_s}(\pi'(t_i))$ will not introduce inconsistencies to t_i.

Example 8. Consider again the specification in Figure 8(a). Let t_4 in Figure 8(c) be the currently considered kth transition, we have $\text{pre}_{F_\sigma}(t_4) = \{p_4\}$, and $\text{pre}_{F_\sigma}(p_4) = \{t_0\}$. All the transitions whose prerequisite is t_0 belong to σ_k, i.e., $\text{post}_{F_\sigma}(\text{post}_{F_\sigma}(t_0)) = \{t_1, t_2, t_3, t_4\} \subseteq \sigma_k$. Since $\pi'(\text{post}_{F_\sigma}(t_0) \setminus \{p_3\}) = \{a, b, c\} = \text{post}_{F_s}(\pi'(t_0)) = \{a, b, c\}$, we can assign any $\pi'(p_4)$ in $P_4^c = \text{post}_{F_s}(\pi'(t_0)) = \{a, b, c\}$ without introducing inconsistencies to t_0. This is the scenario of multiple instances [45], i.e., a task may be submitted multiple times as duplicates.

Case 2.2. If $\text{post}_{F_\sigma}(\text{post}_{F_\sigma}(t_i)) \subseteq \sigma_k$, and

$$|\pi'(\text{post}_{F_\sigma}(t_i) \setminus \{p_i\})| = |\text{post}_{F_s}(\pi'(t_i))| - 1,$$

then we have $P_i^c := \text{post}_{F_s}(\pi'(t_i)) \setminus \pi'(\text{post}_{F_\sigma}(t_i) \setminus \{p_i\})$. This case differs from Case 2.1 in the variance between $\pi'(\text{post}_{F_\sigma}(t_i) \setminus \{p_i\})$ and $\text{post}_{F_s}(\pi'(t_i))$. It states that there is only one choice of $\pi'(p_i)$, i.e., $\text{post}_{F_s}(\pi'(t_i)) \setminus \pi'(\text{post}_{F_\sigma}(t_i) \setminus \{p_i\})$, in order to make the conformance on t_i.

Example 9. Let t_3 in Figure 8(d) be the currently considered kth transition, we have $\text{pre}_{F_\sigma}(t_3) = \{p_3\}$, and $\text{pre}_{F_\sigma}(p_3) = \{t_0\}$. All the transitions whose prerequisite is t_0 belong to σ_k, that is, $\text{post}_{F_\sigma}(\text{post}_{F_\sigma}(t_0)) = \{t_1, t_2, t_3\} \subseteq \sigma_k$. Compared to the related specification in Figure 8(a), since the only difference between $\pi'(\text{post}_{F_\sigma}(t_0) \setminus \{p_3\}) = \{a, b\}$ and $\text{post}_{F_s}(\pi'(t_0)) = \{a, b, c\}$ is $\{c\}$, there is only one choice for $\pi'(p_3)$, i.e., $P_3^c = \text{post}_{F_s}(\pi'(t_0)) \setminus \pi'(\text{post}_{F_\sigma}(t_0) \setminus \{p_3\}) = \{c\}$.

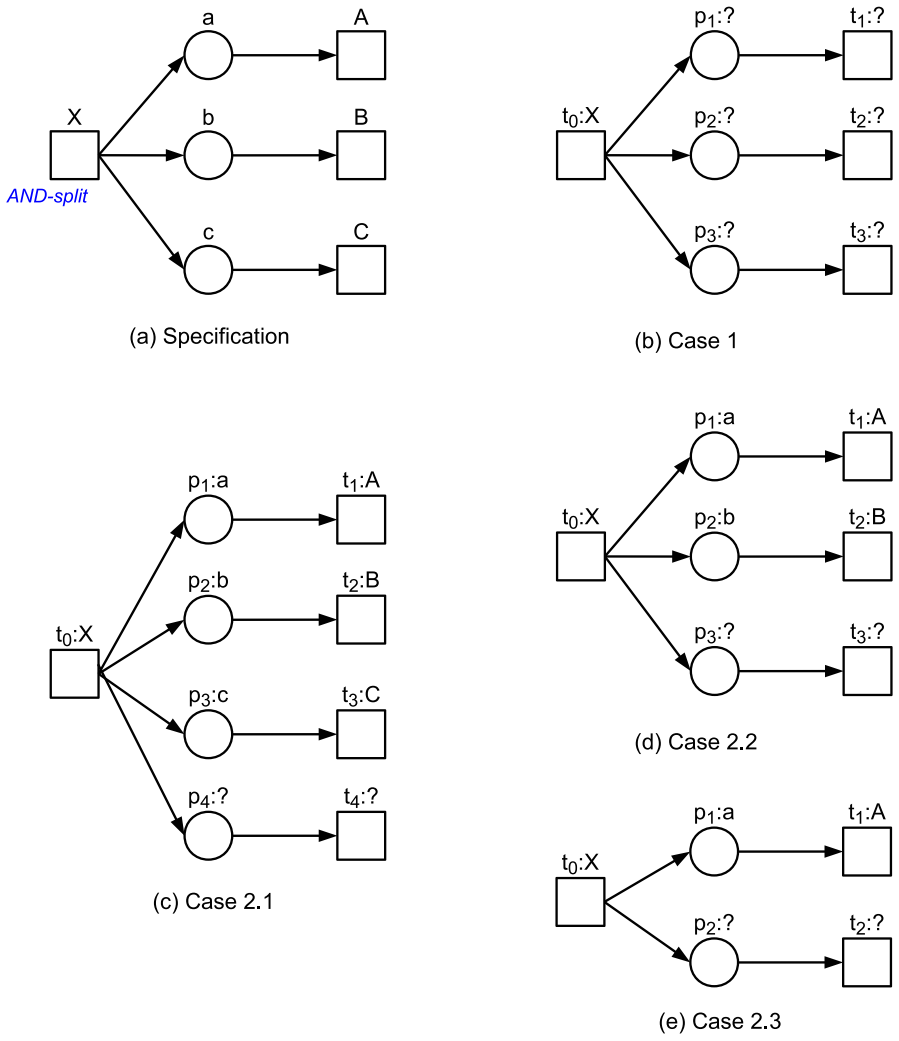


Fig. 8. Example of generating a branch.

Case 2.3. If $\text{post}_{F_\sigma}(\text{pre}_{F_\sigma}(p_i)) \subseteq \sigma_k$, and

$$|\text{post}_{F_s}(\pi'(t_i))| - |\pi'(\text{post}_{F_\sigma}(t_i) \setminus \{p_i\})| > 1,$$

then we have $P_i^c := \emptyset$. In this case, the difference between $\pi'(\text{post}_{F_\sigma}(t_i) \setminus \{p_i\})$ and $\text{post}_{F_s}(\pi'(t_i))$ is at least 2. It is impossible to achieve the conformance on t_i by simply repairing one place p_i. We ignore this case by setting $P_i^c := \emptyset$.

Example 10. Consider the execution in Figure 8(e), and let t_2 in Figure 8(e) be the currently considered kth transition. We have $\text{pre}_{F_\sigma}(t_2) = \{p_2\}$, and $\text{pre}_{F_\sigma}(p_2) = \{t_0\}$. All the transitions whose prerequisite is t_0 belong to σ_k, i.e., $\text{post}_{F_\sigma}(\text{post}_{F_\sigma}(t_0)) = \{t_1, t_2\} \subseteq \sigma_k$. Since the difference between $\pi'(\text{post}_{F_\sigma}(t_0) \setminus \{p_2\}) = \{a\}$ and $\text{post}_{F_s}(\pi'(t_0)) = \{a, b, c\}$ is larger than 1, it is impossible to achieve the conformance on t_0 by simply repairing one place p_2. As a result, we have $P_2^c = \emptyset$.

Thus far, we have presented the assignment of each place p_i in $\text{pre}_{F_\sigma}(t_k)$. Considering all the $r = |\text{pre}_{F_\sigma}(t_k)|$ places, we can enumerate all the labeling π' on places $\text{pre}_{F_\sigma}(t_k)$, i.e., $\Lambda := P_1^c \times \cdots \times P^c_{|\text{pre}_{F_\sigma}(t_k)|}$ in Line 12 in Algorithm 1. For each labeling π' in Λ, there is a set of candidate repairs for $\pi'(t_k)$, denoted by $T_c := \cap_{p_i \in \text{pre}_{F_\sigma}(t_k)} \text{post}_{F_s}(\pi'(p_i))$ in Line 14. Finally, any $\pi'(t_k)$ in T_c, which satisfies the conformance requirement of $\pi'(\text{pre}_{F_\sigma}(t_k)) = \text{pre}_{F_s}(\pi'(t_k))$, generates a possible branch (σ_k, π'), and is added into Q in Line 18.

Example 11 (Example 1 Continued). Let t_5 in Figure 1 be the currently considered kth transition in Line 8 of Algorithm 1. Line 11 computes all valid labeling (w.r.t. prerequisites of t_5) for the places in $\text{pre}_{F_\sigma}(t_5)$, by considering the aforesaid possible cases, i.e., {c} for p_3 referring to $\pi'(t_3)$ = insulation proof and {e} for p_5 referring to $\pi'(t_4)$ = check inventory. The labeling of places p_3, p_5 suggests possible candidates for branching t_5, in Line 14, having $T_c = \text{post}_{F_s}(\text{c}) \cap \text{post}_{F_s}(\text{e}) =$ {evaluate}. By considering the next branching step iteratively, on t_6, since there is no violation left, the program returns the result in Line 6.

Algorithm Analysis. Note that pre and post sets of a transition lead to parallel flows. In most processes, the number of parallel flows of a transition is often small and can be regarded as a constant[3] [35]. Let b and d be the maximum sizes of the pre/post set of any node in the specification and execution, respectively. We have $O(b^d)$ possible labelings in Λ, each of which corresponds to b repairing candidates in T_c, i.e., total $O(b^{d+1})$ repairs for t_k. Consider the branches of possible combinations on n transitions. The worst-case complexity of Algorithm 1 is $O(b^{(d+1)n})$, exponential to n.

4.3 Pruning Invalid Branches

It is worth noting that not all the branches can eventually generate a valid repair (e.g., node ③ in Figure 6). We call the branches that cannot form a valid repair *invalid branches*. The earlier the algorithm could identify invalid branches, the better the repairing performance will be. However, the aforesaid repairing method will not terminate branching until the last step, i.e., no further repairing can be performed on a transition.

The intuition of early termination for invalid branches comes from the scenario of unsound structure. If the maximum length path from the current transition t_k to the end place in the causal net is shorter than the minimum length path from $\pi(t_k)$ to end in the specification, modifying transitions after t_k will form an invalid repair.

Pruning of invalid branches can be deployed before Line 18 in Algorithm 1. Intuitively, in the preprocessing, for each transition t_j in the specification, we can find a shortest path from t_j to end, denoted by $\text{SP}_s(t_j)$. In the causal net of execution, the longest path from any transition t_i to end, say $\text{LP}_\sigma(t_i)$, can be computed by running a shortest-path finding algorithm with negative weights, which are obtained by the additive inverse (opposite number) of the original weights [6]. The labeling $\pi'(t_k)$ having $\text{LP}_\sigma(t_k) < \text{SP}_s(\pi'(t_k))$, i.e., the longest path in execution is shorter than the shortest path in the specification, is not valid for the current transition t_k. In other words, it is impossible to find a valid repair for such a case.

Proposition 4. *A branch (σ_k, π') with $\text{LP}_\sigma(t_k) < \text{SP}_s(\pi'(t_k))$ is an invalid branch that cannot form any valid repair with the current labeling π' on σ_k.*

[3] Although a process may theoretically consist of a large number of parallel flows, in practice, techniques are often applied to keep the process as simple as possible, such as minimize the routing paths per element [35]. According to the survey, the maximum number of parallel flow is 4 in the dataset from SAP Reference Models [12], which includes 69 typical workflow specifications.

Proof. Since $\text{LP}_\sigma(t_k) < \text{SP}_s(\pi'(t_k))$, the number of remaining transitions that have not been repaired in the branch is less than the least number of transitions to reach the end in the specification. It is easy to see that the branch cannot generate any repair that will conform to the specification, which makes it an invalid branch. □

According to the proposition, for any t_k in Line 18 in Algorithm 1, if $\text{LP}_\sigma(t_k) < \text{SP}_s(\pi'(t_k))$, we will not add this (σ_k, π') to Q. That is, (σ_k, π') is pruned as an invalid branch.

Example 12 (Example 4 Continued). Consider again the causal net of execution in Figure 4. Let t_1 be the current transition with labeling $\pi'(t_1) = \text{submit}$. By interpreting places as edges connecting two transitions, the causal net of execution is indeed a directed acyclic graph of transitions. We assign a negative weight -1 for each edge, and run the shortest-path finding algorithm with negative weights. It returns a shortest-path $\langle p_1, p_2 \rangle$ with negative weight -2, from the transition t_1 to end, which corresponds to the longest path with positive weight 1 for each edge, having $\text{LP}_\sigma(t_1) = 2$. Moreover, in the specification in Figure 1(c), we can find a shortest path from submit to end, i.e., $\langle$a, d, e, s$\rangle$, having $\text{SP}_s(\text{submit}) = 4$. Since $\text{LP}_\sigma(t_1) = 2 < \text{SP}_s(\text{submit}) = 4$, i.e., the longest path in execution is shorter than the shortest path in the specification, it is impossible to find a valid repair for the labeling $\pi'(t_1) = \text{submit}$.

