Efficient Recovery of Missing Events

Jianmin Wang, Shaoxu Song, Xiaochen Zhu, Xuemin Lin, *Fellow, IEEE*, and Jiaguang Sun

Abstract—For various entering and transmission issues raised by human or system, missing events often occur in event data, which record execution logs of business processes. Without recovering the missing events, applications such as provenance analysis or complex event processing built upon event data are not reliable. Following the minimum change discipline in improving data quality, it is also rational to find a recovery that minimally differs from the original data. Existing recovery approaches fall short of efficiency owing to enumerating and searching over all of the possible sequences of events. In this paper, we study the efficient techniques for recovering missing events. According to our theoretical results, the recovery problem appears to be NP-hard. Nevertheless, advanced indexing, pruning techniques are developed to further improve the recovery efficiency. The experimental results demonstrate that our minimum recovery approach achieves high accuracy, and significantly outperforms the state-of-the-art technique for up to five orders of magnitudes improvement in time performance.

Index Terms—Data repairing, event data processing, petri net

1 Introduction

Business processes continuously generate huge volume of event data, ranging from traditional enterprise office automation systems or scientific workflows [2], [8] to recent Web services and online transactions [17]. In event data management, provenance analysis [20] identifies the sequence of steps leading to a data item, and complex event processing [5] detects interesting event patterns from data. While querying and mining on event data are widely investigated, the quality of event data itself draws less attention.

According to our survey of real event data recorded by a train manufacturer, at least 47.66 percent events are missed in a total of 4, 470 event sequences. The missing events occur for various reasons, such as (1) forgot to submit when manually recording event logs, (2) suffered from system failures, or (3) mess after collecting the events from heterogeneous execution environment. In this survey, the most typical missing events are routing events (41.43 percent of the 47.66 percent missing events), which pass or distribute tasks (e.g., by the manager) to one or multiple staffs for subsequent processing. Since no changes have been made on products by such routing events, the system did not collect these events (belonging to the aforesaid type (3) missing events). Without recovering such missing routing events, the prerequisites of an event might be absent, e.g., having no idea about which manager the task is passed from. The provenance of a product item, i.e., the sequence of steps used to produce the item, is unlikely to obtain.

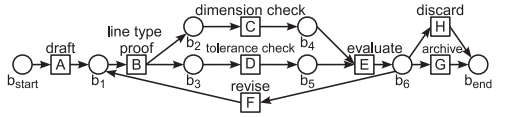
- *J. Wang, S. Song, X. Zhu, and J. Sun are with the Tsinghua National Laboratory for Information Science and Technology, KLiss, MoE, School of Software, Tsinghua University, Beijing 100080, China. E-mail: {jimwang, sxsong, sunjg}@tsinghua.edu.cn, zhu-xc10@mails.tsinghua.edu.cn.*
- *X. Lin is with the University of New South Wales, Sydney, NSW 2052, Australia. E-mail: lxue@cse.unsw.edu.*

Manuscript received 4 Feb. 2015; revised 20 June 2016; accepted 19 July 2016. Date of publication 27 July 2016; date of current version 3 Oct. 2016. Recommended for acceptance by T. Li.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2016.2594785

Moreover, mining patterns over event data is important, since event patterns (with distinct appearance frequencies) can be utilized as discriminative features in matching and identifying heterogeneous events [28]. The basic idea is to enrich the features of singleton events (less discriminative) by more complex event patterns (more discriminative) in event matching (see more details in [27], [28]). As presented in [15], interesting event patterns (a.k.a. episodes) that occur frequently could be discovered from event sequences. For instance, a pattern $\mathsf{SEQ}(\mathsf{B}, \mathsf{AND}(\mathsf{C}, \mathsf{D}), \mathsf{E})$ indicates that after B, two events C and D are executed in parallel, followed by E. It could be discovered with high frequency from a set of sequences <ABCDEH>, <ABDCEH>, <ABCDEG>, . . . , where the pattern appears. Again, without recovering the missing events, e.g., C from the same sources owing to network interrupt, the event patterns could no longer be discovered.

In general, the task of recovering missing events could hardly be performed without any prior knowledge. Fortunately, most business events do not occur randomly. Instead, event data often follow certain business rules or constraints, such as process specifications [5]. We focus on recovering missing events in the light of process specifications.

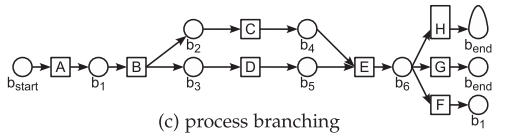
Example 1. Consider a real process specification in Fig. 1a for producing an engineering drawing in a train manufacturer. Each square (namely *transition*) denotes a task in the process specification, e.g., transition A represents a task of drafting. All the arrows attached to a transition denote that the corresponding flows should be executed in parallel. For example, both the dimension checking (task C) and the tolerance checking (task D) should be conducted after line type proofing (task B) in the drawing. Moreover, the process can carry on evaluating the drawing (task E) only if both C and D are executed. Circles in the figure, namely *places* always appearing between transitions, could express choice semantics. That is, when multiple choices are associated, only one of the flows going out a place can be executed. For instance, place b_6



(a) process specification

ID	Sequence of events
1	<A, B, C, D, E, G>
2	<A, B, C, E, G>
3	<A, B, C, D, G>

(b) event log of process execution



(c) process branching

Fig. 1. Example of engineering drawing process.

leads to either revising the drawing (task F), archiving it (task G) or discarding it (task H) after evaluation (E).

An execution of the process generates a sequence of events, where each event corresponds to a task in the process specification. We say that a sequence conforms to the specification if it successfully executes from the source place b_{start} to the sink place b_{end} following the flow constraints in the specification. For example, the first sequence <ABCDEG> in Fig. 1b denotes a complete execution of engineering drawing including steps *drafting*, *line type proofing*, *dimension checking*, *tolerance checking*, *evaluating*, *archiving* from b_{start} to b_{end}.

Owing to various data quality issues, event logs are often incomplete. For instance, the second sequence <ABCEG> has an event D missed during the collection of event logs from the database for dimension checking. Without recovering the missing event D, it is unlikely to find the provenance step of E. Moreover, if such data transmission problems occur frequently in the dimension checking database, an incorrect event pattern without dimension checking step in engineering drawing will be mined.

The minimum change principle is widely considered in data repairing [3], [18], following the intuition that systems or human always try to minimize their mistakes in practice, i.e., to minimally miss events in our scenarios. For instance, to recover the third sequence <ABCDG> in Fig. 1 in Example 1, a minimum recovery <ABCDEG> could be more likely referring to the aforesaid intuition of minimum (missing) mistakes, rather than <ABCDEFBCDEG>, <ABCDEFBCDEFBCDEG>, ... Indeed, as discussed in Example 4, the chance that all events in a loop (e.g., FBCDE in the aforesaid recovery) are missing is low in practice. Consequently, the accuracy of the former minimum recovery could be generally higher than the latter excessive recoveries. Thereby, the minimum recovery is also considered in the previous study of missing events [13].

Challenges: The hardness of recovering the missing events originates from the complexity of the recovery problem. Efficiently obtaining an optimal recovery is not trivial.

We illustrate an example where the sequences cannot be trivially recovered. Consider the process specification presented in Fig. 2, with explicit termination node b_{end} [23]. Each e_i corresponds to an agent of processing some job, which can either skip a job (denoted by s_i) or accept it (by a_i and execute e_i). Suppose that a sequence $<c_1c_2c_3b_1>$ is given. Simply filling the prerequisites of observed events from the process specification, e.g., $<s_1s_2s_3s_4>$ for b_1, may not be able to form a valid recovery. On the other hand, a large number of possible recoveries for the sequence could be enumerated, e.g., $<a_1e_1a_2c_1c_2e_2c_3s_3s_4b_1>$, $<a_1e_1a_2c_1c_2e_2s_3s_4c_3b_1>, \ldots, <s_1a_2e_2s_3s_4c_1c_2c_3b_1>, \ldots$ Efficiently finding the optimal recovery (with the minimum change) is highly non-trivial.

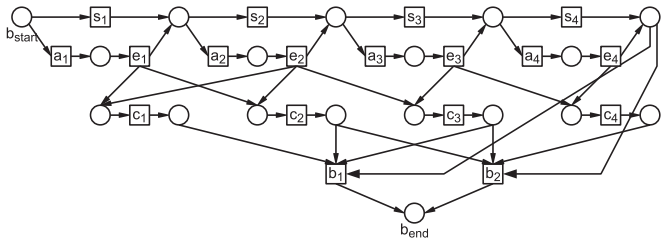


Fig. 2. Example of non-trivial recovery.

Efficiently computing the recovery of missing events is essential and challenging with the following considerations: (1) The number of transitions could be as large as 118 in real process/workflow specifications [7], and 1,000 in event sequences according to our observation in Section 6. The number of possible paths (even w.r.t. specifications without loops) is exponential to the number of transitions [6], [16], and thus may hardly be regarded as a constant. Indeed, infinite sequences of events could be generated when loops exist in process specifications. (2) Real-time business process monitoring [10], e.g., detection of shoplifting, or large/suspicious financial transactions [5], requires recovering the missing events in an online manner. It further motivates us to develop more efficient pruning and heuristics for missing event recovery.

Our main contributions in this paper are summarized as:

- We propose a linear time backtracking algorithm for the recovery of a simple case, where all the events are in parallel execution without any choices.
- We reveal the NP-hardness of finding the minimum recovery of missing events in general settings (with choices). To the best of our knowledge, this is the first study on analyzing the hardness of the missing event recovery problem.
- We present a branching framework for recovery in general cases. A branching index with advanced pruning techniques are developed to speed-up recovery.
- We devise efficient heuristic that performs directly on process specifications without generating branching index. By precomputing possible executions (paths), recovery efficiency is significantly improved. Several typical cases are studied, on which the heuristic shows the best performance.
- Finally, we report the extensive experimental evaluation on real data.

2 Problem Statement

2.1 Preliminaries

Definition 1 (Petri Net). *A Petri net is a triplet $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$, where $\mathcal{P}$ is a finite set of places, $\mathcal{T}$ is a finite set of transitions,*

$\mathcal{P} \cap \mathcal{T} = \emptyset$, and $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is a finite set of directed arcs (flow relation).