4.4 Advanced Bounding Function

The lower bound of repairing cost $LB(\sigma_i, \pi')$ is essential in pruning branches. Before introducing the advanced bounding function LB, we first investigate the lower bound of cost for repairing an execution. Let $LC(N_\sigma, \pi)$ denote the least cost of repairing (N_σ, π). As mentioned, a naive bound is $LC(N_\sigma, \pi) = 0$, as any repair π' must have $\Delta(\pi, \pi') \geq 0$. Indeed, as discussed below, such a naive bound will yield a bounding function LB with weaker pruning power.

To obtain a reasonable bound of least cost for repairing (N_σ, π), we build a conflict graph G with transitions in T_σ as vertexes. For any place $p \in P_\sigma$, let $\text{pre}_{F_\sigma}(p) = \{t_i\}$ and $\text{post}_{F_\sigma}(p) = \{t_j\}$. If $\text{post}_{F_s}\pi(t_i) \cap \text{pre}_{F_s}\pi(t_j) = \emptyset$, i.e., at least one of the transitions t_i, t_j needs to be repaired, we put a conflict edge (t_i, t_j) in G. Each vertex t_i is associated with a weight, $w(t_i) = \min_{x \in T_s} \delta(\pi(t_i), x)$, i.e., the minimum cost on all possible repairs of t_i.

To eliminate inconsistencies, at least one transition of each edge in G should be repaired. The minimum weighted vertex cover of G with total weight $VC^*(G)$ can be interpreted as a lower bound of least cost $LC(N_\sigma, \pi)$, i.e., $VC^*(G) \leq \Delta(\pi, \pi')$ for any repair π'. As computing the exact minimum vertex cover is unlikely to be efficient, we relax the bound as follows. Consider a set $E = \emptyset$ initially. We repeatedly add an edge say (t_i, t_j) of G into E, and remove t_i, t_j, and all the edges incident on t_i or t_j, until there is no edge left in G. Consequently, no two edges in E share the same vertex. As each edge should be covered by at least one vertex from the minimum vertex cover, we have $\sum_{(t_i,t_j)\in E} \min\{w(t_i), w(t_j)\} \leq VC^*(G)$. Considering the relationship between vertex cover and repairing, it follows:

Lemma 5. *For any valid repair π', we have*

$$\sum_{(t_i,t_j)\in E} \min\{w(t_i), w(t_j)\} \leq VC^*(G) \leq \Delta(\pi, \pi').$$

Proof. Firstly, although the minimum weighted vertex cover of G contains at least one transition of each edge in G, it cannot ensure the conformance of the repairs for all the transitions. It is likely that conflicts still exist after repairing the transitions in the minimum weight vertex cover, and therefore requires further repair and more cost. As a result, we have $VC^*(G) \leq \Delta(\pi, \pi')$. Secondly, according to the generation of set E, the edges in E will not overlap in vertexes, i.e., each

vertex appears at most once. The edges in E also exist in G, and should be covered by the vertexes in the minimum weighted vertex cover $VC^*(G)$. Note that $\sum_{(t_i,t_j)\in E} \min\{w(t_i), w(t_j)\}$, considering the smaller weight of two vertexes for each edge in E, denotes a lower bound of the cost to cover the edges in E. Since there are more edges that the minimum weighted vertex cover of G needs to cover, the lower bound of covering E also applies to G, having $\sum_{(t_i,t_j)\in E} \min\{w(t_i), w(t_j)\} \leq VC^*(G)$. □

Hence, we define the lower bound of the least cost for repairing by

$$LC(N_\sigma, \pi) = \sum_{(t_i,t_j)\in E} \min\{w(t_i), w(t_j)\}.$$

Note that each (σ_i, π') divides the transitions into two parts, σ_i and its complement $\sigma \setminus \sigma_i$, denoted as $\bar{\sigma}_i$. We consider $N_{\bar{\sigma}_i}(P_{\bar{\sigma}_i}, T_{\bar{\sigma}_i}, F_{\bar{\sigma}_i})$ as a projection or partition of the net on transitions $T_{\bar{\sigma}_i} \subseteq T_\sigma$ corresponding to the remaining execution trace $\bar{\sigma}_i$. As π' only specifies the repairing of the current σ_i, transitions in $\bar{\sigma}_i$ have not been reassigned by π' yet.

Lemma 6. *We have* $\pi'(t) = \pi(t), \forall t \in \sigma \setminus \sigma_i$.

Proof. According to the definition of (σ_i, π') in Algorithm 1, it is easy to see that the assignment of the transition in the remaining execution trace $\bar{\sigma}_i$ is the same as its original labeling π, i.e., $\pi'(t) = \pi(t)$. □

Finally, the lower bound is defined as

$$LB(\sigma_i, \pi') = \Delta(\pi, \pi') + LC(N_{\bar{\sigma}_i}, \pi),$$

which consists of the repairing cost $\Delta(\pi, \pi')$ that has been made on σ_i, and the least cost of repairing the remaining $\bar{\sigma}_i$. The larger the lower bound is, the higher the power will be in pruning branches. We call this $LB(\sigma_i, \pi')$ with $LC(N_\sigma, \pi) = \sum_{(t_i,t_j)\in E} \min\{w(t_i), w(t_j)\}$ the *advanced bounding function*. It is not surprising that the aforesaid simple bounding function with the naive bound $LC(N_{\bar{\sigma}_i}, \pi) = 0$ shows weaker pruning power.

Example 13 (Example 1 Continued). Let σ_1 with one transition t_1 in Figure 1 be the currently repaired transitions. Since no transition is changed so far, we have $\Delta(\pi, \pi') = 0$. For the remaining transitions t_2, t_3, t_4, t_5, t_6, i.e., $\bar{\sigma}_1$, a conflict graph is constructed with edges (t_2, t_3), (t_2, t_4), (t_3, t_5), (t_4, t_5). Suppose that (t_2, t_3) and (t_4, t_5) are chosen to E for Lemma 5, and t_2, t_5 has smaller minimum cost, say $w(t_2) = 3$ and $w(t_5) = 0$. By removing (t_2, t_3) and (t_4, t_5), there is no edge left in the conflict graph. We have $LB(\sigma_1, \pi') = LC(N_{\bar{\sigma}_1}, \pi) = w(t_2) + w(t_5) = 3$ higher than the simple bound 0.

It is notable that an annotation/labeling may be verified to be correct/mandatory, or fixed, and thus do not allowed to be changed in some repairing scenarios, which is common in processes in manufacturing/medical research. The proposed algorithm can easily extend to such scenarios by setting a very large confidence (e.g., infinity) in Equation (2) for such annotation/labeling. This large confidence $conf(t)$ leads to a very large repair cost $\delta(\pi(t), \pi'(t))$, and thus will not be considered and returned as the minimum repair. In this sense, the fixed annotation/labeling will not be changed.

5 TRANSITION ORIENTED HEURISTICS

Although several advanced pruning techniques are proposed, the exact algorithm is still too costly to meet the fast repairing requirement, such as in online/streaming systems. As event data are continuously generated, the online repairing may only allow one pass through the events (transitions) in executions. In this section, to support fast repairing, we introduce several heuristics for approximation and present a transition-oriented one-pass algorithm.

The idea of approximate repairing is to repair one transition at a time, and repeat until all violations are eliminated or no repairing can be further conducted. As each step of repairing a transition may introduce inconsistencies to others, we heuristically choose a revision that will have the least violations to others. Let us first investigate this intuition on how to repair one transition regarding violation elimination.

5.1 Heuristic for Repairing a Transition

Consider any transition $t_k \in T_\sigma$ in an execution (N_σ, π) whose current labeling $\pi(t_k)$ is inconsistent with the specification N_s, that is, having either $\pi(\text{pre}_{F_\sigma}(t_k)) \neq \text{pre}_{F_s}(\pi(t_k))$ or $\pi(\text{post}_{F_\sigma}(t_k)) \neq \text{post}_{F_s}(\pi(t_k))$. The repairing is to find a new labeling $\pi'(t_k)$ that can eliminate the inconsistency to t_k. We consider possible candidates for repairing t_k. Intuitively, in order to reduce the repairing cost, we prefer the repairing of t_k with least inconsistencies introduced to other transitions.

Given any labeling π', we define the number of violations to a transition t_k as follows. Recall that any place p in a causal net always has $|\text{pre}_F(p)| \leq 1$ and $|\text{post}_F(p)| \leq 1$. For any $p_i \in \text{pre}_{F_\sigma}(t_k)$, we have either $\text{pre}_F(p_i) = \emptyset$ ($\pi'(p_i) = \text{start}$) or $\text{pre}_F(p_i) = \{t_i\}$, a unique transition (prerequisite) in the pre set of p_i. We count place $p_i \in \text{pre}_{F_\sigma}(t_k)$ as one violation to $\pi'(t_k)$ if $\pi'(p_i) \notin \text{post}_{F_s}(\pi'(t_i))$. For the case of $\text{pre}_F(p_i) = \emptyset$, $\pi'(p_i)$ can only be mapped to start with no violation introduced. By considering the symmetric violations to the post set of t_k, the total violation count introduced by π' on t_k is given by

$$\begin{aligned}\tau(t_k, \pi') = &|\{p_i \in \text{pre}_{F_\sigma}(t_k) \mid \pi'(p_i) \notin \text{post}_{F_s}(\pi'(t_i)), \pi'(p_i) \neq \text{start}\}| \\ &+ |\{p_j \in \text{post}_{F_\sigma}(t_k) \mid \pi'(p_j) \notin \text{pre}_{F_s}(\pi'(t_j)), \pi'(p_j) \neq \text{end}\}|.\end{aligned}$$

Therefore, we need to find a π' such that $\tau(t_k, \pi')$ is minimized. If $\tau(t_k, \pi') = 0$, the repair π' is a perfect repairing without introducing any new inconsistencies to others.

Example 14 (Example 1 Continued). Consider t_2 in Figure 1 with $\pi(t_2) = \text{do revise}$. For $p_1 \in \text{pre}_{F_\sigma}(t_2)$, we cannot find any labeling $\pi'(p_1)$ such that $\pi'(p_1) \in \text{pre}_{F_s}(\pi'(t_2))$. Thereby, p_1 is counted as a violation towards $\pi(t_2)$. Similarly, p_2 and p_4 are also counted as having $\tau(t_2, \pi) = 3$. Suppose that t_2 is repaired by $\pi'(t_2) = \text{design}$. We can assign $\pi'(p_1) = \text{a}$ which belongs to $\text{pre}_{F_s}(\pi'(t_2))$, i.e., p_1 is no longer a violation. It reduces the violation count of t_2 to $\tau(t_2, \pi) = 2$.

5.2 One Pass Algorithm

We present a one pass algorithm of repairing one transition at a time from start to end in the execution trace σ. In each step, we determine the repair $\pi'(t_k)$ of a transition $t_k, k = 1, \ldots, |\sigma|$, and its corresponding $\pi'(p_j)$ of $p_j \in \text{post}_{F_\sigma}(t_k)$.

Following the order of execution trace, we show that $\pi'(p_i)$ of all places $p_i \in \text{pre}_{F_\sigma}(t_k)$ must have been assigned. According to Proposition 1, all the prerequisite transitions of the current t_k, say $t_i \in \text{pre}_{F_\sigma}(\text{pre}_{F_\sigma}(t_k))$, should have been repaired. When previously repairing t_i, the corresponding $p_i \in \text{post}_{F_\sigma}(t_i)$, having $\text{post}_{F_\sigma}(p_i) = \{t_k\}$, is determined.

Initially, only one place is processed in the causal net, i.e., the start place. In each step of t_k, as shown in Figure 9, all the places $p_{k-r}, \ldots, p_{k-1}$ in the pre set of t_k are already determined. After repairing the transition t_k (if necessary), we assign all the places $p_{k+1}, \ldots, p_{k+s}$ in the post set of t_k. Finally, the program terminates when it reaches the last transition in the execution trace.

Algorithm 2 presents the pseudo-code of one pass repairing. As illustrated in Line 1, we start from the first transition $t_1 = \sigma(1)$ directly following the start place. In each iteration, Line 5 selects a transition t_k, i.e., the kth transition $\sigma(k)$ in the execution trace σ. If there is no inconsistency with respect to t_k, we directly move to the next transition (Line 23); otherwise, t_k needs to be repaired (Lines 7–22). As discussed, all the places in the pre set of t_k must have been recovered (initially,

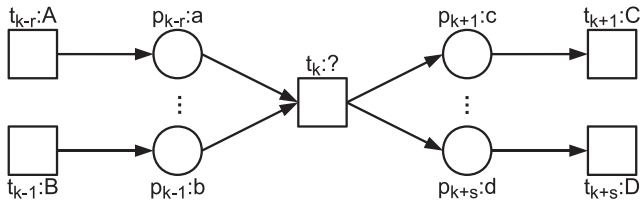


Fig. 9. Repairing one transition t_k.

ALGORITHM 2: OnePass(N_σ, π, N_s)

Input: An execution (N_σ, π) and a specification N_s
Output: A repair π' such that $(N_\sigma, \pi') \vDash N_s$

```
1:  k := 1, π' := π
2:  while (N_σ, π') ⊭ N_s do
3:    if k > |T_σ| then
4:      return unsound structure
5:    t_k := σ(k) the k-th transition in the execution trace
6:    if π'(pre_{F_σ}(t_k)) ≠ pre_{F_s}(π(t_k)) or π(post_{F_σ}(t_k)) ≠ post_{F_s}(π(t_k)) then
7:      T_c := ∩_{p_i ∈ pre_{F_σ}(t_k)} post_{F_s}(π'(p_i))
8:      τ_min := a large positive integer d
9:      for each x ∈ T_c do
10:       if π'(pre_{F_σ}(t_k)) = pre_{F_s}(x) then
11:         π'(t_k) := x
12:         for each complete labeling π_p : post_{F_σ}(t_k) → post_{F_s}(x) do
13:           π'(post_{F_σ}(t_k)) := π_p(post_{F_σ}(t_k))
14:           if τ(t_k, π') < τ_min then
15:             τ_min := τ(t_k, π')
16:             π_min(t_k) := π'(t_k)
17:             π_min(post_{F_σ}(t_k)) := π'(post_{F_σ}(t_k))
18:     if τ_min equals to the original value d then
19:       return unsound structure
20:     else
21:       π'(t_k) := π_min(t_k)
22:       π'(post_{F_σ}(t_k)) := π_min(post_{F_σ}(t_k))
23:   k++
24: return π'
```

the start place in $\text{pre}_{F_\sigma}(t_1)$ leaves unchanged). Hence, the repairing is to determine two aspects: $\pi'(t_k)$ in Line 11 and $\pi'(\text{post}_{F_\sigma}(t_k))$ in Line 13. Possible candidates for these two aspects will be discussed soon. $\tau_{\min}$ in Line 15 records the repairing π' with the minimum $\tau(t_k, \pi')$, i.e., the minimum violations introduced by repairing t_k.

Correctness of conformance in the returned π' is ensured by the condition for each transition on pre set in Line 10 and the complete labeling (defined below) for post set in Line 12.

5.2.1 Candidates for Transition $\pi'(t_k)$. Since the places in $p_i \in \text{pre}_{F_\sigma}(t_k)$ have already been determined, we can only choose candidates for repairing t_k without introducing any inconsistency to p_i. For $\pi'(p_i)$ of each $p_i \in \text{pre}_{F_\sigma}(t_k)$, we can find a set of valid post transition, $\text{post}_{F_s}(\pi'(p_i))$,

in the specification N_s. A candidate x appearing as the valid post transition of all $\pi'(p_i)$ can be a possible repairing of t_k, which will be consistent with all p_i. Hence, the candidates for repairing t_k are given by $T_c := \cap_{p_i \in \text{pre}_{F_\sigma}(t_k)} \text{post}_{F_s}(\pi'(p_i))$ as illustrated in Line 7 in Algorithm 2.

Example 15 (Example 1 Continued). Consider the transition t_2 in Figure 1(b), p_1 is the only place in the pre set of t_2, i.e., $\text{pre}_{F_\sigma}(t_2) = \{p_1\}$. Suppose that $\pi'(p_1) = \text{a}$ has already been determined, the candidates T_c for repairing t_2 should be chosen without introducing inconsistency to p_1. As a result, we have $T_c := \cap_{p_i \in \text{pre}_{F_\sigma}(t_2)} \text{post}_{F_s}(\pi'(p_i)) = \text{post}_{F_s}(\pi'(p_1)) = \{\text{design}, \text{revise}\}$.

5.2.2 Candidates for Places $\pi'(\text{post}_{F_\sigma}(t_k))$. Next, given a candidate $x \in T_c$ for $\pi'(t_k)$, we aim at determining the assignment of places $p_j \in \text{post}_{F_\sigma}(t_k)$ such that $\tau(t_k, \pi')$ is minimized. Again, the assignment of p_j should be consistent with t_k. For a fixed $\pi'(t_k) = x$, it is equivalent to find a labeling from $\text{post}_{F_\sigma}(t_k)$ to $\text{post}_{F_s}(x)$, denoted by π_p. We say a labeling $\pi_p : \text{post}_{F_\sigma}(t_k) \to \text{post}_{F_s}(x)$ is *complete*, if $\pi_p(\text{post}_{F_\sigma}(t_k)) = \text{post}_{F_s}(x)$. This complete labeling π_p, as a candidate labeling of π', ensures the consistency on t_k, i.e., $\pi'(\text{post}_{F_\sigma}(t_k)) = \text{post}_{F_s}(\pi'(t_k))$.

Example 16 (Example 15 Continued). In Example 15, the repairing candidates for the transition t_2 is given by $T_c := \{\text{design}, \text{revise}\}$. Similarly, the assignment of places $p_j \in \text{post}_{F_\sigma}(t_2)$ should be consistent with t_2. For the repairing $\pi'(t_2) = \text{design}$, a complete labeling π_p could be $\pi_p(p_2) = \text{b}$ and $\pi_p(p_4) = \text{d}$, for $p_2, p_4 \in \text{post}_{F_\sigma}(t_2)$. In contrast, for the repairing $\pi'(t_2) = \text{revise}$, there does not exist any complete labeling π_p which ensures the consistency on t_2, since $|\pi'(\text{post}_{F_\sigma}(t_2))| = 2 \neq |\text{post}_{F_s}(\pi'(t_2))| = 1$.

All the possible complete labeling π_p can be enumerated by considering the combination of $\text{post}_{F_s}(x)$ with repetition. Let b and d be the maximum sizes of the pre/post set of any node in the specification and execution, respectively. Each p_i in $\text{post}_{F_\sigma}(t_k)$ has b choices of $\pi'(p_i)$ for repairing. Considering all d places, the total number of possible labelings π_p is bounded by $O(b^d)$.

Example 17 (Example 1 Continued). Consider the first transition t_1 in Figure 1. Its name submit is already consistent with p_0 : start in $\text{pre}_{F_\sigma}(t_1)$. For t_2, we can find a repairing $\pi'(t_2) = \text{design}$, and find a complete labeling π_p, e.g., $\pi_p(p_2) = \text{b}$ and $\pi_p(p_4) = \text{d}$, for $p_2, p_4 \in \text{post}_{F_\sigma}(t_2)$ such that the violation count on t_2 is minimized. Similarly, for the next t_3, we can find a repairing, say $\pi'(t_3) = \text{insulation proof for example}$, and its corresponding $\pi'(p_3) = \text{c}$ in the post set, following the minimum violation count heuristic. Repairing carries on by one pass through the execution trace, and yields $\pi'(t_4) = \text{check inventory}$, $\pi'(p_5) = \text{e}$, $\pi'(t_5) = \text{evaluate}$, $\pi'(p_6) = \text{s}$.

5.2.3 Algorithm Analysis. Let b and d be the maximum sizes of the pre/post set of any node in the specification and execution, respectively. (Note that in most processes, b and d are often small and can be regarded as constants. As shown in Table 1, the maximum parallel flow of the datasets used in the experiment is 4. Additionally, techniques are often applied to keep the process as simple as possible in practice [35].) As each place $p_i \in \text{pre}_{F_\sigma}(t_k)$ can suggest $|\text{post}_{F_s}(\pi'(p_i))|$ (at most b) repairs for t_k, the total number of candidates in T_c is bounded by b. Considering all the $O(b^d)$ possible labelings, we have cost $O(b^{d+1})$ for repairing one transition. The **while** iteration repeats at most n times, $n = |T_\sigma|$ the number of transitions in the execution. Hence, the complexity of Algorithm 2 is $O(b^{d+1}n)$.

We select one of the alternatives for repairing a transition in the one pass algorithm, which is heuristically good but might not be optimal. The repairing results could be possibly bad, i.e., significantly differ from the original one compared to the optimal solution, as each transition may lead to a completely different flow in execution. Nevertheless, the one pass solution offers an alternative of trading time efficiency from repairing cost. As shown in the experiments, the one pass algorithm needs extremely low time cost while the observed approximation ratio is still considerable.

6 PLACE ORIENTED HEURISTICS

Although the one pass algorithm excels in time efficiency, it may generate too many false negatives regarding the detection of unsound structure sometimes due to its greedy selection of repair at each step. According to our experiments in Section 7, the accuracy of detecting unsound structure may drop to only 0.2. In this section, we first introduce a special case of simple path, which is prevalent in real-world scenarios. Then we put forward a place-oriented heuristic algorithm that not only gives an exact solution in simple path cases, but also outperforms the one pass algorithm in the previous section in terms of accuracy in general cases.

6.1 Simple Path Cases

Structured event logs as illustrated in Figure 1(a) widely exist in BPM and OA systems such as jBPM or IBM Lotus Notes. However, more general or simpler event logs often involve only timestamps. According to our survey among 46 process specifications in the bus manufacturer, 21 (45.6%) of them are simple path specifications.

A sequence of events is indeed a simple path. That is, each transition in the execution net has only one input place and one output place. There is no AND-split or AND-join in the corresponding specification.

Example 18. Figure 10(c) shows a specification collected from an order processing system. The first step submits a new order, then check inventory and pay by cash/credit card. Moreover, the specification requires a task validate when the purchase is done. After the validation, it could be followed with either a cancel task to cancel an invalid order, or a delivery task to ship all the ordered goods to customers. Obviously, the specification contains no AND-split/join, so that execution traces can be presented as causal nets which have only one path, as illustrated in Figure 10(a) and (b).

6.2 Place Mapping Algorithm

Considering that each transition in the execution has at most one previous/post place, the labeling of the transition can be uniquely determined by the labelings of its neighbor places. In contrast, there are many possibilities for the labelings of a place given the labelings of its neighbor transitions. Instead of repairing one transition at a time in the one pass algorithm in the previous section, the idea of the place mapping algorithm is to enumerate all the possible labelings for each place in the execution, and greedily merge valid labelings from the start place to the end place. In particular, since there is no AND structure in the simple path cases, no conflict will appear in the merge.

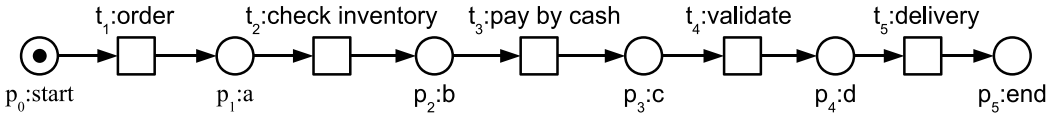
6.2.1 Valid Labeling with Minimum Cost. Unlike the transitions, places are not recorded in the execution trace. However, there still exist precedence relationships among the places in the corresponding recovered causal net. The prerequisite set of a place $pre\_set(p_i)$ is defined on the execution net N_σ as

$$pre\_set(p_i) = \{p_j \cup pre\_set(p_j) \mid p_j \in pre_{F_\sigma}(pre_{F_\sigma}(p_i))\},$$

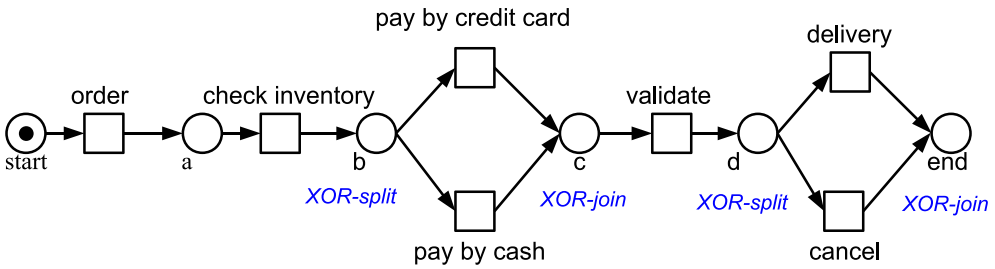
That is, the prerequisite set $pre\_set(p_i)$ contains all the places that on the path from the first place p_0 to the target place p_i, such as $pre\_set(p_3) = \{p_0, p_1, p_2\}$ in Figure 1(b). In particular, the prerequisite set of a place following an AND-join transition should include all the places between the corresponding AND-split and AND-join transitions. For example, in Figure 1(b), places $\{p_2, p_3, p_4, p_5\}$, which are between the AND-split transition t_2 and the AND-join transition t_5, are all in the prerequisite set of place p_6.

Event	Name	Timestamp
t_1	order	03-18 09:12:07
t_2	check inventory	03-18 09:27:14
t_3	pay by cash	03-18 09:30:18
t_4	validate	03-18 09:35:32
t_5	delivery	03-18 09:50:12

(a) An execution trace without structure information



(b) Representing execution as causal net



(c) Specification for order processing

Fig. 10. Example of simple path specification.

Consider any pair $(p_\sigma, p_s), p_\sigma \in P_\sigma, p_s \in P_s$, we define $\pi\{p_\sigma, p_s\}$ as a valid labeling with minimum repairing cost, where $\pi(p_\sigma) = p_s$ and all the places in $pre\_set(p_\sigma)$ are also assigned valid labelings. Since each place gets its labeling, the transition before that place also gets its labeling, and we record the corresponding minimum repairing cost as $cost\{p_\sigma, p_s\}$.

6.2.2 Generation of $\pi\{p_\sigma, p_s\}$. We present the place mapping algorithm in Algorithm 3. Breadth-first search is used to traverse all the places in the execution net N_σ (Lines 4–5 and 8–9), which ensures that each place is processed only after all the places in its prerequisite set have been assigned valid labelings. In each iteration, we consider all the places in specification as candidates for $\pi'(p_\sigma)$. The generation of $\pi\{p_\sigma, p_s\}$ is to find out a valid labeling for p_σ with minimum cost (Lines 11–23). To check if the attempt to assign p_σ to p_s is valid, we need to further consider the transition t_σ before the place p_σ, and the corresponding labeling t_s before p_s.