For $x \in \mathcal{P} \cup \mathcal{T}$, $\bullet x = \{y \mid (y, x) \in \mathcal{F}\}$ denotes the pre-set of x and $x\bullet = \{y \mid (x, y) \in \mathcal{F}\}$ denotes the post-set of x. The pre/post-set representation can be nested, such as $\bullet(\bullet x)$ denoting $\cup_{y \in \bullet x} \bullet y$, i.e., (union of) the pre-sets of a pre-set.

Definition 2 (Process Specification). *A process specification $\langle \mathcal{N}, \mathsf{b}_{\mathsf{start}}, \mathsf{b}_{\mathsf{end}} \rangle$, or simply denoted by $\mathcal{N}$, is a special Petri net $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$ with a unique source place $\mathsf{b}_{\mathsf{start}} \in \mathcal{P}$, $\bullet \mathsf{b}_{\mathsf{start}} = \emptyset$, and a unique sink place $\mathsf{b}_{\mathsf{end}} \in \mathcal{P}$, $\mathsf{b}_{\mathsf{end}} \bullet = \emptyset$. Each $x \in \mathcal{P} \cup \mathcal{T}$ is on a path from $\mathsf{b}_{\mathsf{start}}$ to $\mathsf{b}_{\mathsf{end}}$.*

Each transition $\mathsf{e} \in \mathcal{T}$ corresponds to an *event* in the execution of the process. An *event sequence* σ, or simply sequence, w.r.t. a process specification $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$ is a finite sequence of events (transitions), i.e., $\sigma \in \mathcal{T}^*$. Each sequence logs an execution of the process defined by $\mathcal{N}$.

Definition 3 (Firing Sequence). *A* firing sequence *of a process specification $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$, and its post-set which is a multi-set of places, are defined recursively as follows:*

1) *The empty sequence ε is a firing sequence, and $\varepsilon\bullet = \{\mathsf{b}_{\mathsf{start}}\}$;*
2) *If σ is a firing sequence, $\mathsf{e} \in \mathcal{T}$ is a transition (event), and $\bullet\mathsf{e} \subseteq \sigma\bullet$, then $\sigma\mathsf{e}$ is also a firing sequence, and $(\sigma\mathsf{e})\bullet = (\sigma\bullet) \cup (\mathsf{e}\bullet) \backslash (\bullet\mathsf{e})$.*

A sequence σ is said *conforming* to a process specification, denoted by $\sigma \models \mathcal{N}$, if σ is a firing sequence w.r.t. $\mathcal{N}$ and $\{\mathsf{b}_{\mathsf{end}}\} \subseteq \sigma\bullet$ that successfully executes to $\mathsf{b}_{\mathsf{end}}$.

Example 2 (Example 1 Continued). Consider the process specification $\mathcal{N}$ in Fig. 1 and a sequence $\sigma =$ <ABCDEG>. To investigate whether σ is a firing sequence with respect to $\mathcal{N}$, we start from the empty sequence ε with $\varepsilon\bullet = \{\mathsf{b}_{\mathsf{start}}\}$. Since the first event A has $\bullet\mathtt{A} = \{\mathsf{b}_{\mathsf{start}}\} \subseteq \varepsilon\bullet$, the augmentation <A> is also a firing sequence with $<\mathtt{A}>\bullet = \{\mathsf{b}_1\}$. It follows $<\mathtt{AB}>\bullet = \{\mathsf{b}_2, \mathsf{b}_3\}$. For the next C, as $\bullet\mathtt{C} = \{\mathsf{b}_2\} \subseteq <\mathtt{AB}>\bullet$, the firing sequence becomes <ABC> with post-set $\{\mathsf{b}_3, \mathsf{b}_4\}$. Similarly, we have $<\mathtt{ABCD}>\bullet = \{\mathsf{b}_4, \mathsf{b}_5\}$ by appending D. As $\bullet\mathtt{E} = \{\mathsf{b}_4, \mathsf{b}_5\}$ and $\mathtt{E}\bullet = \{\mathsf{b}_6\}$, it leads to $<\mathtt{ABCDE}>\bullet = \{\mathsf{b}_6\}$, and finally $\{\mathsf{b}_{\mathsf{end}}\} \subseteq <\mathtt{ABCDEG}>\bullet$. Therefore, the sequence <ABCDEG> conforms to the specification.

Missing events may occur either in the start/middle of a sequence which prevent it from being a firing sequence, or at the end of the sequence having $\{\mathsf{b}_{\mathsf{end}}\} \not\subseteq \sigma\bullet$.

Definition 4 (Gap). *Let σ_k be a firing sequence. For the next event (transition) e, if $\bullet\mathsf{e} \not\subseteq \sigma_k\bullet$, we call $(\sigma_k\bullet, \bullet\mathsf{e})$ a* gap *with at least one missing event between σ_k and e.*

Example 3 (Example 2 Continued). Let us consider another sequence <ABCEG>. As illustrated, <ABC> is a firing sequence with $<\mathtt{ABC}>\bullet = \{\mathsf{b}_3, \mathsf{b}_4\}$. For the next event E, however, we have $\bullet\mathtt{E} = \{\mathsf{b}_4, \mathsf{b}_5\} \not\subseteq <\mathtt{ABC}>\bullet$. Thereby, there is a gap between <ABC> and E, where an event D is missing indeed.

Moreover, a sequence <ABCDE> is a firing sequence but does not conform to the specification as $\{\mathsf{b}_{\mathsf{end}}\} \not\subseteq <\mathtt{ABCDE}>\bullet = \{\mathsf{b}_6\}$. At least one event is missing at the end of the sequence.

2.2 Problem Definition

We call σ a subsequence of σ', if σ is a sequence that can be derived from σ' by deleting some events without changing the order of the remaining events.

Definition 5. *A* recovery *of a sequence σ w.r.t. a process specification $\mathcal{N}$ is also a sequence σ', such that σ is a subsequence of σ' and $\sigma' \models \mathcal{N}$, i.e., the recovered sequence σ' conforms to process specification $\mathcal{N}$.*

The distance between σ' and σ is given by $\Delta(\sigma', \sigma) = |\sigma'| - |\sigma|$. Following the minimum change principle in improving data quality [3], we find a recovery that minimally differs from the original sequence. Multiple minimal recoveries possibly exist. However, these multiple minimal recoveries could be redundant (i.e., equivalent). For instance, to recover the sequence <ABCEG> in Example 1, the results <ABCDEG> and <ABDCEG> have no difference w.r.t. the process specification, as C and D are executed in parallel after B and before E. An event pattern $\mathsf{SEQ}(\mathsf{B}, \mathsf{AND}(\mathsf{C}, \mathsf{D}), \mathsf{E})$ could be discovered from either the recovery result <ABCDEG> or <ABDCEG>. Referring to the aforesaid redundancy and equivalence in pattern mining application, we practically find one of the minimal recovery sequences, rather than reporting all possible minimal recovery sequences.

Besides the equivalent cost function on counting the number of insertions, it is possible to employ fine-grained cost functions by considering the weight of each recovered event, e.g., denoting the confidence of the event being lost. We leave this interesting direction, including how to determine the weights of recovered events, as future studies.

Problem 1. *Given a sequence σ over a process specification $\mathcal{N}$, σ is not conforming with $\mathcal{N}$, the minimum recovery problem is to find a recovery σ' of σ such that $\sigma' \models \mathcal{N}$ and the distance $\Delta(\sigma', \sigma)$ between σ' and σ is minimized.*

Example 4 (Example 3 Continued). To fill gap $(<\mathtt{ABC}>\bullet, \bullet\mathtt{E})$ in a sequence <ABCEG>, we look for the next event w.r.t. firing sequence. There is only one candidate D whose preset is involved in the post-set $<\mathtt{ABC}>\bullet$. The firing sequence carries on with the new post-set $<\mathtt{ABCD}>\bullet = \{\mathsf{b}_4, \mathsf{b}_5\}$. It matches with $\bullet\mathtt{E}$ and finally returns a recovery <ABCDEG> conforming to the specification. A recovery <ABCDEFBCDEG> with a complete loop between <ABC> and E is not minimal. According to our survey of the real event data recorded by a train manufacturer, 47.66 percent events are missing. It could be interpreted as a probability 0.4766 of an event missing. The probability that all the m events in a loop are missing is thus 0.4766^m. For instance, for the loop <BCDEF> in Fig. 1, the probability of all five events missing is 0.0245. In other words, the chance that all events in a loop are missing is low in practice.

For the sequence <ABCDE>, which is already a firing sequence, the recovery can directly move on (by considering all possible alternatives w.r.t. firing sequence) till $\mathsf{b}_{\mathsf{end}}$ is reached. Since there are two candidate events G/H to carry on, both <ABCDEG> and <ABCDEH> could be returned as the minimum recovery.

For the special case of process specifications without loops, we can represent the minimum recovery problem by

integer linear programming (ILP). Since no loop exists, an event appears in a sequence at most once. Let $x_i = 1$ denote that an event $\mathsf{e}_i \in \mathcal{T}, i = 1, \ldots, |\mathcal{T}|$ appears in sequence σ. The minimum recovery problem is thus

$$\begin{aligned} \min \quad & \sum_{i=1}^{|\mathcal{T}|} x_i \\ \text{s.t.} \quad & \sum_{\mathsf{e}_j \in \bullet \mathsf{b}} x_j = x_i \qquad \mathsf{b} \in \bullet \mathsf{e}_i, 1 \le i \le |\mathcal{T}| \\ & \sum_{\mathsf{e}_j \in \mathsf{b} \bullet} x_j = x_i \qquad \mathsf{b} \in \mathsf{e}_i \bullet, 1 \le i \le |\mathcal{T}| \\ & \sum_{i=1}^{|\mathcal{T}|} x_i > 0 \\ & x_i \in \{0, 1\} \\ & x_i = 1 \end{aligned}$$

By inserting e_i into σ following the topological ordering of events in the process specification, where $\mathsf{e}_i \notin \sigma$ and $x_i = 1$ in the solution of the aforesaid ILP, we obtain the minimum recovery of σ. For the general case of specifications with loops, where events may appear multiple times in a sequence with distinct orders, it is unclear whether the requirements on such orders specified by Petri nets can be captured by linear constraints. The ILP formalization of the minimum recovery problem is still open.

2.3 Hardness Analysis

Owing to choices and parallelization of flows, there are vast alternatives to enumerate in the recovery. de Leoni et. al. [4] propose an alignment based solution with exponential time complexity but fail to uncover the hardness of the problem.