There are two conditions as illustrated in Figure 11. If it is not AND-join structure (as shown in Figure 11(a)), only one place exists before t_σ and t_s. The if-branch of Lines 14–15 will be conducted, and the labeling of current place p_σ can be directly merged with the labeling of previous place to calculate the minimum repairing labeling. Otherwise (like Figure 11(b)), else-branch of Lines 16–17 will be conducted to calculate a valid merged labeling with minimum cost. Suppose there are r places before the AND-join transition, i.e., $p_{k-1}, \ldots, p_{k-r}$, a valid merged labeling means merging all $\pi\{p_{k-1}, b_r\}, \ldots, \pi\{p_{k-r}, b_1\}$ without conflicts. In other words, all the same place/transition of the execution net in $\pi\{p_{k-1}, b_r\}, \ldots, \pi\{p_{k-r}, b_1\}$ should be assigned the same

ALGORITHM 3: PLACEMAPPING(N_σ, π, N_s)

Input: An execution (N_σ, π) and a specification N_s
Output: A repair π' such that $(N_\sigma, \pi') \vDash N_s$

```
1: let p_0 and p_n be the start and end place in the execution net
2: Initialize a queue Q
3: set π{p_0, start} := {p_0 : start}, cost{p_0, start} := 0
4: for each p_next ∈ post_{F_σ}(post_{F_σ}(p_0)) do
5:    insert p_next into the queue Q
6: while |Q| ≠ 0 do
7:    fetch the first element p_σ from the queue Q
8:    for each p_next ∈ post_{F_σ}(post_{F_σ}(p_σ)) do
9:       insert p_next into the queue Q
10:   t_σ := pre_{F_σ}(p_σ)
11:   for each p_s ∈ P_s do
12:      cost_min := +∞
13:      for each t_s ∈ pre_{F_s}(p_s) do
14:         if |pre_{F_σ}(t_σ)| = |pre_{F_s}(t_s)| = 1 then
15:            π* := π{pre_{F_σ}(t_σ), pre_{F_s}(t_s)}
16:         else
17:            let π* be the valid labeling with minimum repairing cost after merging
18:         if π* ≠ ∅ and Δ(π, π*) + Δ(π(t_σ), t_s) < cost_min then
19:            π_min := π* ∪ {t_σ : t_s}
20:            cost_min := Δ(π, π*) + Δ(π(t_σ), t_s)
21:      if cost_min ≠ +∞ then
22:         π{p_σ, p_s} := π_min ∪ {p_σ : p_s}
23:         cost{p_σ, p_s} := cost_min
24: if π{p_n, end} ≠ ∅ then
25:    return π{p_n, end}
26: else
27:    return unsound structure
```

labeling. The $\pi\{p_\sigma, p_s\}$ with minimum cost can be calculated by enumerating all the possible combination labelings $(\pi\{p_{k-i}, b_j\}, i, j = 1, \ldots, r)$ for these r places.

Finally, $\pi\{p_n, \text{end}\}$ will be returned if it is a valid repair. Otherwise, the unsound structure is identified.

Example 19 (Example 1 Continued). Consider the first place p_0 in Figure 1. We have $\pi\{p_0, \text{start}\} = \{p_0 : \text{start}\}$ and $cost\{p_0, \text{start}\} = 0$. Due to the breadth-first search strategy, p_1 is the next considered p_σ. After enumerating all possible labelings, we find only one valid pair (p_1, a), and get $\pi\{p_1, \text{a}\} = \pi\{p_0, \text{start}\} \cup \{t_1 : \text{submit}\} \cup \{p_1 : \text{a}\}$, $cost\{p_1, \text{a}\} = cost\{p_0, \text{start}\} + 0 = 0$. Similarly, we have $cost\{p_2, \text{b}\} = 6$, $cost\{p_2, \text{d}\} = 6$, $cost\{p_2, \text{f}\} = 3$, $cost\{p_4, \text{b}\} = 6$, $cost\{p_4, \text{d}\} = 6$, $cost\{p_4, \text{f}\} = 3$, $cost\{p_3, \text{c}\} = 17$, $cost\{p_3, \text{e}\} = 20$, $cost\{p_3, \text{g}\} = 12$, $cost\{p_5, \text{c}\} = 22$, $cost\{p_5, \text{e}\} = 21$, $cost\{p_5, \text{g}\} = 17$.

When considering p_6 as p_σ, since $|pre_{F_\sigma}(pre_{F_\sigma}(p_6))| = 2$, only one pair (p_6, s) is valid. Merging $\pi\{p_3, \text{c}\}$ and $\pi\{p_5, \text{e}\}$ will get a non-conflict result with minimum repairing cost $cost\{p_6, \text{s}\} = 32$, since they have the same labeling of the AND-split transition, i.e., $\{t_2 : \text{design}\}$, and the repairing cost is lower than merging $\pi\{p_3, \text{e}\}$ and $\pi\{p_5, \text{c}\}$. Finally, we get $cost\{p_7, \text{end}\} = 32$ and return $\pi\{p_7, \text{end}\}$.

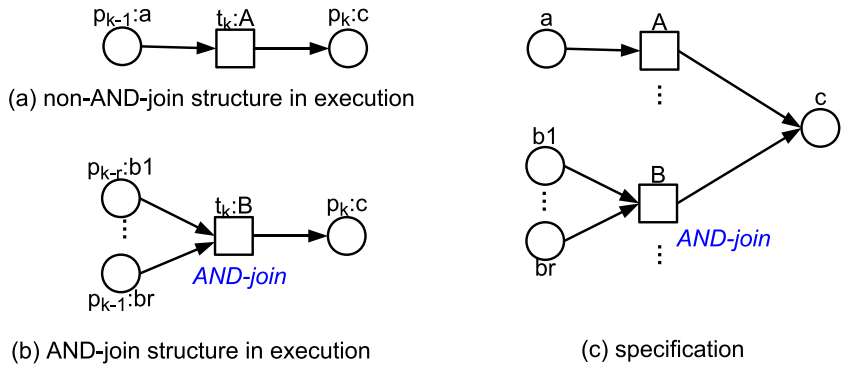


Fig. 11. Generation of $\pi\{p_\sigma, p_s\}$.

6.2.3 Algorithm Analysis. When considering the labeling of each place p_σ, we need to check if there is a valid labeling of all the previous places and then calculate the merged result. Let b and d be the maximum sizes of the pre/post set of any node in the specification and execution, respectively. (Note that in most processes, b and d are often small and can be regarded as constants. As shown in Table 1, the maximum parallel flow of the datasets used in the experiment is 4. Additionally, techniques are often applied to keep the process as simple as possible in practice [35].) There are at most $O(b^d)$ valid labelings for the previous places of p_σ. Each labeling can suggest at most b candidates for the transition t_σ before p_σ. By comparing all these $O(b^{d+1})$ cases, we get the $\pi\{p_\sigma, p_s\}$ with the minimum cost. The total number of different labelings of all the places is at most $O(n^2)$, where n is the maximum number of places in the specification and execution. Hence, the complexity of Algorithm 3 is $O(b^{d+1}n^2)$.

Proposition 7. *The place-oriented heuristic algorithm gives an exact solution when applied to simple path cases, and runs in $O(bn^2)$ time.*

Proof. Since there are no AND splits/joins, each transition is followed by exactly one place in the specification. We only need to consider the if-branch (Lines 14–15) in Algorithm 3, without attempting to merge optimal repairs from different AND-split branches, and conflicts will never occur. Therefore, an exact solution with minimum cost is produced. The inconsistent labeling repair problem with a simple path case can be solved by Algorithm 3 in $O(bn^2)$ time. A place of XOR-split always leads to one and at most one post transition in the execution. Note that b denotes the maximum size of the pre/post set of any node in the specification, including the maximum size of the pre/post set of the XOR-split nodes. Therefore, for each labeling of p_σ, there are at most b candidates for the transition t_σ before p_σ. Besides, for each candidate of transition t_σ, there is only one possible labeling for the place before the transition t_σ, i.e., the if-branch (Lines 14–15) in Algorithm 3 applies. For any pair $(p_\sigma, p_s), p_\sigma \in P_\sigma, p_s \in P_s$, we can get the valid labeling $\pi\{p_\sigma, p_s\}$, as defined in Section 6.2.1, with the minimum cost by considering all these $O(b)$ cases. Since the total number of all the possible pairs $(p_\sigma, p_s), p_\sigma \in P_\sigma, p_s \in P_s$ is at most $O(n^2)$, where n is the maximum number of places in the specification and execution, the time complexity for the simple path case is $O(bn^2)$. □

7 EXPERIMENTS

In this section, we first introduce the experimental settings. Then, the performance of the proposed repairing methods is evaluated on both effectiveness and efficiency. All programs are implemented in Java, and experiments run on a computer with 2.67 GHz CPU and 16 GB memory.

Table 1. Statistics on Datasets

Dataset	# traces	max trace size	max parallel flow
Workflow	4,540	75	3
Log	1,000,000	48	4
Bank	10,000	59	3
Synthetic	1,000,000	350	3

7.1 Experimental Settings

7.1.1 Data Set. In order to study the accuracy of repairing inconsistent labeling by the proposed methods, we employ four datasets. Table 1 reports the statistics of datasets.

The real Workflow data set is collected from a bus manufacturer. The event data are extracted from processes related to the bus design and customization affairs. The specification considered in the experiments consists of 22 transitions and 24 places with the maximum size of pre/post set 3 (the maximum parallel flows). There are 4,722 traces collected during the execution of the process. Most of the traces are small in size, in the range of 6 to 20. The maximum size observed in all the traces is no greater than 75. According to our observation, the maximum size of pre/post set in the execution is 3 as well. In particular, 3.85% of execution traces are irrelevant to the specification, i.e., with all the event names not from the specification. The experiments are conducted on the rest 4,540 traces.

The Log dataset contains a million log traces from a distributed monitoring system, which involves several micro-services such as user login and database query management. The process specification is built referring to the system design document. The events are parsed from the original system logs by several predefined templates. Since the service invocation chains are also traced, structural information among events can be obtained through the invoking relationships. Each tuple in the trace can be parsed as ⟨ Event, Name, Operator, Successor ⟩ format, which is analogous to Figure 2(a). It is noted that all the traces in Log dataset are simple paths with up to 48 events. The maximum size of pre/post set (the maximum parallel flows) is 4, which comes from the XOR structure in the specification.

We also consider a public Bank dataset[4] which records the bank transaction processes. Since the structure information is not provided in this dataset, we first compare the execution trace to the provided specification, and try to recover the execution in the form of the casual net by re-executing the trace according the specification. After obtaining the casual net, we can easily transform the trace to the format shown in Figure 1(a). The Bank dataset consists of 10 thousand traces with up to 59 events in one execution, and the maximum size of pre/post set is 3.

In addition, to evaluate the scalability of the proposed methods, we generate the synthetic event data following the method in [42] by using the commonly used workflow patterns, such as parallel, sequential, and so on. There are 1 million generated traces in total, with the sizes ranging from 50 to 350, and the maximum parallel flows is 3.

We manually confirm the ground truth for the inconsistent labelings which are naturally embedded in the real Workflow dataset collected from our industrial partner, a bus manufacturer. There are 4,540 traces collected during the execution of the process which are relevant to the specification. We label the correct event name of the inconsistent trace or mark it as unsound structure with the help of the specialist knowledge.

For the other larger datasets, Bank and Log, we do not have the knowledge of ground truth and they are too large to label manually. Indeed, we assume the data are originally clean, since

[4]https://data.4tu.nl/articles/dataset/Large_Bank_Transaction_Process/12714395.

Table 2. Statistics of the Structures in Specification

Dataset	XOR-split	XOR-join	AND-split	AND-join
Workflow	4.44%	4.44%	4.44%	4.44%
Log	11.54%	11.54%	0	0
Bank	6.41%	6.41%	5.98%	5.98%
Synthetic	1.46%	1.46%	1.46%	1.46%

Table 3. Execution Trace Quality Statistics of Real Workflow Data

Traces	#	%	% inconsistencies
consistent	665	14.08	
inconsistent (event name repairable)	3,665	77.62	94.58
inconsistent (unsound structure)	210	4.45	5.42
irrelevant	182	3.85	

they all conform to the specification when downloaded. Instead, we manually inject faults by randomly changing the event names in the traces, e.g., for fault possibility 0.1, each event name is randomly altered with a possibility of 0.1. In order to simulate the scenarios of the noise observed in real event traces, we alter the event names by (1) randomly deleting/adding some words/letters or (2) replacing with another event name. The repairing methods are then applied to modify the execution traces to eliminate violations. We study the accuracy of the repairing results via comparison with the truth of faulty data previously replaced. For each trace, we conduct the random insertion of faults 1,000 times and compute the average accuracy. Meanwhile, the repairing time performance is also reported.

7.1.2 Criteria. Let truth be the set of original correct events $(t, \pi_o(t))$ that are randomly replaced in an execution trace. Let found be the set of $(t, \pi'(t))$ that are repaired in π', i.e., the repairing results. To evaluate the accuracy, we use the f-measure of precision and recall [46], given by $precision = \frac{|\text{truth} \cap \text{found}|}{|\text{found}|}$, $recall = \frac{|\text{truth} \cap \text{found}|}{|\text{truth}|}$, and $f\text{-}measure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$. It is natural that a higher f-measure is preferred.

To study the difference between optimal and heuristic solutions, we report the relative performance (Δ/Δ^*), where Δ and Δ^* denote the repairing cost of the heuristic and exact approaches. The closer the relative ratio is to 1, the better the approximation performance.

Recall that the heuristic methods may generate **false negatives** (**FN**) regarding the detection of unsound structure, while the exact approach computes the **true positives** (**TP**) and **true negatives** (**TN**). It is noted that the heuristic methods will not return **false positive** (**FP**), i.e., return the structure is sound but the truth is unsound, since a valid repairing is already found to make the execution conforming to the specification. To compare the effectiveness of detecting unsound structure, we also report the accuracy of the heuristic methods $\frac{\#\text{TP}+\#\text{TN}}{\#\text{TP}+\#\text{TN}+\#\text{FN}}$.

7.1.3 Statistics. Table 2 illustrates some statistics of the structures in the specification of the datasets used in the experiments. As shown, XOR-split and XOR-join appear in all the real-world datasets. The execution traces in the Log dataset are simple paths, i.e., there is no AND-split or AND-join in the specification.

Furthermore, to illustrate how the constrained model matches the process specification, Table 3 studies the statistics on the real Workflow dataset with errors naturally embedded and labeled. As shown, only about 14.08% traces match exactly the process specification. For those execution traces with inconsistencies, there are 77.62% traces that can be repaired under the constraints of

specification, while the remaining 4.45% traces are identified with unsound structure. In particular, 3.85% of execution traces are even found irrelevant to the specification.

7.2 Comparison with Existing Methods

The approaches in comparison are (1) our proposed exact algorithm (*Exact*), the transition-oriented heuristic algorithm (*OnePass*), and place-oriented heuristic algorithm (*PlaceMapping*). (2) The partially ordered based trace alignment method [34] (*p-alignment*). (3) The state-of-the-art graph repair techniques *GMend* [28], and Graph Relabel [40] (*GRelabel*). The graph quality rules of *GMend* are derived according to the specification of the dataset. Specifically, the AND structure is retained in the graph pattern of the quality rules, while only one of the branches in the XOR structure could be preserved. That is, the obtained graph quality rules cannot express the semantic of XOR. The reason is that the graph pattern used to identify the repairing entity requires an exact matching of all the nodes in the graph. As a consequence, only the preserved XOR branch could be considered in the repairing while others are ignored by *GMend*.

We first report the results over the real Workflow dataset. The comparison is performed on various possibilities of inserted faults in Figure 12 and various trace sizes in Figure 13. The accuracies of both the *Exact* and heuristic algorithms are considerable, with f-measure no less than 0.8, as illustrated in Figure 12(c). Remarkably, the *PlaceMapping* approach is comparable to the *Exact*, having f-measures as high as 0.9. While the accuracy performance of the *OnePass* method is not as stable as the *Exact* and *PlaceMapping*. The rationale is that *OnePass* determines a heuristically good assignment as the repair of a transition without trying other alternatives like the exact algorithm. Consequently, by choosing an incorrect assignment in a step, the repairing may vary in the following steps.

The accuracy of *p-alignment* drops quickly on large fault possibilities. The reason is that the number of events may be different from the original execution trace after repairing, since the structure of execution could be modified when the unsynchronized model/log move occurs. The graph based repairing techniques *GMend* and *GRelabel* have lower accuracies compared to our proposal. The reason lies in that *GRelabel* is originally designed for repairing simple graphs by neighborhood constraints, which do not consider AND and XOR semantics. The graph quality rules such as **graph functional dependencies** (**GFDs**) specified in *GMend* cannot support the AND/XOR semantics on events very well either.

As shown in Figure 13(a), our *Exact* and heuristic algorithms keep high accuracies when the size of trace grows up, with f-measure no less than 0.8. Figures 12(d) and 13(b) report the efficiency evaluation. It is not surprising that the repairing time cost of *Exact* increases with the increase of the fault possibility in Figure 12(d). According to our analysis, the *Exact* algorithm has exponential complexity in the number of events (transitions). Therefore, its time costs increase heavily with the increase of trace sizes in Figure 13(b). Nevertheless, *OnePass* algorithm shows significantly lower time costs (similar to *p-alignment*, but with higher accuracy than *p-alignment*, especially in large fault sizes and trace sizes). The *PlaceMapping* algorithm can achieve comparable accuracy to the *Exact* while keeping relatively lower time costs.

The experiments over the Bank dataset show similar results. The comparison result over various possibilities of inserted faults is presented in Figure 14, while the performance over various trace sizes is reported in Figure 15. As illustrated in Figures 14(a) and 15(a), our exact and heuristic methods still achieve very high accuracies with f-measures no less than 0.8. Figures 14(b) and 15(b) report the efficiency evaluation. The complex specification, with more nested AND/XOR structures of Bank dataset, makes the exact method generate much more branches in the repairing, and significantly increases the total time cost. Therefore, the more efficient heuristic algorithm is needed.

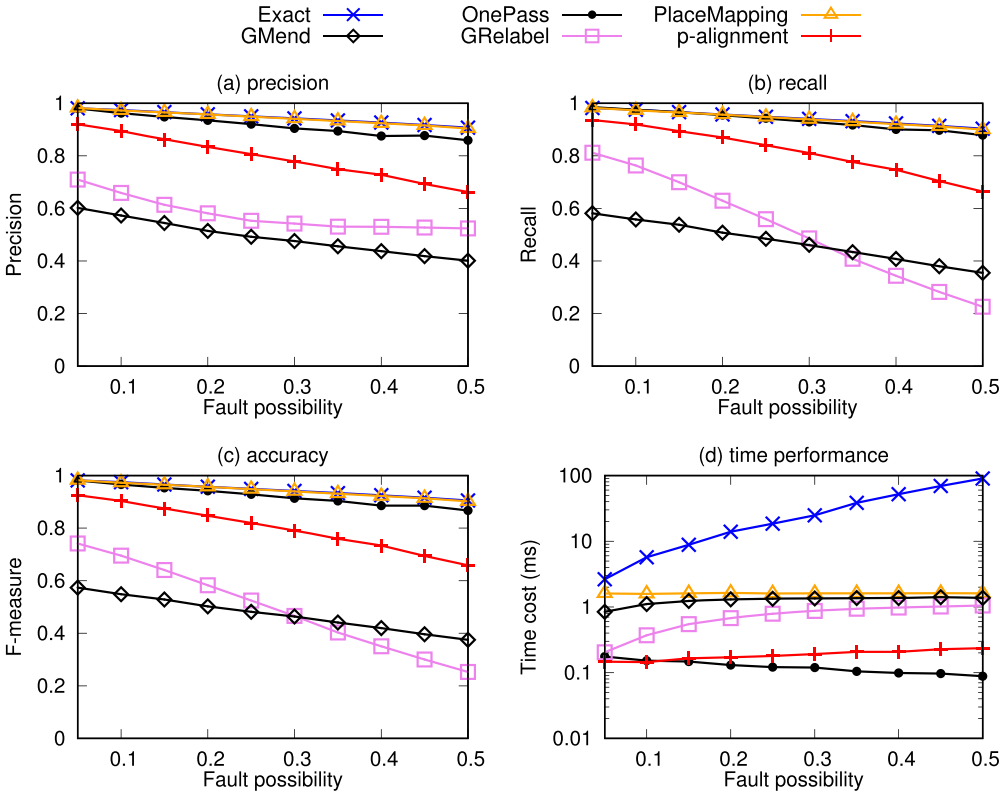


Fig. 12. Performance of repairing various faults over the Workflow dataset.

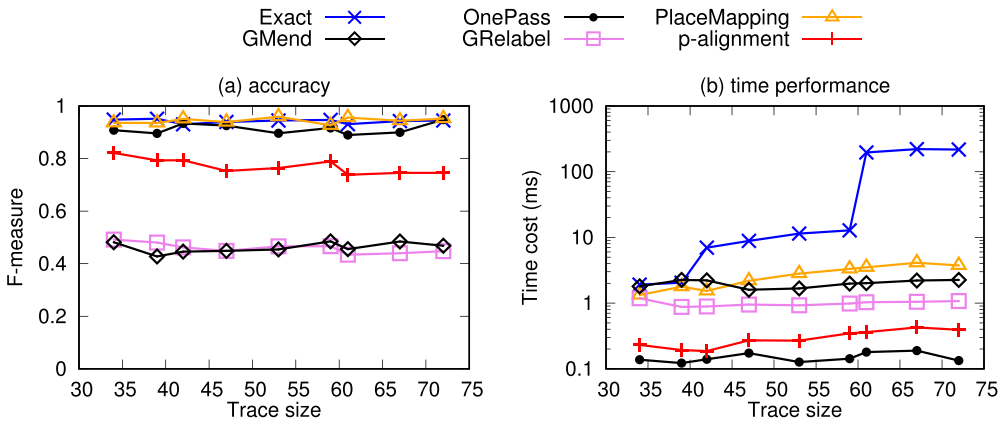


Fig. 13. Performance on various trace sizes over the real dataset.

It is noted that *p-alignment* also shows much higher time cost in larger fault possibility. The reason is that the complex specification also increases the search space of the A* algorithm in *p-alignment*.

Figures 16 and 17 evaluate the compared methods over the Log dataset on various possibilities of inserted faults and various trace sizes, respectively. The f-measure of the proposed exact and

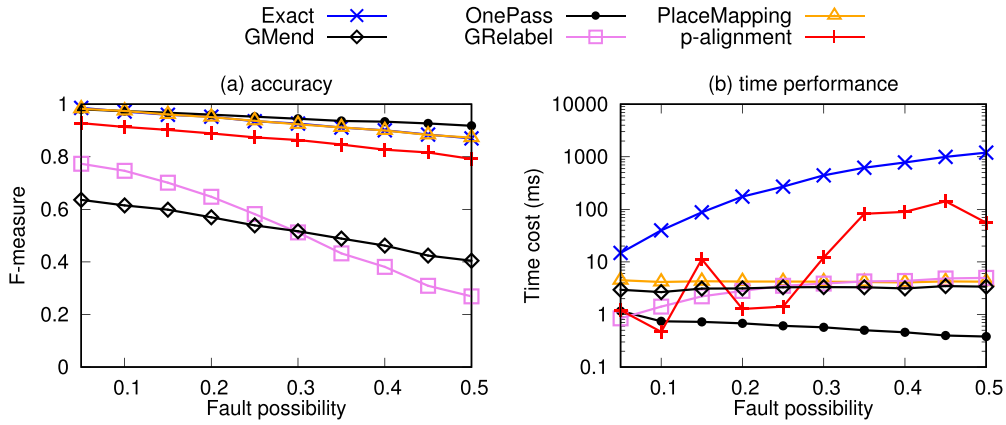


Fig. 14. Performance of repairing various faults over the Bank Dataset.

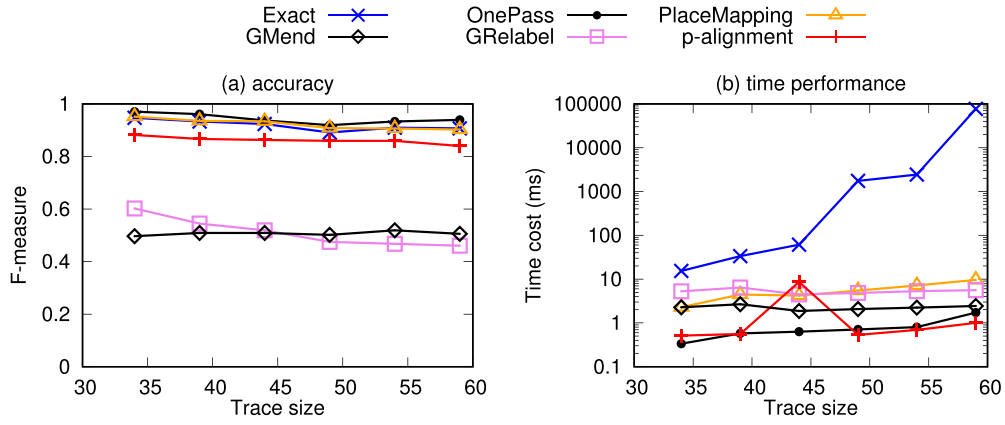


Fig. 15. Performance on various trace sizes over the Bank dataset.

heuristic methods remain as high as 0.9 with various fault possibilities. The accuracy of *OnePass* is not as high as *Exact* and *PlaceMapping* in larger trace size. The rationale is that the dataset contains more XOR in specification which makes it harder to determine the correct assignment. Lower accuracy is also observed in larger trace size for *p-alignment*. Since the standard cost function used in *p-alignment* assigns the same cost to different unsynchronized model/log moves without distinguishing them, it is more likely to lead to wrong branches with more XOR branches. While our proposal follows the minimum change idea in database repairing [9], as discussed in Section 2.4, and find a repair that minimally differs from the original data. Figures 16(b) and 17(b) evaluate the time efficiency. Although all the traces are simple path in Log dataset, the time cost of *Exact* still increases quickly with the increase of fault possibilities and trace sizes.

7.3 A Closer Look at Proposed Techniques

We compare our proposed pruning techniques in Figures 18–20, including the Exact algorithm with the Simple bounding function (*ES*), the Exact algorithm with the Advanced bounding function (*EA*), and the Pruning of Invalid branches for the exact algorithm (*PI*).