As one of our major contributions, we find that generating the optimal recovery of missing events is NP-hard. In other words, it is NP-complete to determine whether a recovery exists with distance less than a certain constant.

Theorem 1. *Given a sequence σ over a process specification $\mathcal{N}$ and a constant k, the problem to determine whether there exists a recovery σ' of σ s.t. $\sigma' \models \mathcal{N}$ and $\Delta(\sigma', \sigma) \le k$ is NP-complete.*

At the end of Section 2.2, we have shown that the minimum recovery problem can be represented by integer linear programming, which is one of Karp's 21 NP-complete problems [11]. Therefore, it is not surprising to obtain the hardness result in Theorem 1.

3 Getting Started on Causal Net

Let us start from a simple special case of process specifications where no choices of flows exist. Such a special case is interesting for two reasons. First, as we will see soon, it can be easily extended to more general cases with choices. Second, the existing aligning approach [4] even fails to perform efficiently in this simple case.

A *causal net* is a Petri net $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$, such that for every $\mathsf{b} \in \mathcal{P}, |\bullet \mathsf{b}| \le 1$ and $|\mathsf{b} \bullet| \le 1$. According to the definition of process specifications, only the $\mathsf{b}_{\mathsf{start}}/\mathsf{b}_{\mathsf{end}}$ places can have empty pre/post-sets. The remaining places have exactly one in degree and one out degree, respectively. We can equivalently represent the causal net as a directed acyclic graph (DAG), where transitions (events) in $\mathcal{T}$ denote vertexes, and places with $|\bullet \mathsf{b}| = |\mathsf{b} \bullet| = 1$ are edges.

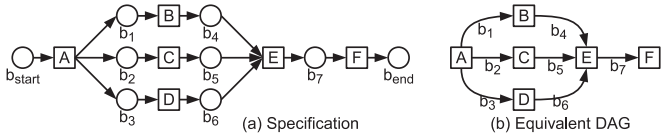


Fig. 3. Example of causal net.

Example 5. Consider the process specification in Fig. 3a, which is a causal net. All the places $\mathsf{b}_1, \ldots, \mathsf{b}_7$ other than $\mathsf{b}_{\mathsf{start}}/\mathsf{b}_{\mathsf{end}}$ have $|\bullet \mathsf{b}_i| = |\mathsf{b}_i \bullet| = 1$, for instance, $\bullet \mathsf{b}_1 = \{\mathtt{A}\}, \mathsf{b}_1 \bullet = \{\mathtt{B}\}$. b_1 can be interpreted as an edge between A and B as illustrated in Fig. 3b. Following the same line, the causal net can be represented as a DAG.

Since there is no multiple choice in a causal net process specification, as illustrated in Example 5, the conformance of a sequence to the process specification can be checked by inspecting whether all the order relationships between events are satisfied. It leads to Lemma 2 stating that a sequence conforms to the process specification iff the sequence is a topological ordering of the corresponding DAG.

Lemma 2. *Given a sequence σ over a causal net specification $\mathcal{N}$, the checking of conformance is equivalent to validate whether the sequence σ is one of the topological sorts on the DAG of $\mathcal{N}$.*

Proof. The lemma can be easily verified. First, according to the definition of firing sequence, an event e can be executed only when all the places in $\bullet \mathsf{e}$ appear in the post-set of the current firing sequence, say $\sigma \bullet$. Since any place except $\mathsf{b}_{\mathsf{start}}$ has $|\bullet \mathsf{b}| = 1$ in the causal net, each place in $\bullet \mathsf{e}$ corresponds to an event in σ. It follows that $\bullet(\bullet \mathsf{e}) \subseteq \sigma$. In other words, all the prerequisites of e in the DAG have been conducted in σ. Moreover, since there is no choice in a causal net, all the transitions (events) in $\mathcal{T}$ should be performed. Therefore, each firing sequence with post-set $\{\mathsf{b}_{\mathsf{end}}\}$ corresponds to a topological sort in the DAG of $\mathcal{N}$. □

Lemma 2 states that any sequence conforming to the process specification must be a topological sort of the corresponding DAG. According to Lemma 2, any topological sort σ', subsuming σ as a subsequence, must be a valid recovery of σ. Lemma 3 further state that such a topological sort σ' is always the optimal solution (minimum recovery) as well.

Lemma 3. *For an incomplete sequence σ over a casual net $\mathcal{N}$, any topological sort σ' on the DAG of $\mathcal{N}$ such that σ is a subsequence of σ' is always the minimum recovery of σ.*

Proof. As each topological sort outputs a vertex exactly once, all the recoveries must have the same size, in other words, having the same minimum distance to σ. □

The existing alignment approach [4] considers the space of all possible firing sequences. That is, it enumerates all the possible topological sorts with respect to the causal. As another contribution of our study, we have indicated that any topological sort is an optimal solution and there is no need to enumerate all of them.

Example 6 (Example 5 Continued). To recover a sequence <AEF> over the process specification in Fig. 3a, the existing aligning approach enumerates all

possible combinations of events in parallel, i.e., <ABCDEF>, <ABDCEF>, <ACBDEF>, <ACDBEF>, <ADBCEF> and <ADCBEF>. According to Lemma 3, however, any topological sort is a minimum recovery, e.g., <ABCDEF> with the minimum distance three to the input sequence <AEF>. There is no need to enumerate other redundant recoveries.

A Backtracking Idea: Motivated by the defeat of enumerating unnecessary firing sequences, a backtracking strategy is used to find a topological sort as the optimal recovery. We introduce how to fill a gap in a sequence, and find a recovery by checking all gaps in one pass through the sequence.

Definition 6 (Fill). *For a gap* $(\sigma\bullet, \bullet\mathsf{e})$*, we call a transition (event) sequence* $\tau \in \mathcal{T}^*$ *a* fill *of the gap, if it ensures 1)* $\sigma\tau$ *is a firing sequence, and 2)* $\bullet\mathsf{e} \subseteq (\sigma\tau)\bullet$.

Algorithm 1 $gap(\sigma\bullet, \bullet\mathsf{e})$ fills the gap between post-set places $\sigma\bullet$ and the following event e. As shown in Lines 2-5, let X denote the places in $\bullet\mathsf{e}$ but not in $\sigma\bullet$. According to the definition of firing sequence, events in $\bullet X$ are necessary to execute e, which are not observed in the current sequence σ, i.e., missing events. The program checks and adds each event $\mathsf{e}_i \in \bullet X$ into τ in Line 9. It fills the gap between $\sigma\tau$ and e_i by recursively calling the function in Line 8.

Algorithm 1. Gap$(\sigma\bullet, \bullet\mathsf{e})$

Input: A firing sequence post-set $\sigma\bullet$ and a transition pre-set $\bullet\mathsf{e}$
Output: A fill of the gap between σ and e

```
1: τ := ε
2: X := •e
3: for each place b_i ∈ X do
4:    if b_i ∈ σ• then
5:       X := X − b_i
6: if X ≠ ∅ then
7:    for each transition e_i ∈ •X, e_i ∉ τ do
8:       τ' := GAP((στ)•, •e_i)
9:       τ := ττ'e_i
10: return τ
```

The correctness of Algorithm 1 is ensured by showing that the produced result $\sigma\tau\mathsf{e}$ is always a firing sequence. First, if there is no gap between $(\sigma\bullet, \bullet\mathsf{e})$, i.e., $X = \emptyset$ in Line 6, it returns $\tau = \varepsilon$ and $\sigma\mathsf{e}$ is a firing sequence. Otherwise, for each $\mathsf{e}_i \in \bullet X$, it either has been included in the current τ, or generates a fill between e_i and the firing sequence $\sigma\tau$ w.r.t. the current τ. According to topological sorting, as long as the prerequisite relationship is guaranteed, the order of inserting e_i will not affect τ being a firing sequence.

Example 7 (Example 6 continued). To recover the gap between a firing sequence <A> and event E, the program generates a set of places $X = \{\mathsf{b}_4, \mathsf{b}_5, \mathsf{b}_6\}$ in $\bullet$E but not in $<\mathtt{A}>\bullet = \{\mathsf{b}_1, \mathsf{b}_2, \mathsf{b}_3\}$. For each event in $\bullet X = \{\mathtt{B}, \mathtt{C}, \mathtt{D}\}$, e.g., B, we fill the gap between the current firing sequence $<\mathtt{A}>\bullet$ and $\bullet$B. It outputs a firing sequence <AB> with post-set $<\mathtt{AB}>\bullet = \{\mathsf{b}_4, \mathsf{b}_2, \mathsf{b}_3\}$. Next, by inserting $\mathtt{C} \in \bullet X$, we have $<\mathtt{ABC}>\bullet = \{\mathsf{b}_4, \mathsf{b}_5, \mathsf{b}_3\}$. It follows $<\mathtt{ABCD}>\bullet = \{\mathsf{b}_4, \mathsf{b}_5, \mathsf{b}_6\}$. Finally, the gap between <A> and E is filled by <BCD> such that $\bullet\mathtt{E} \subseteq <\mathtt{ABCD}>\bullet$.

Finally, we recover a sequence over a causal net specification by iteratively calling the GAP$(\sigma'\bullet, \bullet\sigma[k])$ function for each event $\sigma[k]$ in σ, where σ' is the current firing sequence. Initially, σ' is empty, i.e., $\tau = gap(\{\mathsf{b}_{\mathsf{start}}\}, \bullet\sigma[1])$, and at the end $\tau = \text{GAP}(\sigma'\bullet, \{\mathsf{b}_{\mathsf{end}}\})$ leads the firing sequence σ' to the sink place. The minimum recovery $\sigma'\tau$ is computed.

The complexity is linear in the number of transitions and places in the specification. The backtracking visits each vertex (event) at most once by trying possible edges (places). It is obvious to see the complexity is $O(|\mathcal{T}| + |\mathcal{P}|)$.

4 The Branching Framework

Now, we consider a general process specification with both choices and parallelization of flows. Differently from causal net, there will be multiple choices of execution flows. A straightforward idea is to index all valid flows by branching over choice nodes [6], [16], where each branch denotes a causal net without any choice. By applying the GAP algorithm on the causal net, we find a minimal recovery for each branch, if it exists. The minimum recovery can be found by traversing all the possible branches returned by index.