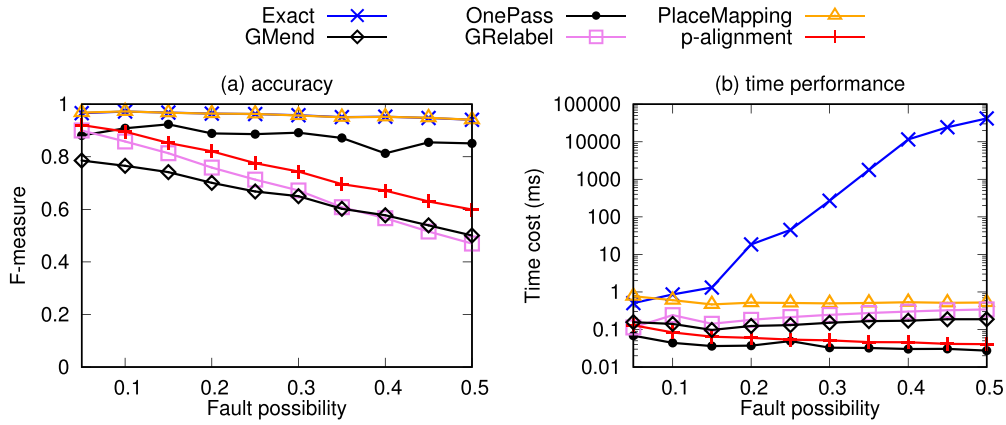


Fig. 16. Performance of repairing various faults over the Log dataset.

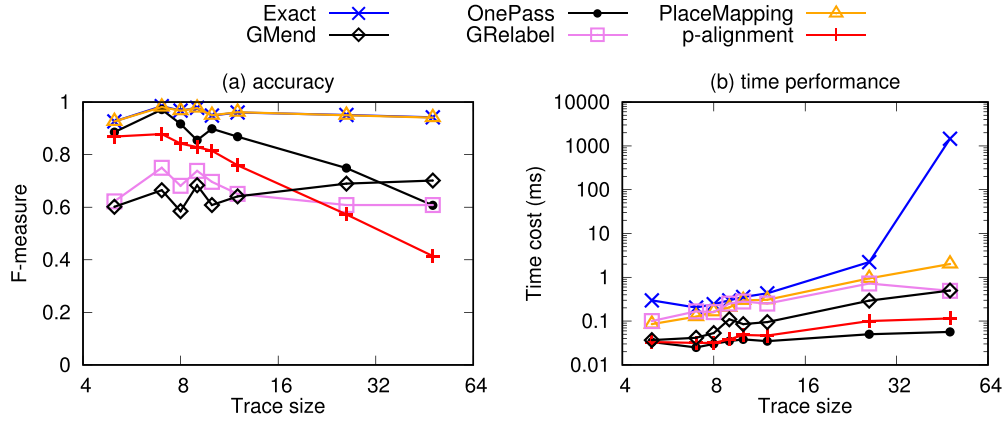


Fig. 17. Performance on various trace sizes over the Log dataset.

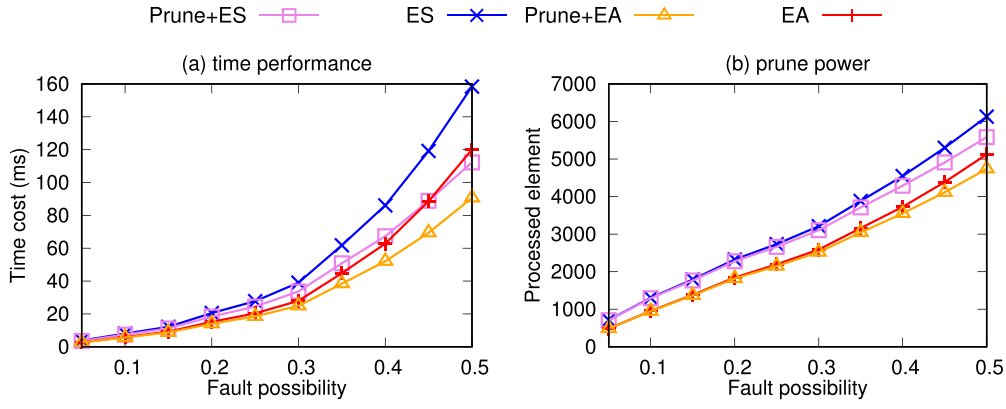


Fig. 18. Comparison of proposed methods on repairing various faults.

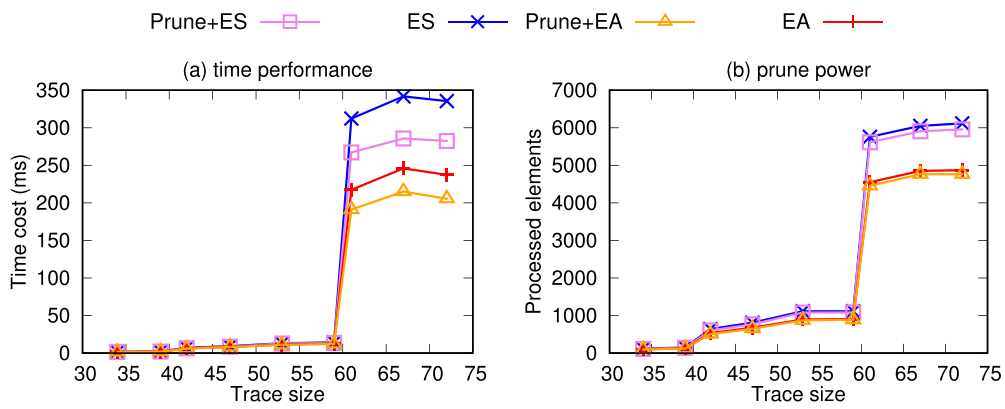


Fig. 19. Comparison of proposed methods on various trace sizes.

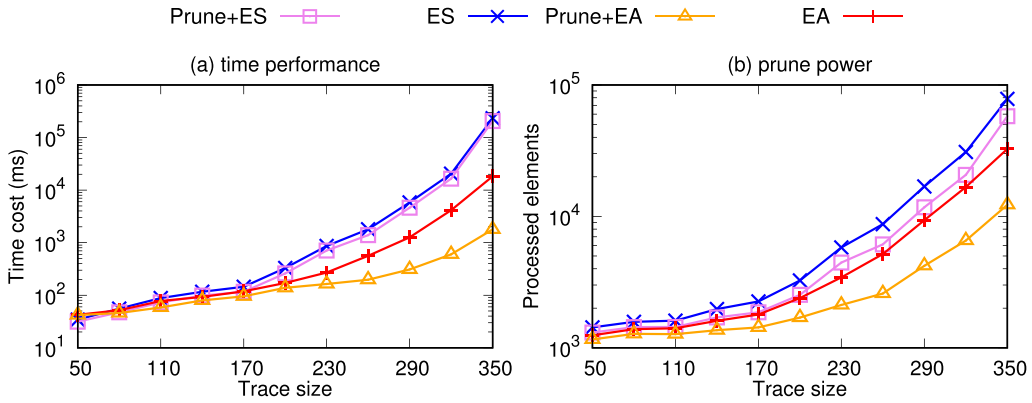


Fig. 20. Scalability on Synthetic data.

In Figure 18(a), we demonstrate that the advanced bounding function (*EA*) can reduce the repairing time significantly compared with the simple one (*ES*). In order to illustrate the pruning power of different bounding functions, in Figure 18(b), we show that *EA* needs fewer elements of repairing states to be processed (i.e., the total number of nodes in Figure 6). The effectiveness of pruning on invalid branches is limited, since the traces with sound structure in this experiment have lower chance to involve invalid branches.

Similarly, as illustrated in Figure 19(a), *EA* method with the advanced bounding function can reduce time cost considerably, compared with *ES*. Indeed, the time cost in Figure 19(a) is proportional to the size of processed elements of branching states in Figure 19(b). The processed elements as well as the pruning power may not increase strictly with the trace size, owing to the structural difference in the process. A sudden rise is observed when the trace size reaches around 60, because the traces with size larger than 60 have an extra sub-process of AND structure. Since the number of parallel flows after the AND-split transition is more than one, there are more combinations of possible labelings being considered. Referring to the property of bounding functions, the pruning power of the advanced pruning bound is at least no worse than that of the simple one, which is also observed in Figure 19(b). It is notable that the pruning method of invalid branches does not show significant improvement. The reason is that our currently employed real

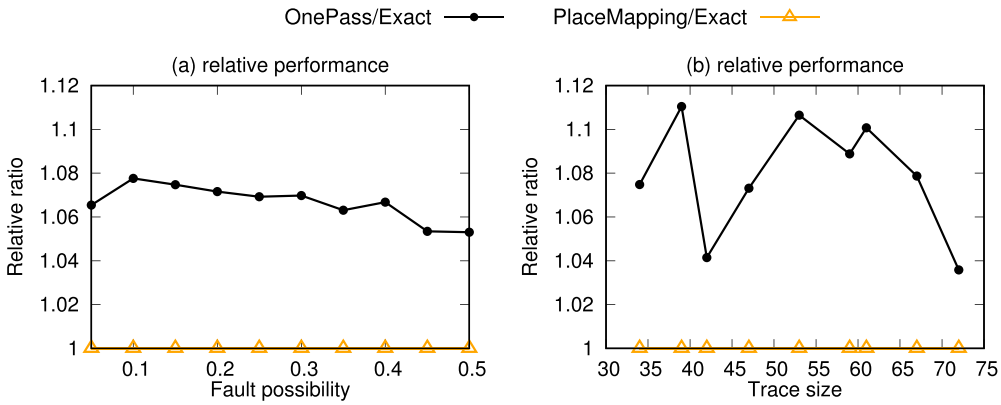


Fig. 21. Comparison of exact and heuristic repair methods over the Workflow dataset.

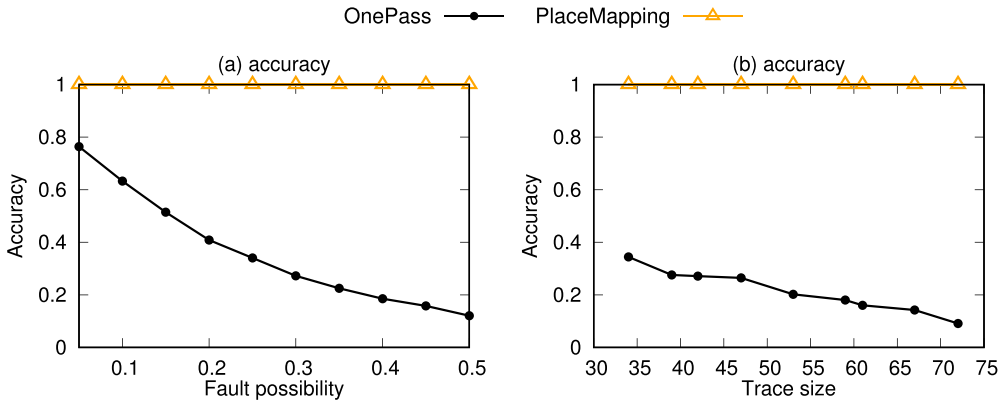


Fig. 22. Comparison of the heuristic methods with regard to the unsound structure detection over the Workflow dataset.

data set has a small portion of unsound structure traces, i.e., only 4.45% as shown in Table 3. The opportunity of pruning on invalid branches is thus limited during repairing.

In order to evaluate the scalability of the proposed methods, Figure 20 reports the experiment on larger synthetic data. Note that we can find a valid repair for most execution traces in the previous real data. In order to study the performance of unsound structure cases, the synthetic data contains 20% traces that do not exist any valid labeling. As illustrated in the results, the advanced bounding function (*EA*) can always show better pruning power and needs much lower time cost than *ES*. Remarkably, the pruning method performs well together with both *ES* and *EA*, since it can prune the invalid branches especially in those traces that contain unsound structures.

We evaluate our proposed heuristic techniques in Figures 21–24. Figure 21 shows the relative performance Δ/Δ^* of the repairing cost Δ by the heuristic algorithms (*OnePass* and *PlaceMapping*) and the optimal solution Δ^* by the *Exact* algorithm. As illustrated, both the repairing costs are very close to the optimal one , with relative difference no greater than 1.12. Moreover, since the *OnePass* algorithm may generate false negatives regarding the detection of unsound structure, we check the accuracy (defined in Section 7.1.2) of the detection results returned by the heuristic methods.

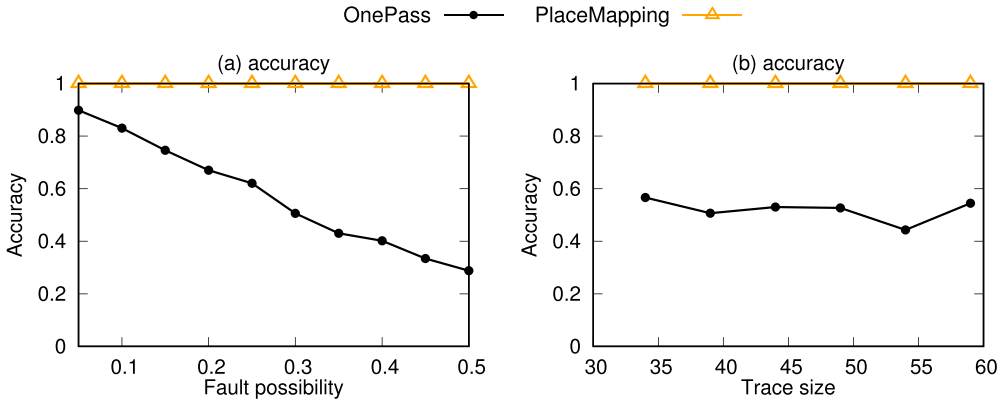


Fig. 23. Comparison of the heuristic methods with regard to the unsound structure detection over the Bank dataset.