Let us first introduce the idea of branching over a process specification. While a causal net requires both the in-degree $|\bullet\,\mathsf{p}\,\bullet| \leq 1$ and the out-degree $|\mathsf{p}\,\bullet\,\bullet| \leq 1$, an occurrence net only needs the in-degree to be $|\bullet\,\mathsf{p}| \leq 1$ for each place. There are choice-splits in process branching but no choice-join. To establish the relationship between process branching and specification, we consider the mapping below.

Definition 7 (Homomorphism). *A* homomorphism *from a net* $\mathcal{N}_u(\mathcal{P}_u, \mathcal{T}_u, \mathcal{F}_u)$ *to another net* $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$ *is a mapping* $\pi : \mathcal{P}_u \cup \mathcal{T}_u \to \mathcal{P} \cup \mathcal{T}$ *such that*

1) $\pi(\mathcal{P}_u) \subseteq \mathcal{P}$ *and* $\pi(\mathcal{T}_u) \subseteq \mathcal{T}$,
2) *for every* $\mathsf{t} \in \mathcal{T}_u$, $\pi(\bullet\mathsf{t})$ *is a bijection between* $\bullet\mathsf{t}$ *and* $\bullet\pi(\mathsf{t})$, *and* $\pi(\mathsf{t}\bullet)$ *is a bijection between* $\mathsf{t}\bullet$ *and* $\pi(\mathsf{t})\bullet$.

A net $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$ can be unfolded, where each branch corresponds to a non-choice execution, i.e., a causal net.

Definition 8 (Process Branching). *A* process branching *of a specification* $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$ *is a pair* $(\mathcal{N}_u, \pi)$, *where*

1) $\mathcal{N}_u(\mathcal{P}_u, \mathcal{T}_u, \mathcal{F}_u)$ *is an* occurrence net, *such that for every* $\mathsf{p} \in \mathcal{P}_u, |\bullet\,\mathsf{p}| \leq 1$,
2) π *is a homomorphism from* $\mathcal{N}_u$ *to* $\mathcal{N}$, *and*
3) *for every* $\mathsf{t}_1, \mathsf{t}_2 \in \mathcal{T}_u$, *if* $\bullet\mathsf{t}_1 = \bullet\mathsf{t}_2$ *and* $\pi(\mathsf{t}_1) = \pi(\mathsf{t}_2)$, *then* $\mathsf{t}_1 = \mathsf{t}_2$.

We explain the relationships of four different definitions over Petri net. Process specification in Definition 2 is a special type of Petri net (Definition 1), where a unique source place and a unique sink place are associated to denote the start and end of a process, respectively. Causal net in Section 3 is a simple special case of process specifications where no choices of flows exist. Since there is no choice of multiple alternatives, the causality relationships between events are certain. It is the reason why such a net is called a causal net. Occurrence net employed in Definition 8 is a special form of Petri net, which involves only choice-splits but no choice-join. It enumerates all the possible "occurrences" of various flows specified in a process specification. These definitions and terminology are adopted from [6].

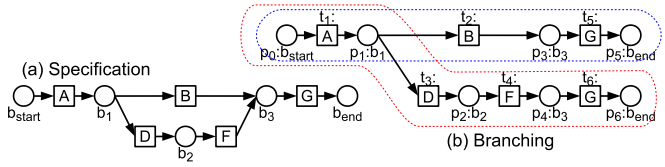


Fig. 4. Example of branching.

Let $AN(\mathsf{p}_i)$ denote all the places and transitions that are ancestors of p_i in the branching net.

Proposition 4. *Each branch in the branching net $\mathcal{N}_u$ is a causal net $\mathcal{N}_b(\mathcal{P}_b, \mathcal{T}_b, \mathcal{F}_b)$ leading to a unique sink place $\mathsf{p}_i : \mathsf{b}_{\mathsf{end}}$ having $\mathcal{P}_b \subseteq \mathcal{P}_u$, $\mathcal{T}_b \subseteq \mathcal{T}_u$, $\mathcal{F}_b \subseteq \mathcal{F}_u$, $\mathcal{P}_b \cup \mathcal{T}_b = AN(\mathsf{p}_i)$.*

Proof. According to the requirement of occurrence net, only one place in the branching can be mapped to $\mathsf{b}_{\mathsf{start}}$, while there may be multiple places mapping to $\mathsf{b}_{\mathsf{end}}$. Each sink place p_i having $\pi(\mathsf{p}_i) = \mathsf{b}_{\mathsf{end}}$ denoted as $\mathsf{p}_i : \mathsf{b}_{\mathsf{end}}$ exactly corresponds to a branch, i.e., a causal net projected by recursively backtracking all the pre-set nodes from $\mathsf{p}_i : \mathsf{b}_{\mathsf{end}}$ to $\mathsf{p}_0 : \mathsf{b}_{\mathsf{start}}$. It involves all the places and transitions that are ancestors of p_i in the branching net, i.e., $AN(\mathsf{p}_i)$. □

A straightforward idea is to apply the aforesaid topological sort based recovery on each branch, and return the one with the minimum size among all the branches.

Example 8. Consider the process specification $\mathcal{N}$ in Fig. 4a. The corresponding process branching as illustrated in Fig. 4b is an occurrence net with only choice-splits but no choice-joins. According to the homomorphism π, each node in the branching maps to a node in the specification, such as $\pi(\mathsf{t}_1) = \mathtt{A}$ denoted as $\mathsf{t}_1 : \mathtt{A}$ or $\pi(\mathsf{p}_1) = \mathsf{b}_1$ denoted by $\mathsf{p}_1 : \mathsf{b}_1, \mathsf{p}_1 \in \mathcal{P}_u, \mathsf{b}_1 \in \mathcal{P}$.

There are two places p_5 and p_6 mapping to $\mathsf{b}_{\mathsf{end}}$, which correspond to two branches ended with $\mathsf{p}_5 : \mathsf{b}_{\mathsf{end}}$ and $\mathsf{p}_6 : \mathsf{b}_{\mathsf{end}}$, respectively. Each branch is a causal net obtained by a projection on all the ancestors of an end place, such as $(\mathsf{p}_0 \to \mathsf{t}_1 \to \mathsf{p}_1 \to \mathsf{t}_2 \to \mathsf{p}_3 \to \mathsf{t}_5 \to \mathsf{p}_5)$ as presented in dotted line in Fig. 4b.

For a sequence $\sigma =$ <AG>, we call the GAP function on each branch. The first branch returns a recovery <ABG>, while the second one outputs <ADFG>. Referring to the minimum distance principle, <ABG> is returned as the minimum recovery of <AG>.

4.1 Branching Approach

Intuitively, there is no need to trying all the branches, especially on those not containing the events of the input sequence. For instance, to recover a sequence <AF>, the first branch with events ABG is not necessary to be considered, as it would never generate a sequence containing event F. Motivated by this, we construct an index on branching to efficiently identify potentially valid branches.

4.1.1 Index on Branches

For any event $\mathsf{e} \in \mathcal{T}$, we can identify all the transitions t in the branching net $\mathcal{N}_u$, whose $\pi(\mathsf{t}) = \mathsf{e}$, denoted by $\mathcal{T}_u(\mathsf{e})$. It is worth noting that two different events sharing the same name is not allowed in a process specification. Referring to the third term in Definition 8 of branching, duplicate events should not appear in a branch as well.[1]

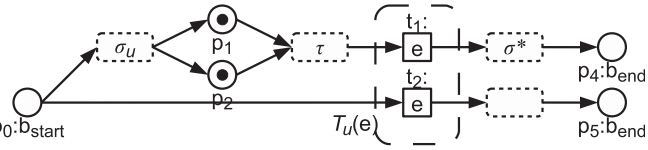


Fig. 5. Index on branches.

Proposition 5. *No events appear twice in one branch.*

Each $\mathsf{t} \in \mathcal{T}_u(\mathsf{e})$ identifies all the branches that may output a recovery containing e. Other branches can be pruned.

The size of index is linear on the number of transitions in $\mathcal{T}_u$. Let d be the maximum out-degree of a place in the specification $\mathcal{N}$. There are $O(d^{|\mathcal{P}|-1})$ branches. The number of places $|\mathcal{P}_u|$ in the branching net is bounded by $O(d^{|\mathcal{P}|})$. In an occurrence net, each place can have at most one transition. The number of transitions $|\mathcal{T}_u|$ in a branching net should not be greater than $|\mathcal{P}_u|$, i.e., bounded by $O(d^{|\mathcal{P}|})$.

4.1.2 Branch Algorithm

To fill gaps on possible branches, we consider the firing sequence σ_u of transitions $\mathcal{T}_u$ in the branching net instead of the specification net. According to the homomorphism mapping, σ_u can be transformed to a firing sequence $\sigma = \pi(\sigma_u)$ with respect to the specification.

Algorithm 2 presents the recovery with branching. Given a current firing sequence σ_u in the branching net as illustrated in Fig. 5, let $\sigma[k] : \mathsf{e}$ be the kth (current) event in the input sequence. For each transition $\mathsf{t} \in \mathcal{T}_u(\mathsf{e})$, we call the GAP$(\sigma_u\bullet, \bullet\mathsf{t})$ function for a causal net. It returns a fill τ between σ_u and t, if it exists, i.e., the new firing sequence $\sigma_u\tau\mathsf{t}$. The program carries on by recursively branching on the next event $\sigma[k+1]$, in Line 11 BRANCH$(\sigma_u\tau\mathsf{t}, \sigma[k+1])$. Finally, the minimum recovery is obtained by transforming the results $\sigma_{\min} =$ BRANCH$(\varepsilon, \sigma[1])$ to $\pi(\sigma_{\min})$.

Algorithm 2. Branch$(\sigma_u, \sigma[k])$

Input: A firing sequence σ_u and the kth event $\sigma[k]$ in σ
Output: A minimum recovered sequence after σ_u

```
1: σ_min := an infinite sequence
2: if σ[k] is null then
3:   for each sink place p ∈ P_u, π(p) = b_end do
4:     τ := GAP(σ_u•, {p})
5:     if |σ_min| > |τ| then
6:       σ_min := τ
7: else
8:   for each t ∈ T_u(σ[k]) do
9:     τ := GAP(σ_u•, •t)
10:    if τ exists then
11:      σ* := BRANCH(σ_u τ t, σ[k+1])
12:      if |σ_min| > |τ t σ*| then
13:        σ_min := τ t σ*
14: return σ_min
```

In the worst case, the program needs to traverse all the transitions in branching. In each iteration, the GAP function

1. The case of process specifications with loops needs further instruments to ensure non-duplicate events.

visits at most $O(|\mathcal{T}| + |\mathcal{P}|)$ places and transitions, according to Proposition 5. Referring to the size analysis of branching net, the complexity of Algorithm 2 is $O((|\mathcal{T}| + |\mathcal{P}|) \cdot d^{|\mathcal{P}|})$, where d is the maximum out-degree of a place in $\mathcal{N}$.

Example 9 (Example 8 Continued). For a sequence $\sigma = <\mathtt{F}>$, we call the BRANCH$(\sigma_u, \sigma[1])$, where $\sigma_u = \varepsilon, \sigma[1] = \mathtt{F}$. It first locates possible branches via $\mathcal{T}_u(\sigma[1]) = \{t_4\}$. There is only one branch containing F in Fig. 4b. The GAP$(\varepsilon\bullet, \bullet t_4)$ function returns a fill $\tau = <\mathsf{t}_1\mathsf{t}_3>$ of the gap between ε and t_4. It follows the branching on the next event $\sigma[2]$, i.e., null. A recovered sequence $\sigma^* =$ BRANCH$(<\mathsf{t}_1\mathsf{t}_3\mathsf{t}_4>, \sigma[2]) = <\mathsf{t}_6>$ is returned. Finally, the minimum recovered sequence is $\sigma = <\mathsf{t}_1\mathsf{t}_3\mathsf{t}_4\mathsf{t}_6>$ and transformed to <ADFG> as the minimum recovery.

4.2 Local Optimality

In general, for any intermediate event e, we cannot obtain the minimal recovery on the branches w.r.t. e until all these branches are fully computed. According to the intuition of firing sequence semantics, however, the branching produced by any two firing sequences with the same post-sets should be exactly the same. In the following, we identify those $\mathsf{t} \in \mathcal{T}_u(\sigma[k])$ that may lead to firing sequences with the same post-set, and prove that a local optimal result could be generated by branching on only one of the transitions.

4.2.1 Branching Equivalence Classes

To define groups of transitions in $\mathcal{T}_u(\sigma[k])$, we introduce a binary relation on transitions, namely *branching equivalence relation*, denoted as $\triangleq$. Let σ_1 be a *minimum prefix firing sequence* of a transition t_1, which only consists of transitions that are ancestor transitions of t_1 and can form a new firing sequence $\sigma_1\mathsf{t}_1$. This σ_1 equals to the minimum fill of the gap from an empty firing sequence ε to t_1, i.e., GAP$(\varepsilon\bullet, \bullet\mathsf{t}_1)$.

Definition 9 (Branching Equivalence Relation). *For any two transitions $\mathsf{t}_1, \mathsf{t}_2 \in \mathcal{T}_u$, let σ_1 and σ_2 be the minimum prefix firing sequences of t_1 and t_2, respectively. Then, t_1 and t_2 are said* branching equivalent, *denoted by $\mathsf{t}_1 \triangleq \mathsf{t}_2$ iff: 1) $\pi(\mathsf{t}_1) = \pi(\mathsf{t}_2)$; and 2) $\pi((\sigma_1\mathsf{t}_1)\bullet) = \pi((\sigma_2\mathsf{t}_2)\bullet)$.*

Obviously, relation $\triangleq$ is reflexive, symmetric and transitive. We define the branching equivalence classes as follows.

Definition 10 (Branching Equivalence Classes). *For an event e, the transitions $\mathcal{T}_u(\mathsf{e})$ can be divided into a collection of n subsets, $\{\mathcal{T}_1^{EC}, \mathcal{T}_2^{EC}, \ldots, \mathcal{T}_n^{EC}\}$, namely* branching equivalence classes, *such that,*

1) $\mathcal{T}_i^{EC} \cap \mathcal{T}_j^{EC} = \emptyset, i \neq j$;
2) $\bigcup_{i=1}^{n} \mathcal{T}_i^{EC} = \mathcal{T}_u(\mathsf{e})$;
3) $\mathsf{t}_1 \triangleq \mathsf{t}_2, \forall \mathsf{t}_1, \mathsf{t}_2 \in \mathcal{T}_i^{EC}$;
4) $\mathsf{t}_1 \not\triangleq \mathsf{t}_2, \forall \mathsf{t}_1 \in \mathcal{T}_i^{EC}, \forall \mathsf{t}_2 \in \mathcal{T}_j^{EC}$ *and* $i \neq j$.

Example 10 (Example 8 Continued). Consider the transitions $\mathsf{t}_5, \mathsf{t}_6$ in Fig. 4b. Let $\sigma_1 = <\mathsf{t}_1\mathsf{t}_2>$ and $\sigma_2 = <\mathsf{t}_1\mathsf{t}_3\mathsf{t}_4>$ be the minimum prefix firing sequences of t_5 and t_6, respectively, i.e., the minimum fills from empty firing sequence ε to t_5 and t_6. As $\pi(\mathsf{t}_5) = \pi(\mathsf{t}_6) = \mathtt{G}$ and

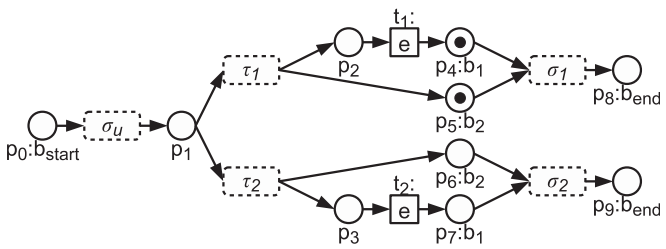


Fig. 6. Local optimality.

$\pi((\sigma_1\mathsf{t}_5)\bullet) = \pi(\{\mathsf{p}_3\}) = \{\mathsf{b}_3\} = \pi(\{\mathsf{p}_4\}) = \pi((\sigma_2\mathsf{t}_6)\bullet)$, we say $\mathsf{t}_5, \mathsf{t}_6$ in the same branching equivalence class having $\mathsf{t}_5 \triangleq \mathsf{t}_6$.

For each branching equivalence class $\mathcal{T}_i^{EC}$, we find a $\mathsf{t} \in \mathcal{T}_i^{EC}$ which has a minimum fill $\tau_{\min}$ for the gap between σ_u and t compared with those of other t′ in $\mathcal{T}_i^{EC}$. Below, we will show that this local minimal recovery $\sigma_u\tau_{\min}\mathsf{t}$ can always lead to a minimum recovery with respect to all the branches on transitions in $\mathcal{T}_i^{EC}$, i.e., the local optimality.

4.2.2 Proof of Local Optimality

First, we can show that, for any σ_u, the post-sets of firing sequences generated by filling the gaps between σ_u and the transitions in a branching equivalent class must map to the same set of places in specification.

Lemma 6. *Given a firing sequence σ_u, for any $\mathsf{t}_1, \mathsf{t}_2 \in \mathcal{T}_i^{EC}$, let $\tau_1 =$ GAP$(\sigma_u\bullet, \bullet\mathsf{t}_1)$ and $\tau_2 =$ GAP$(\sigma_u\bullet, \bullet\mathsf{t}_2)$ be the fills from σ_u to t_1 and t_2, respectively. Then, $\pi((\sigma_u\tau_1\mathsf{t}_1)\bullet) = \pi((\sigma_u\tau_2\mathsf{t}_2)\bullet)$ always holds.*

For example, in Fig. 6, let τ_1 be a fill of GAP$(\sigma_u\bullet, \bullet\mathsf{t}_1)$, i.e., a firing sequence with post-set $(\sigma_u\tau_1\mathsf{t}_1)\bullet = \{\mathsf{p}_4, \mathsf{p}_5\}$, and similarly $(\sigma_u\tau_2\mathsf{t}_2)\bullet = \{\mathsf{p}_6, \mathsf{p}_7\}$ for another t_2. Suppose that $\mathsf{t}_1 \triangleq \mathsf{t}_2$ with $\pi(\mathsf{t}_1) = \pi(\mathsf{t}_2) = \mathsf{e}$. $\pi((\sigma_u\tau_1\mathsf{t}_1)\bullet) = \pi((\sigma_u\tau_2\mathsf{t}_2)\bullet) = \{\mathsf{b}_1, \mathsf{b}_2\}$ always holds.

Next, we can prove that the branches on all $\mathsf{t} \in \mathcal{T}_i^{EC}$ must be the same, e.g., σ_1 yielded by t_1 is equivalent to σ_2 yielded by t_2, in Fig. 6. Consequently, only the branching corresponding to the local optimal t^* in $\mathcal{T}_i^{EC}$ needs to be considered. That is, if $|\tau_1| < |\tau_2|$, σ_1 can always produce the minimal recovery on the branches yielded by σ_u, and the other σ_2 can be safely pruned.

Theorem 7. *For a firing sequence σ_u, the branching on*

$$\mathsf{t}^* = \arg\min_{\mathsf{t} \in \mathcal{T}_i^{EC}} |\text{GAP}(\sigma_u\bullet, \bullet\mathsf{t})|,$$

can always generate the minimum recovery with respect to all the branches on transitions in $\mathcal{T}_i^{EC}$.

The beauty of branching equivalence classes is that they are defined independent of any firing sequence σ_u. We can identify them offline, and apply directly the local optimality.

4.2.3 Local Algorithm

Finally, we introduce the LOCAL algorithm by adapting the aforesaid BRANCH in Algorithm 2. In Line 8, the program considers each branching equivalence class $\mathcal{T}_i^{EC}$ over $\mathcal{T}_u(\mathsf{e})$. A transition $\mathsf{t} \in \mathcal{T}_i^{EC}$ with the minimum fill between σ_u and t is

found and recorded as $\tau_{\min}$. As illustrated in Line 15, for each $\mathcal{T}_i^{EC}$, only this $\sigma_u\tau_{\min}$ keeps on branching.