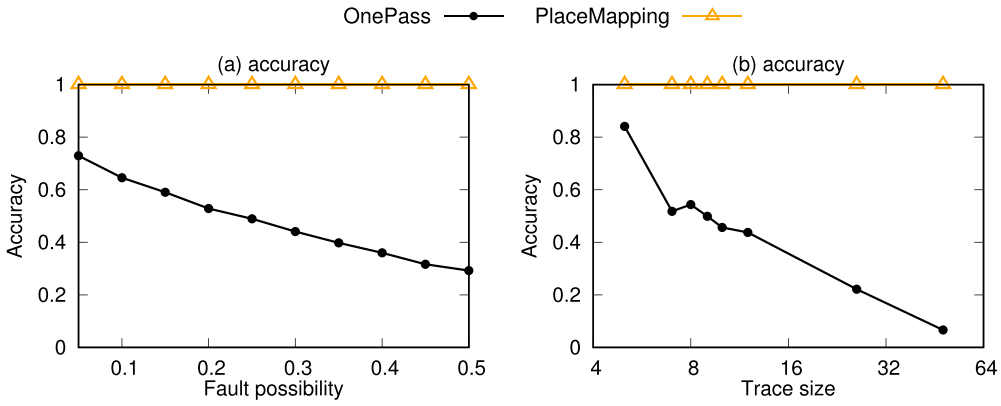


Fig. 24. Comparison of the heuristic methods with regard to the unsound structure detection over the Log dataset.

Figures 22(a) reports the evaluation on the real Workflow dataset. As illustrated, the accuracy of *OnePass* drops quickly with the increase of fault possibilities, since *OnePass* is more likely to lead to a completely different flow with more faults in the trace. The evaluations over Bank dataset (Figure 23(a)) and Log dataset (Figure 24(a)) show similar results. Owing to the structural difference in the process, the accuracy of *OnePass* has no great change with various trace sizes in Figure 23(b). While in Figures 22(b) and 24(b), the accuracy of *OnePass* also indicates a decreasing trend with the increase of trace size. In contrast, the *PlaceMapping* algorithm makes better approximation by keeping high accuracy with the increasing of both the fault possibility and trace size. Since all the traces are simple paths in the Log dataset, the repairing results of *PlaceMapping* are the same as the *Exact*. The results verify the conclusion in Proposition 7 that the *PlaceMapping* algorithm gives an exact solution when applied to the special case of simple path.

Figures 22 and 23 show that the place-oriented heuristic algorithm can reach better approximation ratio, compared with the transition-oriented heuristic algorithm. While this is not guaranteed theoretically in general, it is always the case for the special case of simple path. Referring to Proposition 7, the place-oriented heuristic algorithm gives an exact solution when applied to simple path cases, while the transition-oriented heuristic algorithm does not have a theoretical bound due to

its greedy strategy. It is worth noting that simple paths are prevalent and applicable not only in the data with all execution traces in this special case, such as dataset Log, but also in solving the sub-problems of execution traces in general. For instance, after processing t_5 in Figure 1(b), the remaining execution trace is indeed a simple path, and thus the place-oriented algorithm returns the optimal solution in this sub-problem. In this sense, owing to the performance guarantee of returning the optimal solutions in the (sub)problems, the place-oriented algorithm shows better results than the transition-oriented one.

7.4 Experiments on Recovery of Missing Events

We compare a baseline using Petri net, namely *EventRecovery*, which studies the efficient techniques for recovering missing events [49]. Our proposal can also be applied in this similar job. For example, to recover the missing event name—of t_4 in Figure 1(a), we consider a fixed *dis* 1 between—and any other event name, and a fixed *freq* 1, when evaluating the repair cost of the special (missing) event name—in Equation (2). The labeling repair algorithm will find a repair check inventory of—for t_4 that can satisfy the specification in Figure 1(c) and is with the minimum cost in total.

To prepare the datasets for evaluation, instead of changing the event names in the traces as errors, we randomly delete the event names to simulate the missing information in this experiment. A missing rate, for example, 0.1, denotes that 10 percent events are missing in the dataset. Different from *EventRecovery* that simply views event logs as sequences, our proposed exact and heuristic methods leverage the structure information in execution. Following the same setting in the evaluation of *EventRecovery* [49], we use the f-measure to evaluate the accuracy of recovery. Let *removed* be the set of all the removed events and *recovered* be the set of all the recovered events. We have $precision = \frac{|\text{removed} \cap \text{recovered}|}{|\text{recovered}|}$, $recall = \frac{|\text{removed} \cap \text{recovered}|}{|\text{removed}|}$, $f\text{-}measure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$. A larger f-measure indicates a higher recovery accuracy.

As illustrated in Figure 25, our *Exact* and heuristic algorithms keep higher accuracies when the missing rate increases. Since the structure information is not exploited, the minimum recovery result in *EventRecovery* may not be as accurate as the proposed methods in this work, especially when the missing rate is high. Similar to the results in Section 7.2, the time cost of the *Exact* algorithm grows faster when the missing rate increases. *EventRecovery* has the lowest time costs due to its pre-computing recovery path between two events. *OnePass* algorithm shows similar time costs to *EventRecovery*, but with higher accuracy than *EventRecovery*, especially in large missing rates. The *PlaceMapping* algorithm can keep relatively lower time costs while achieving comparable accuracy to the *Exact* algorithm.

7.5 Experiments on Application Performance Management

While the exact methods are generally efficient as shown in Figures 18(a)–20(a), they may still not be fast enough for the streaming traces of events, e.g., in the APM for monitoring the performance of software applications. This experiment demonstrates how the repairing may fall behind the streaming data generation, in a real APM scenario from our industrial partner Cloudwise.

Figure 26 shows the start time and end time of execution traces being generated and then cleaned by the proposed methods over a data stream. The red line in Figure 26 illustrates the start time and end time of each trace, where a new trace comes in every second. The repair starts immediately when the entire trace arrives and the previous trace finishes repairing. It means every trace is expected to be cleaned in one second, before the next trace comes. As illustrated in Figure 26, the two heuristic methods *OnePass* and *PlaceMapping* could satisfy the online repairing, i.e., most of the points are under the red line. In contrast, the *Exact* method suffers from relatively high repairing

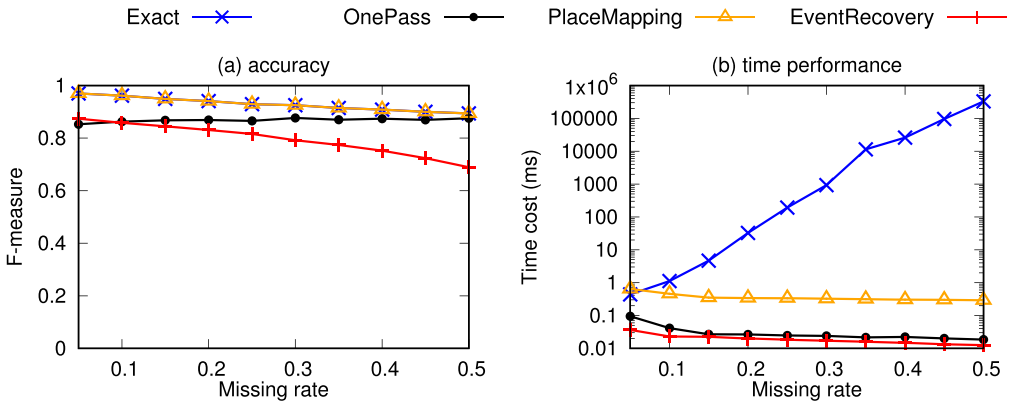


Fig. 25. Event recovery performance on various missing rates over the Log dataset.

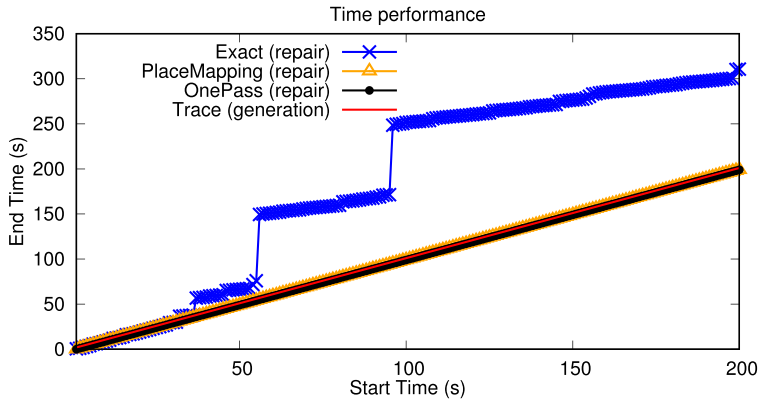


Fig. 26. Trace generation and repairing time in a data stream.

delay, since all the subsequent traces are delayed by certain traces with high repairing time. In addition, as illustrated in Figures 16 and 17, the *PlaceMapping* heuristic shows higher accuracy than *OnePass*, close to the *Exact*. In this sense, an even more efficient and effective algorithm is always needed.

8 RELATED WORK

The cooperation of business processes and data management has been emphasized for various workflow networks involving both data and flow, e.g., in Web applications, e-governance, and electronic patient records [17]. In particular, workflow techniques are useful for data management tasks such as data lineage and data provenance [7, 42]. Instead of repairing inconsistencies, the existing study assumes event data to be clean and is dedicated to improving the execution performance, i.e., optimize the execution [30]. As described, repairing event data is highly demanded and non-trivial.

8.1 Process Data Management

Studies on process data conducted by the data management community mainly focus on processing queries over workflow executions [5, 14, 15, 18]. A typical query inputs a process specification and

a pattern of execution, and tries to identify all the executions that have the structure specified by the pattern. Additional conditions may be added in the query, such as type information [14] or probability [16]. Moreover, as an important application, provenance queries on workflows are well investigated [2–4, 31]. A typical provenance query calculates the transitive closure of dependencies of an event in the process data. In particular, Bao et al. [2] studied the difference provenance, i.e., computing the structural difference of two executions. Note that the repairing studied in this work employs the modification of names in events (transitions) without changing the structure. Our approaches either identify the executions with unsound structures or repair them for conformance. As the prerequisite of execution is not changed in repairing, the repairing cost is directly computed by modification count.

The conformance checking [13, 34, 37] studied in the process mining field also assesses the deviations of event data with respect to the expected behavior of the process. The commonly used alignment algorithm [1, 13] only considers the sequential information in execution without utilizing the structural information. Although the partial orders between events are studied in p-alignment [34], the structure of execution could be modified when the unsynchronized model/log move occurs. As a result, the alignment-based techniques cannot detect the unsound structure. In addition, the number of events may be different from the original execution trace after alignment, which changes the structure of the execution and is not a valid repair in this work. Consequently, the performance of our proposal demonstrates higher repair accuracy in the experiments.

8.2 Database Repairing

Integrity constraints are often employed to eliminate inconsistencies in databases [22, 41]. Most previous works consider equality constraints such as inclusion dependencies, functional dependencies or conditional functional dependencies [8]. The repairing aims at modifying a minimum set of tuple values in order to make the revised data satisfy the given constraints [9, 50]. Although we adopt the same modification repairing, the constraints are very different between data dependencies and process specifications. In particular, the equality based data dependencies specifies groups of tuples with equal values, which do not exist among transitions in event data. Approaches are also proposed that do not follow the minimality, such as fix with master data and edit rules [26], partial currency orders [24], or accuracy rules [10], and so on. To cooperate with the art techniques, extra information is often needed, e.g., master data or additional rules.

A variety of dependencies have recently been studied for graph data [25, 28, 29, 40]. GFDs [29] provide a primitive form of integrity constraints to specify a fundamental part of the semantics of the schemaless graph-structured data. **Numeric graph dependencies** (**NGDs**) [27] extend GFDs with linear arithmetic expressions and built-in comparison predicates. **Graph association rules** (**GARs**) [25] make an effort to incorporate ML classifiers into logic rules for association deduction to catch missing links and attributes. **Graph quality rules** (**GQRs**) [28] are introduced to simultaneously repair data, identify objects and deduce entities that do not match. All these dependencies are specified with a graph pattern, which is different from the process specification, and cannot support the AND/XOR semantics on transitions in event data very well. In addition, the repairing according to these dependencies may involve structure modification [40], and thus cannot detect the unsound structure. Consequently, as illustrated in the experiments in Section 7.2, the performance of adapting existing graph repairing techniques is not as good as our proposal.

9 CONCLUSIONS

In this article, we study the problems of efficiently detecting unsound structure and repairing inconsistent event names. Firstly, to repair event data with inconsistent labeling but sound structure, we follow the widely used minimum change principle to preserve the original information as much

as possible. We also reveal the NP-completeness of the studied repairing problem. Then, we devise a novel, practically efficient exact algorithm to conduct detection and repairing dirty event data simultaneously so that it either (1) reports unsound structure or (2) gives the minimum repair of inconsistent event names. In addition, we present a PTIME transition-oriented heuristic algorithm to approximately deliver the results. Moreover, considering that the transition-oriented heuristic algorithm may generate too many false negatives regarding the detection of unsound structure, we further put forward a place-oriented heuristic algorithm. It not only gives an exact solution in the special case of simple path, but also makes better approximation in general cases.