Algorithm 3. Local($\sigma_u, \sigma[k]$)

Input: A firing sequence σ_u and the kth event $\sigma[k]$ in σ
Output: A minimum recovered sequence after σ_u

```
1: σ_min := an infinite sequence
2: if σ[k] is null then
3:    for each sink place p ∈ P_u, π(p) = b_end do
4:       τ := GAP(σ_u•, {p})
5:       if |σ_min| > |τ| then
6:          σ_min := τ
7: else
8:    for each T_i^EC over T_u(e) do
9:       τ_min := an infinite sequence
10:      for each t ∈ T_i^EC do
11:         τ := GAP(σ_u•, •t)
12:         if τ exists and |τ_min| > |τt| then
13:            τ_min := τt
14:      if |τ_min| is not infinite then
15:         σ* := LOCAL(σ_u τ_min, σ[k+1])
16:         if |σ_min| > |τ_min σ*| then
17:            σ_min := τ_min σ*
18: return σ_min
```

Example 11 (Example 10 Continued). Given a sequence $<$AG$>$, let $\sigma_u = <t_1>$ be the current firing sequence and G be the next event. As introduced, t_5, t_6 in Fig. 4 are in the same branching equivalence class. Since the fill $<t_3t_4>$ of σ_u and t_6 is larger than that of t_4, we have $\tau_{\min} = <t_2>$ in Line 14 in Algorithm 3. Consequently, only the branching on $\sigma_u t_2 t_5$ is considered for the next event. The branching on the other t_6 in the same branching equivalence class is ignored.

4.3 Extensions on Loops

To extend for general specifications with loops, we can simply introduce a type of shadow place in process branching for loops. For example, for the specification with a loop BDEF in Fig. 1a, instead of pointing back to place b_1, the branching net will introduce a new shadow place $p_8 : b_1$, which maps to b_1 in the specification as well. Consequently, the backtracking (Algorithm 1 GAP) with loops should not only consider the place $p_1 : b_1$ before B, but also the shadow place $p_8 : b_1$ to handle the loop situation. By considering each shadow place at most once in a gap, the algorithm returns the minimum recovery. The details of extension for general specifications with loops are presented in Section 5 in the preliminary/conference version of this paper [25].

5 Efficient Algorithm

The exact algorithms with advanced indexing and pruning techniques (in Sections 3 and 4) are still costly since the size of process branching could be exponentially large. For example, a specification with 20 adjacent two-way choice structures has only 40 transitions, but its process branching has $2^{21} - 2 = 20,097,150$ transitions. An index query over some ending transition may return huge (almost all) branches for recovery consideration. This leads to a high time and space cost which cannot be avoided by exploiting branching index or local optimality.

It is possible to employ the substructure indexing techniques [26], instead of considering all the possible branches. While the existing techniques utilize the frequency information of subgraphs, such information is not available given barely the process specifications. We leave this interesting topic of determining branches to index as the future work. In this study, rather than materializing the huge size branching process (generated w.r.t. the process specification), we present a heuristic algorithm that directly performs on the process specification.

Intuitively, rather than attempting all the possible executions, the idea is to heuristically decide a branch for filling each gap. Owing to possible combinations of parallel execution tasks, it is still costly to find the minimum fill of a gap $(\sigma\bullet, \bullet e)$. Therefore, in Section 5.1, we propose a heuristic to fill the gap by iteratively assembling the shortest paths that connect the gap. Advantages of assembling shortest paths lie in two aspects: 1) It generates a fill relatively close to the minimum fill of a gap; 2) Most importantly, shortest paths between any two nodes can be pre-computed on the given process specification, and cached/reused for filling the gaps in sequences (instances of the specification). Indeed, since it may need to assemble multiple shortest paths for an (heuristic) minimum execution, we can directly pre-compute/cache/reuse the minimum execution between two nodes (Section 5.2). In particular, we show that the minimum execution based approach guarantees to terminate and obtain the optimal solution in certain cases (Section 5.3).

5.1 Algorithm with Shortest Paths

Let us first introduce the heuristic of recovering executions by assembling shortest paths between two nodes.

Similar to the exact algorithm, let σ be the currently processed sequence with two parts σ_u and σ_k, where σ_u is a firing sequence (previously recovered) without any gap, and σ_k is the remaining part of σ starting from the kth event. Suppose that there is a gap $(\sigma_u\bullet, \bullet e)$ between σ_u and the first event e of σ_k. To fill this gap, algorithm HEURISTIC(σ_u, σ_k) finds the shortest path τ by conducting an inverted BFS search starting from e, till one of the places in $\sigma_u\bullet$ is reached. By appending τ to the front of σ_k, it forms a new sequence $\sigma'_k = \tau\sigma_k$. If a gap still exists between σ_u and σ'_k, the function HEURISTIC(σ_u, σ'_k) is performed recursively. Otherwise, the first event e in σ_k is moved to the end of σ_u, and we keep on checking the gap between $\sigma_u e$ and σ_{k+1}. The algorithm ends by conducting HEURISTIC(σ, e_{end}), where e_{end} is an artificial event such that $\bullet e_{end} = \{b_{end}\}$. Initially, we call HEURISTIC(ε, σ) to start recovering.

Algorithm 4 shows the pseudo-code of heuristic recovery. Lines 4 to 9 check whether there is a gap $(\sigma_u\bullet, \bullet e)$. If yes, Lines 11 to 14 find the shortest path between a place b in $\sigma_u\bullet$ and e, insert those events to the front of σ_k, and recheck the existence of a gap by recursively conducting HEURISTIC($\sigma_u, \tau\sigma_k$). Otherwise, Lines 16 and 17 move the processing to the next event in σ_{k+1}. Once σ_k is empty, it means that the end of the input sequence has been reached, and the algorithm terminates.

Algorithm 4. Heuristic(σ_u, σ_k)

Input: A firing sequence σ_u and the remaining part σ_k of the sequence σ starting from the kth event.
Output: A recovered sequence after σ_u

```
if σ_k = ε then
    return ε
else
    e := the first event of σ_k
    τ := ε
    X := σ•
    for each place b_i ∈ X do
        if b_i ∈ •e then
            X := X − b_i
    if X ≠ ∅ then
        b := a place from X
        τ := the pre-computed shortest path fill from b to e
        σ* := HEURISTIC(σ_u, τσ_k)
        return σ*
    else
        σ* := HEURISTIC(σ_u e, σ_{k+1})
        return σ[k]σ*
```

Given a specification $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$, for each gap, we need to conduct a BFS search to find the shortest path, with complexity $O(|\mathcal{T}| + |\mathcal{P}|)$ in the worst case. Moreover, for a sequence σ, HEURISTIC will be recursively performed at most $|\sigma| \cdot |\mathcal{T}|$ times. Therefore, the worst-case time complexity of HEURISTIC is $O(|\sigma| \cdot |\mathcal{T}|(|\mathcal{T}| + |\mathcal{P}|))$.

Example 12. Consider the process specification in Fig. 7and a sequence $<$DEH$>$. There is a gap (DE•, •H) in the sequence since $\bullet\mathtt{H} = \{\mathsf{b}_7\} \not\subseteq \{\mathsf{b}_4, \mathsf{b}_5\} = \mathtt{DE}\bullet$. We find the shortest path within the gap by conducting an invert BFS search from event H, which results in $<$G$>$. However, a gap (DE•, •G) still exists after inserting $<$G$>$ to the front of H. Therefore, HEURISTIC is conducted again and a shortest path $<$F$>$ is inserted between DE and GH. The recovered sequence $<$DEFGH$>$ is exactly the minimum recovery of the sequence.

The shortest path heuristic does not always lead to the minimum recovery (due to the parallelization of flows).

Example 13. Consider a gap ($\varepsilon\bullet$, •H) w.r.t. the process specification shown in Fig. 7. One of the shortest paths from $\varepsilon\bullet$ to •H is $<$DEGH$>$. However, $<$DEGH$>$ is not a recovery since it is not a firing sequence. A new gap (DE•, •G) is detected in event sequence $<$DEGH$>$ and event F needs to be filled into this gap. Obviously, $<$DEFGH$>$ is not the minimum recovery of the gap, since another recovery $<$ABCH$>$ is shorter.

5.2 Amendment with Minimum Execution

As illustrated in Example 13, if the shortest path is not an execution, the result of HEURISTIC may deviate from the minimum recovery. We note that not only the short paths but also the minimum executions between some node pairs can be uniquely determined over the process specification. Hence, in the following, we present another heuristic method, which fills a gap with the pre-computed minimum execution (if available). Compared to shortest paths, this method manipulates the exact minimum execution and thus has higher accuracy. In particular, the time cost of directly obtaining an execution is less than that of assembling shortest paths for an execution.

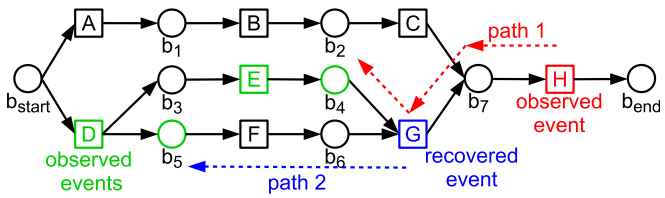


Fig. 7. Example of HEURISTIC.

5.2.1 Process Reduction

Intuitively, there are some particular structures in a specification whose minimum executions are determined independent of the preceding and succeeding contexts. For example, consider a sequential structure, i.e., all the places and transitions within the structure have at most one incoming/outgoing edge. Obviously, it has a determined execution which contains all the transitions within the structure in sequential order. Moreover, a choice structure whose branches are all sequential structures also has a determined minimum execution, which is the execution of the shortest sequential structure among all the branches. A parallel structure whose branches are all sequential structures also has a determined minimum execution as well. It could be any combination of the sequential executions in any order, since all the distinct combinations have the same length.

Following this intuition, we can reduce some structures with determined minimum execution by replacing them with a single place or transition, on which the minimum execution is recorded. It is worth noting that more structures may be able to reduce after reducing some existing structures, thus we keep doing process reduction as much as possible. Now, each path on the reduced specification may correspond to an execution, if we connect the executions on the nodes within the path. For any two nodes, if there is only one path between them, it is indeed the (minimum) execution uniquely determined.

The detailed reduction algorithm is introduced in the following. Initially, each place $\mathsf{b} \in \mathcal{P}$ is assigned with an empty execution $<>$ and each transition $\mathsf{e} \in \mathcal{T}$ is assigned with an execution $<\mathsf{e}>$. Then, we iteratively capture some particular structures which can be reduced, and replace them with single place/transitions recording the minimum executions of the reduced structure.

The process reduction rules that can be applied are:

1) *Sequential Reduction:* As illustrated in Fig. 8a, two places $\mathsf{b}_1, \mathsf{b}_2$ and one transition e can be replaced by one place b_x with $\bullet\mathsf{b}_x = \bullet\mathsf{b}_1$ and $\mathsf{b}_x\bullet = \mathsf{b}_2\bullet$, if:
   1) $|\bullet\,\mathsf{b}_1| = |\bullet\,\mathsf{b}_2| = |\bullet\,\mathsf{e}| = |\mathsf{b}_1\,\bullet| = |\mathsf{b}_2\,\bullet| = |\mathsf{e}\,\bullet| = 1$;
   2) $\bullet\mathsf{e} = \{\mathsf{b}_1\}$ and $\mathsf{e}\bullet = \{\mathsf{b}_2\}$.

Assume that b_1, e and b_2 record the executions $<\tau_1>$, $<\tau_2>$ and $<\tau_3>$, respectively. The execution recorded by b_x is the catenation of $<\tau_1>$, $<\tau_2>$ and $<\tau_3>$ in sequential order. Likewise, two transitions and a place in sequential order can also be replaced by one transition.

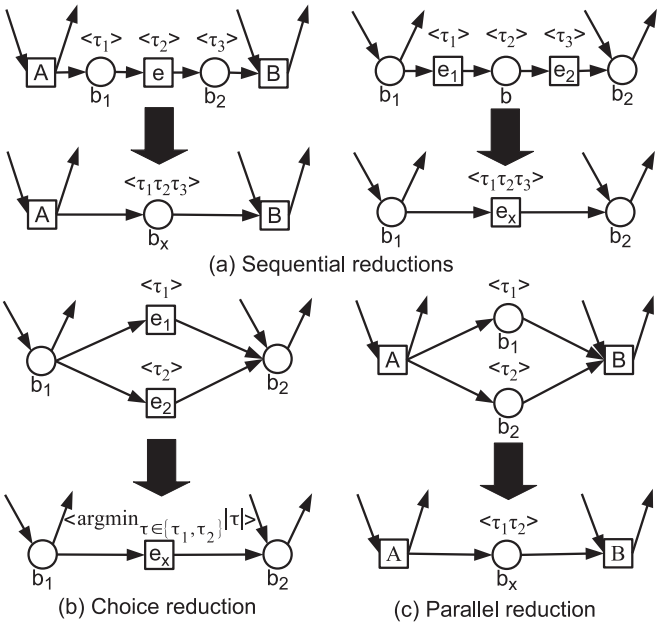


Fig. 8. Rules of process reduction.

2) *Choice Reduction:* As illustrated in Fig. 8b, two transitions e_1 and e_2 can be replaced by one transition e_x with $\bullet\mathsf{e}_x = \bullet\mathsf{e}_1$ and $\mathsf{e}_x\bullet = \mathsf{e}_1\bullet$, if:
   1) $|\bullet\,\mathsf{e}_1| = |\bullet\,\mathsf{e}_2| = |\mathsf{e}_1\,\bullet| = |\mathsf{e}_2\,\bullet| = 1$;
   2) $\bullet\mathsf{e}_1 = \bullet\mathsf{e}_2$ and $\mathsf{e}_1\bullet = \mathsf{e}_2\bullet$.

Assume that e_1 and e_2 record the executions $<\tau_1>$ and $<\tau_2>$, respectively. The execution recorded by e_x is the shorter execution between $<\tau_1>$ and $<\tau_2>$.

3) *Parallel Reduction:* As shown in Fig. 8c, two places b_1 and b_2 can be replaced by one place b_x with $\bullet\mathsf{b}_x = \bullet\mathsf{b}_1$ and $\mathsf{b}_x\bullet = \mathsf{b}_1\bullet$, if:
   1) $|\bullet\,\mathsf{b}_1| = |\bullet\,\mathsf{b}_2| = |\mathsf{b}_1\,\bullet| = |\mathsf{b}_2\,\bullet| = 1$;
   2) $\bullet\mathsf{b}_1 = \bullet\mathsf{b}_2$ and $\mathsf{b}_1\bullet = \mathsf{b}_2\bullet$.

Assume that b_1 and b_2 record the executions $<\tau_1>$ and $<\tau_2>$, respectively. The execution recorded by b_x is the catenation of $<\tau_1>$ and $<\tau_2>$.

5.2.2 Incorporating with the Algorithm

Given a process specification, we reduce it in preprocessing till no reduction is further applicable. For any two nodes b and e, if there is a unique path from b to e during the process reduction, we store the minimum execution τ recorded in the path for b and e.

In the HEURISTIC(σ_u, σ_k) recovery, before applying the shortest path between b and e in Line 12 in Algorithm 4, we first check whether there is a minimum execution precomputed from b to e. If yes, we can directly use the minimum execution τ recorded during reduction.

Example 14 (Example 13 Continued). Consider the process specification in Fig. 7. As shown in Fig. 9a, three sequential structures surrounded by blue dotted lines can be reduced and we get the specification in Fig. 9b. Then, two places b_8 and b_9 are replaced by one place b_{10} that records both the executions of b_8 and b_9, so that we have Fig. 9c. Next, we perform sequential reduction again on transitions D and G, which reduces the specification to Fig. 9d. The reduction carries on by replacing two transitions I and J with one transition K, where the transition D recording a longer execution is removed from the specification as shown in Fig. 9e.

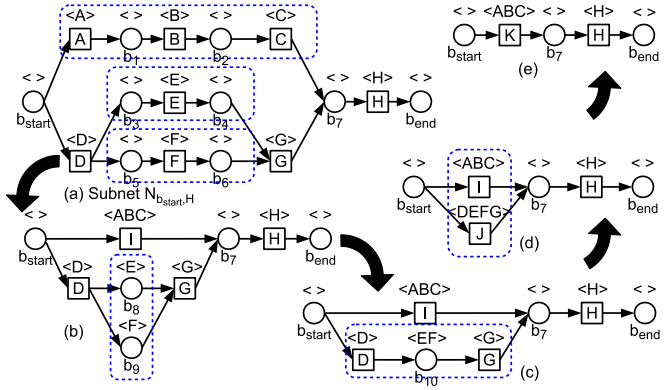


Fig. 9. Example of process reduction between b_{start} and H.

For the gap $(\varepsilon\bullet, \bullet\mathtt{H})$ in Example 13, there is a unique path from place b_{start} to transition H in Fig. 9e during the reduction. The sequence $<$ABC$>$ recorded on transition K is directly used as the minimum execution recovery between b_{start} and H.

For any stored minimum execution, there are at most $|\mathcal{T}|$ transitions. Considering the minimum executions between all places and transitions, the space cost is $O(|\mathcal{T}|^2 \cdot |\mathcal{P}|)$.

5.3 Performance Analysis

We analyze the performance of the heuristic in several typical classes of process specifications. We show that the algorithm always terminates and guarantees to return the optimal solution in some of the specifications. (See Section 6.2 for a survey on such classes in practice.)

5.3.1 State Machine

In Example 13, the shortest path recovery may lead to non-optimal result, due to the existence of parallel flows. Consider a special specification without any parallel structure, called state machine [21]. We show that the state machine is the best case for the heuristic using shortest paths.

Definition 11 (State Machine). *A state machine is a specification $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$ such that for every $e \in \mathcal{T}$, $|\bullet\,\mathsf{e}| = |\mathsf{e}\,\bullet| = 1$.*

Example 15. Fig. 10a presents an example specification that is a state machine. As illustrated, each transition e has only one input place and one output place, i.e., no parallelization of flows. Consequently, the process specification can be equivalently represented by a DAG with places as nodes and transitions as edges, as shown in Fig. 10b.

Proposition 8. *For a state machine $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$, a sequence σ over $\mathcal{N}$ and a gap $(\sigma_u\bullet, \bullet\mathsf{e})$ in σ,* HEURISTIC *with shortest paths always returns the minimum recovery of $(\sigma_u\bullet, \bullet\mathsf{e})$.*

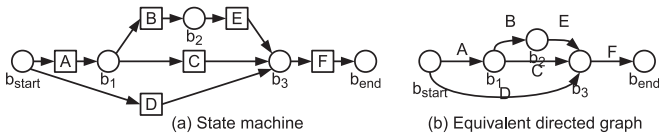


Fig. 10. Example of state machine.

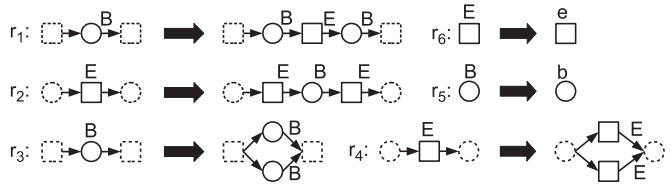


Fig. 11. Well-formed specification productions R.

5.3.2 Well-Formed Specification

We consider another class of specifications that can be reduced into only one place by applying the reduction rules in Section 5.2, known as well-formed specifications [24]. We prove that in such a case, the heuristic with minimum executions always return the optimal solution.

Definition 12 (Well-Formed Specification). *A well-formed specification grammar is a context-free grammar* $G = (V, \Sigma, R, \mathsf{B})$*, where the non-terminal variable set is* $V = \{\mathsf{B}, \mathsf{E}\}$*, the terminal variable set is* $\Sigma = \{\mathsf{b}, \mathsf{e}\}$*, the production set is* $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ *shown in Fig. 11and* B *is the start variable. The well-formed specification is the language* $L(G)$ *of* G*, where* $L(G) = \{W \in \Sigma^* | \mathsf{B} \Rightarrow^* W\}$.

Example 16. Starting from one single place, the process specification in Fig. 10 can be produced, by applying the production rules in Fig. 11. First, with rules r_1 and r_2, a path on `D,F` is produced. By applying r_4, a choice structure on `A` and `D` is generated. Similarly productions are applied to generate `B,E` and `C`. Finally, the specification in Fig. 10a is produced and is thus a well-formed specification.

Production rules r_1 and r_2 in R can be viewed as inverse operations of the sequential reduction shown in Fig. 9a. Similarly, r_3 and r_4 can be seen as inverse operations of the choice reduction and parallel reduction, respectively.

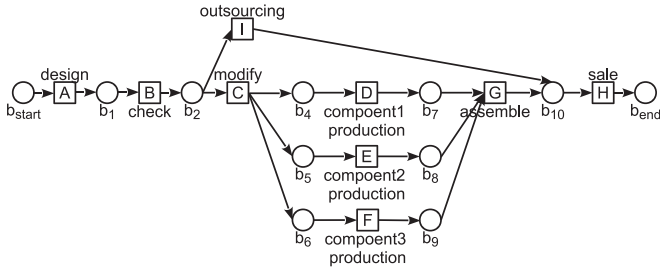
Proposition 9. *For a well-formed specification* $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$*, a sequence* σ *over* $\mathcal{N}$ *and a gap* $(\sigma_u\bullet, \bullet\mathsf{e})$ *in* σ*,* HEURISTIC *with process reduction returns the minimum recovery of* $(\sigma_u\bullet, \bullet\mathsf{e})$.