Experiments on both real and synthetic data demonstrate the effectiveness and efficiency of the proposed methods. In particular, the repair accuracy of our proposal is significantly higher than the existing partially ordered based repair [34] and the direct application of graph repair [28, 40]. The transition-oriented heuristic algorithm is much more efficient than the exact one, while the place-oriented heuristic algorithm exhibits higher approximation ratios. According to the statistics on real datasets, among the execution traces with detectable inconsistencies (82.07%), most are structurally sound with repairable event names (77.62%). After detecting unsound structures by this proposal, an interesting future study is to automatically suggest possible structural explanations during the manual consultation by business owners.

REFERENCES

[1] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. 2013. Memory-efficient alignment of observed and modeled behavior. *BPM Center Report* 3 (2013), 1–44.

[2] Zhuowei Bao, Sarah Cohen Boulakia, Susan B. Davidson, Anat Eyal, and Sanjeev Khanna. 2009. Differencing provenance in scientific workflows. In *Proceedings of the 25th International Conference on Data Engineering*. Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng (Eds.), IEEE Computer Society, 808–819. DOI: https://doi.org/10.1109/ICDE.2009.103

[3] Zhuowei Bao, Susan B. Davidson, Sanjeev Khanna, and Sudeepa Roy. 2010. An optimal labeling scheme for workflow provenance using skeleton labels. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Ahmed K. Elmagarmid and Divyakant Agrawal (Eds.), ACM, 711–722. DOI: https://doi.org/10.1145/1807167.1807244

[4] Zhuowei Bao, Susan B. Davidson, and Tova Milo. 2011. Labeling recursive workflow executions on-the-fly. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis (Eds.), ACM, 493–504. DOI: https://doi.org/10.1145/1989323.1989375

[5] Catriel Beeri, Anat Eyal, Tova Milo, and Alon Pilberg. 2007. Monitoring business processes with queries. In *Proceedings of the 33rd International Conference on Very Large Data Bases*. Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold (Eds.), ACM, 603–614. Retrieved from http://www.vldb.org/conf/2007/papers/research/p603-beeri.pdf.

[6] Richard Bellman. 1958. On a routing problem. *Quarterly of Applied Mathematics* 16, 1 (1958), 87–90.

[7] Olivier Biton, Sarah Cohen Boulakia, Susan B. Davidson, and Carmem S. Hara. 2008. Querying and managing provenance through user views in scientific workflows. In *Proceedings of the 24th International Conference on Data Engineering*. Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen (Eds.), IEEE Computer Society, 1072–1081. DOI: https://doi.org/10.1109/ICDE.2008.4497516

[8] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional functional dependencies for data cleaning. In *Proceedings of the 23rd International Conference on Data Engineering*. Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis (Eds.), IEEE Computer Society, 746–755. DOI: https://doi.org/10.1109/ICDE.2007.367920

[9] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. 2005. 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*. Fatma Özcan (Ed.), ACM, 143–154. DOI: https://doi.org/10.1145/1066157.1066175

[10] Yang Cao, Wenfei Fan, and Wenyuan Yu. 2013. Determining the relative accuracy of attributes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias (Eds.), ACM, 565–576. DOI: https://doi.org/10.1145/2463676.2465309

[11] Fabio Casati, Malú Castellanos, Norman Salazar, and Umeshwar Dayal. 2007. Abstract process data warehousing. In *Proceedings of the 23rd International Conference on Data Engineering*. Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis (Eds.), IEEE Computer Society, 1387–1389. DOI: https://doi.org/10.1109/ICDE.2007.369018

[12] Thomas Curran, Gerhard Keller, and Andrew Ladd. 1998. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model.* Prentice-Hall, Inc., Upper Saddle River, NJ.

[13] Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. 2012. Aligning event logs and declarative process models for conformance checking. In *Proceedings of the International Conference on Business Process Management.* Alistair Barros, Avigdor Gal, and Ekkart Kindler (Eds.), Lecture Notes in Computer Science, Vol. 7481, Springer, 82–97. DOI: https://doi.org/10.1007/978-3-642-32885-5_6

[14] Daniel Deutch and Tova Milo. 2008. Type inference and type checking for queries on execution traces. *Proceedings of the VLDB Endowment* 1, 1 (2008), 352–363. DOI: https://doi.org/10.14778/1453856.1453898

[15] Daniel Deutch and Tova Milo. 2009. Evaluating TOP-K queries over business processes. In *Proceedings of the 25th International Conference on Data Engineering*. Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng (Eds.), IEEE Computer Society, 1195–1198. DOI: https://doi.org/10.1109/ICDE.2009.199

[16] Daniel Deutch and Tova Milo. 2009. TOP-K projection queries for probabilistic business processes. In *Proceedings of the 12th International Conference on Database Theory.* Ronald Fagin (Ed.), Vol. 361, ACM, 239–251. DOI: https://doi.org/10.1145/1514894.1514923

[17] Daniel Deutch and Tova Milo. 2011. A quest for beauty and wealth (or, business processes for database researchers). In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems.* Maurizio Lenzerini and Thomas Schwentick (Eds.), ACM, 1–12. DOI: https://doi.org/10.1145/1989284.1989286

[18] Daniel Deutch, Tova Milo, Neoklis Polyzotis, and Tom Yam. 2010. Optimal top-k query evaluation for weighted business processes. *Proceedings of the VLDB Endowment* 3, 1 (2010), 940–951. DOI: https://doi.org/10.14778/1920841.1920960

[19] Luping Ding, Songting Chen, Elke A. Rundensteiner, Jun'ichi Tatemura, Wang-Pin Hsiung, and K. Selçuk Candan. 2008. Runtime semantic query optimization for event stream processing. In *Proceedings of the 24th International Conference on Data Engineering*. Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen (Eds.), IEEE Computer Society, 676–685. DOI: https://doi.org/10.1109/ICDE.2008.4497476

[20] Sebastian Dunzer, Matthias Stierle, Martin Matzner, and Stephan Baier. 2019. Conformance checking: A state-of-the-art literature review. In *Proceedings of the 11th International Conference on Subject-Oriented Business Process Management.* Stefanie Betz (Ed.), ACM, 4:1–4:10. DOI: https://doi.org/10.1145/3329007.3329014

[21] Joost Engelfriet. 1991. Branching processes of petri nets. *Acta Informatica* 28, 6 (1991), 575–591. DOI: https://doi.org/10.1007/BF01463946

[22] Wenfei Fan. 2008. Dependencies revisited for improving data quality. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems.* Maurizio Lenzerini and Domenico Lembo (Eds.), ACM, 159–170. DOI: https://doi.org/10.1145/1376916.1376940

[23] Wenfei Fan. 2018. Constraint-driven database repair. In *Proceedings of the Encyclopedia of Database Systems, Second Edition.* DOI: https://doi.org/10.1007/978-1-4614-8265-9_599

[24] Wenfei Fan, Floris Geerts, and Jef Wijsen. 2011. Determining the currency of data. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems.* Maurizio Lenzerini and Thomas Schwentick (Eds.), ACM, 71–82. DOI: https://doi.org/10.1145/1989284.1989295

[25] Wenfei Fan, Ruochun Jin, Muyang Liu, Ping Lu, Chao Tian, and Jingren Zhou. 2020. Capturing associations in graphs. *Proceedings of the VLDB Endowment* 13, 11 (2020), 1863–1876. Retrieved from http://www.vldb.org/pvldb/vol13/p1863-fan.pdf.

[26] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2010. Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment* 3, 1 (2010), 173–184. DOI: https://doi.org/10.14778/1920841.1920867

[27] Wenfei Fan, Xueli Liu, Ping Lu, and Chao Tian. 2020. Catching numeric inconsistencies in graphs. *ACM Trans. Database Syst.* 45, 2 (2020), 9:1–9:47. DOI: https://doi.org/10.1145/3385031

[28] Wenfei Fan, Ping Lu, Chao Tian, and Jingren Zhou. 2019. Deducing certain fixes to graphs. *Proceedings of the VLDB Endowment* 12, 7 (2019), 752–765. DOI: https://doi.org/10.14778/3317315.3317318

[29] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *Proceedings of the 2016 International Conference on Management of Data.* Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.), ACM, 1843–1857. DOI: https://doi.org/10.1145/2882903.2915232

[30] Daniela Grigori, Fabio Casati, Umeshwar Dayal, and Ming-Chien Shan. 2001. Improving business process quality through exception understanding, prediction, and prevention. In *VLDB 2001, Proceedings of the 27th International Conference on Very Large Data Bases.* Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass (Eds.), Morgan Kaufmann, 159–168. Retrieved from http://www.vldb.org/conf/2001/P159.pdf.

[31] Thomas Heinis and Gustavo Alonso. 2008. Efficient lineage tracking for scientific workflows. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* Jason Tsong-Li Wang (Ed.), ACM, 1007–1018. DOI: https://doi.org/10.1145/1376616.1376716

[32] R. M. Karp. 1972. Reducibility among combinatorial problems. In *Proceedings of the Complexity of Computer Computations*. Plenum Press, 85–103.
[33] Ann Langley. 1999. Strategies for theorizing from process data. *The Academy of Management Review* 24, 4 (1999), 691–710. Retrieved from http://www.jstor.org/stable/259349.
[34] Xixi Lu, Dirk Fahland, and Wil M. P. van der Aalst. 2014. Conformance checking based on partially ordered event data. In *Proceedings of the International Conference on Business Process Management.* Fabiana Fournier and Jan Mendling (Eds.), Lecture Notes in Business Information Processing, Vol. 202. Springer, 75–88. DOI: https://doi.org/10.1007/978-3-319-15895-2_7
[35] Jan Mendling, Hajo A. Reijers, and Wil M. P. van der Aalst. 2010. Seven process modeling guidelines (7PMG). *Information and Software Technology* 52, 2 (2010), 127–136.
[36] T. Murata. 1989. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 77, 4 (1989), 541–580. DOI: https://doi.org/10.1109/5.24143
[37] Marco Pegoraro, Merih Seran Uysal, and Wil M. P. van der Aalst. 2021. Conformance checking over uncertain event data. *Information Systems* 102 (2021), 101810. DOI: https://doi.org/10.1016/j.is.2021.101810
[38] Anne Rozinat and Wil M. P. van der Aalst. 2008. Conformance checking of processes based on monitoring real behavior. *Information Systems* 33, 1 (2008), 64–95.
[39] Benjamin H. Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. Dapper, a large-scale distributed systems tracing infrastructure. (2010).
[40] Shaoxu Song, Hong Cheng, Jeffrey Xu Yu, and Lei Chen. 2014. Repairing vertex labels under neighborhood constraints. *Proceedings of the VLDB Endowment* 7, 11 (2014), 987–998. DOI: https://doi.org/10.14778/2732967.2732974
[41] Shaoxu Song, Fei Gao, Ruihong Huang, and Chaokun Wang. 2022. Data dependencies extended for variety and veracity: A family tree. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2022), 4717–4736. DOI: https://doi.org/10.1109/TKDE.2020.3046443
[42] Peng Sun, Ziyang Liu, Susan B. Davidson, and Yi Chen. 2009. Detecting and resolving unsound workflow views for correct provenance analysis. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul (Eds.), ACM, 549–562. DOI: https://doi.org/10.1145/1559845.1559903
[43] Wil M. P. van der Aalst. 2011. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer. I–XVI, 1–352 pages.
[44] Wil M. P. van der Aalst et al. 2011. Process mining manifesto. In *Proceedings of the Business Process Management Workshops*. 169–194.
[45] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. 2003. Workflow patterns. *Distributed Parallel Databases* 14, 1 (2003), 5–51. DOI: https://doi.org/10.1023/A:1022883727209
[46] C. J. van Rijsbergen. 1979. *Information Retrieval*. Butterworth.
[47] Jianmin Wang, Shaoxu Song, Xuemin Lin, Xiaochen Zhu, and Jian Pei. 2015. Cleaning structured event logs: A graph repair approach. In *Proceedings of the 31st IEEE International Conference on Data Engineering*. Johannes Gehrke, Wolfgang Lehner, Kyuseok Shim, Sang Kyun Cha, and Guy M. Lohman (Eds.), IEEE Computer Society, 30–41. DOI: https://doi.org/10.1109/ICDE.2015.7113270
[48] Jianmin Wang, Shaoxu Song, Xiaochen Zhu, and Xuemin Lin. 2013. Efficient recovery of missing events. *Proceedings of the VLDB Endowment* 6, 10 (2013), 841–852. DOI: https://doi.org/10.14778/2536206.2536212
[49] Jianmin Wang, Shaoxu Song, Xiaochen Zhu, Xuemin Lin, and Jiaguang Sun. 2016. Efficient recovery of missing events. *IEEE Trans. Knowl. Data Eng.* 28, 11 (2016), 2943–2957. DOI: https://doi.org/10.1109/TKDE.2016.2594785
[50] Jef Wijsen. 2005. Database repairing using updates. *ACM Transactions on Database Systems* 30, 3 (2005), 722–768.
[51] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.), ACM, 807–817. DOI: https://doi.org/10.1145/3338906.3338931

Received 4 December 2021; revised 22 September 2022; accepted 2 November 2022