5.3.3 Post-Set Equivalence Specification

Next, we discuss a class of process specification named post-set equivalence specification, where the HEURISTIC algorithm can always return a solution but may not be the optimal one. This kind of specifications, a.k.a. the free-choice Petri net, is very common in practical specifications since it does not cause any dead-lock of execution [21].

Definition 13 (Post-Set Equivalence Specification). *Let* $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$ *be a specification whose process branching is* $\mathcal{N}_u(\mathcal{P}_u, \mathcal{T}_u, \mathcal{F}_u)$*. We denote* $\pi(\mathsf{t}\bullet) = \{\pi(\mathsf{p}) | \mathsf{p} \in \mathsf{t}\bullet\}$*. For every event* $\mathsf{e} \in \mathcal{T}$*, if all the transitions* $\mathsf{t} \in \mathcal{T}_u(\mathsf{e})$ *have the same* $\pi(\mathsf{t}\bullet)$*, we call* $\mathcal{N}$ *a post-set equivalence specification, denoted by* $\mathcal{N}_{pes}$.

Example 17. The process specification presented in Fig. 4a is a post-set equivalence specification. As illustrated in Fig. 4b, for any event (e.g., `G`) in the corresponding branching, we have the same $\pi(\mathtt{G}\bullet) = \pi(\{\mathsf{p}_5\}) = \pi(\{\mathsf{p}_6\}) = \{\mathsf{b}_{\mathsf{end}}\}$.



(a) process specification

ID	Sequence of events
1	<A, B, C, D, E, F, G, H>
2	<A, B, C, D, G, H>
3	<A, B, C, G, H>
4	<A, B, C, D, G>
5	<G, H>
6	<H>

(b) event log of process execution

Fig. 12. Example of component manufacturing.

Proposition 10. *For a post-set equivalence specification* $\mathcal{N}_{pes}$*, a sequence* σ *over* $\mathcal{N}$ *and a gap* $(\sigma_u\bullet, \bullet\mathsf{e})$ *in* σ*,* HEURISTIC *always returns a recovery of* $(\sigma_u\bullet, \bullet\mathsf{e})$*, if exists.*

6 Experiment

The experimental evaluation compares our proposed approaches with the state-of-the-art technique Alignment [4].

We employ a real data set collected from a train manufacturer. There are 149 process specifications with sizes up to 63 transitions and 79 places. The average in/out degree of transitions (parallel) is 2.61 (maximum 17). The average in/out degree of places (choice) is 2.41 (maximum 11). The event logs are extracted from the company's ERP systems. A total of 4,470 event sequences were collected from execution logs of the specifications. Among them, 3,513 sequences have at least one event missed. The minimum recovery gives the indication that at least 47.66 percent events are missing in these sequences. The maximum event sequence size is up to 1,000 events due to the loop structures in specifications (for example, task `F` in Fig. 1a).

Fig. 12 illustrates some samples for the event sequences (before recovering and after recovering, along with the corresponding specification). Specifically, the first sequence <ABCDEFGH> in Fig. 12b is a complete/correct execution of component manufacturing, while the other five sequences have missing events. The proposed method can successfully recover all the missing events in the sequences from No. 2 to 5, with recovery results <ABCDEFGH>. It is notable that the missing rate of the fifth sequence is up to 75 percent. Only the recovery over the last sequence (No. 6 with missing rate 87.5 percent) fails, which returns <ABIH>.

To verify the effectiveness of minimum recovery, we randomly remove events from the complete sequences in the data set, and apply the recovery methods to recover the removed events. Let removed be the set of all the sequences that are removed between two events, and recovered be the set of the recovered sequences between two events. We use the F-measure of precision and recall to evaluate the accuracy, given by $precision = \frac{|\text{removed} \cap \text{recovered}|}{|\text{recovered}|}$, $recall = \frac{|\text{removed} \cap \text{recovered}|}{|\text{removed}|}$,

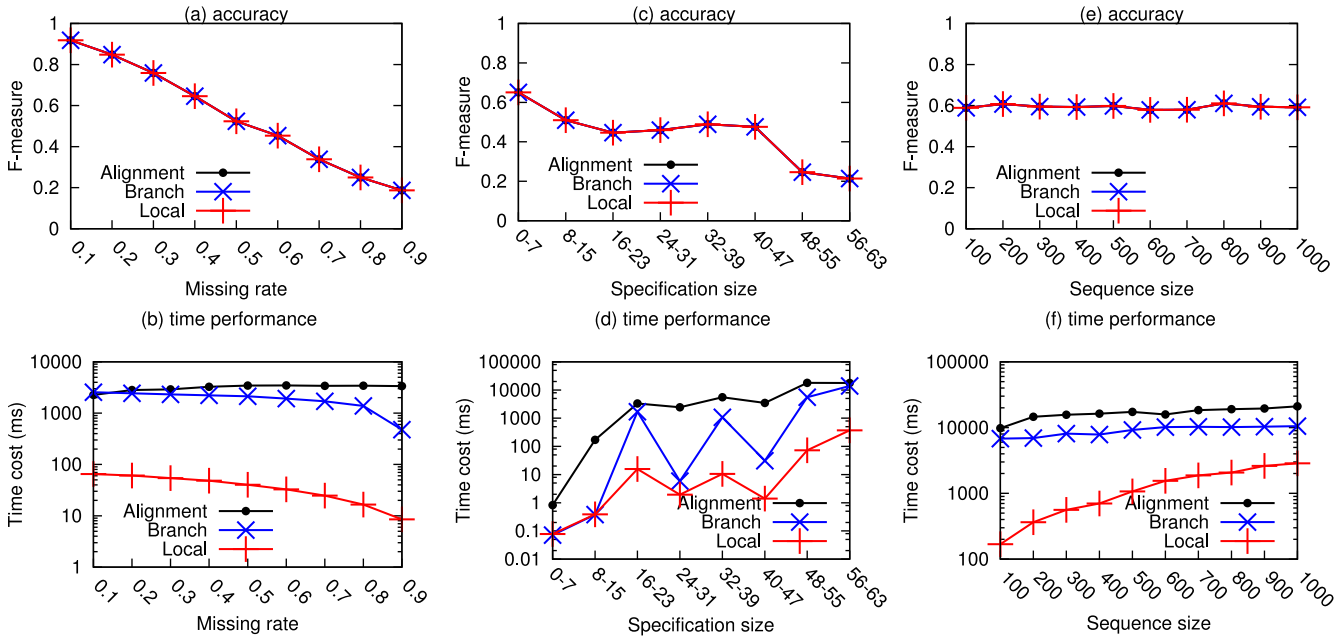


Fig. 13. Performance of exact algorithms over real dataset.

and *F-measure*$= 2 \cdot \frac{precision \cdot recall}{precision+recall}$. A larger F-measure denotes a higher recovery accuracy.

Besides accuracy and time performance, we also study the space cost of branching indexes in exact approaches, and cached shortest paths/executions for heuristic. The numbers of transitions and places are quite similar. Below, we use the size of transitions to evaluate space costs.

6.1 Experiments on Exact Algorithm

We first compare the proposed exact algorithms with Alignment [4]. Fig. 13 presents the comparison on accuracies and times costs by varying missing rates, specification sizes and sequence sizes.

In Figs. 13a and 13b, we use all the specifications with sizes varying from 3 to 63, and all the event sequences with sizes ranging from 3 to 200. Figs. 13c and 13d report the average F-score/time of all the missing rates from 0.1 to 0.9. In Figs 13e and 13f, we use the specifications with loop structures (67 of 149, about 45 percent), and the missing rate is fixed to 0.4. The same settings are applied to the other figures.

A missing rate, e.g., 0.4, denotes that 40 percent events are missing in the dataset. That is, only 60 percent events are retained in sequences. It is worth noting that the recovery relies on the (60 percent) remaining events to infer the missing events (w.r.t. process specifications). The higher the missing rate is (> 0.4), the fewer the events are retained and utilized in recovery. The corresponding F-scores are lower as well. Nevertheless, as illustrated in Fig. 13, the F-score of our proposed method is as high as the existing Alignment approach [4], while the time cost of our method (Local) is significantly lower. The results verify the major contribution of this study on improving the recovery efficiency.

For a high missing rate, e.g., 0.9, most events are missing in a sequence. It is very likely that almost any branch, in our branch-and-bound based method, could form a valid recovery and quickly prune the other branches. Moreover, our Local Optimality technique proposed in Section 4.2 further narrows down the candidate branches that have to be considered. Thereby, the algorithm runs efficiently, even over a large sequence length. On the other hand, if the missing rate is low, the algorithm needs to try more branches for seeking valid recoveries that do not conflict with existing events. Consequently, the time cost drops a bit as the increase of missing rates in Fig. 13b. Nevertheless, as illustrated in Fig. 13f, the exponential cost is still clearly observed in our Local method.

6.2 Experiments on Heuristic

Next, we evaluate the performance of the heuristic, 1) with Path which fills the gaps with shortest paths, and 2) with Reduce which uses pre-computed minimum executions by process reduction in Section 5. The Exact algorithm (Local) is compared as the baseline.

According to our survey on the real dataset, 103 (69 percent) specifications are state machines, 69 (46 percent) specifications are well-formed specifications, and all the 149 (100 percent) specifications are post-set equivalence specifications.

As illustrated in Figs. 14a, 14d, and 14g, the accuracy of (shortest) Path-based heuristic is a bit lower than that of Exact, while Reduce with the more accurate minimum executions shows better performance (closer to Exact). Indeed, the recovery distance (number of recovered events) of heuristic is very close to Exact, in Figs. 14b, 14e, and 14h. In particular, the Reduce-based heuristic shows almost the same distance as Exact, i.e., with approximation ratio almost 1. The number of recovered events by Path is much higher on large specifications than the others due to the existence of loops. On the other hand, Reduce achieves the lowest time cost in Figs. 14c, 14f, and 14i, and shows an improvement of up to two orders of magnitudes compared to Exact.

Fig. 15 reports the space cost (number of cached events) of shortest Path, minimum execution in process Reduction for heuristic, and branching net for Exact algorithms. As

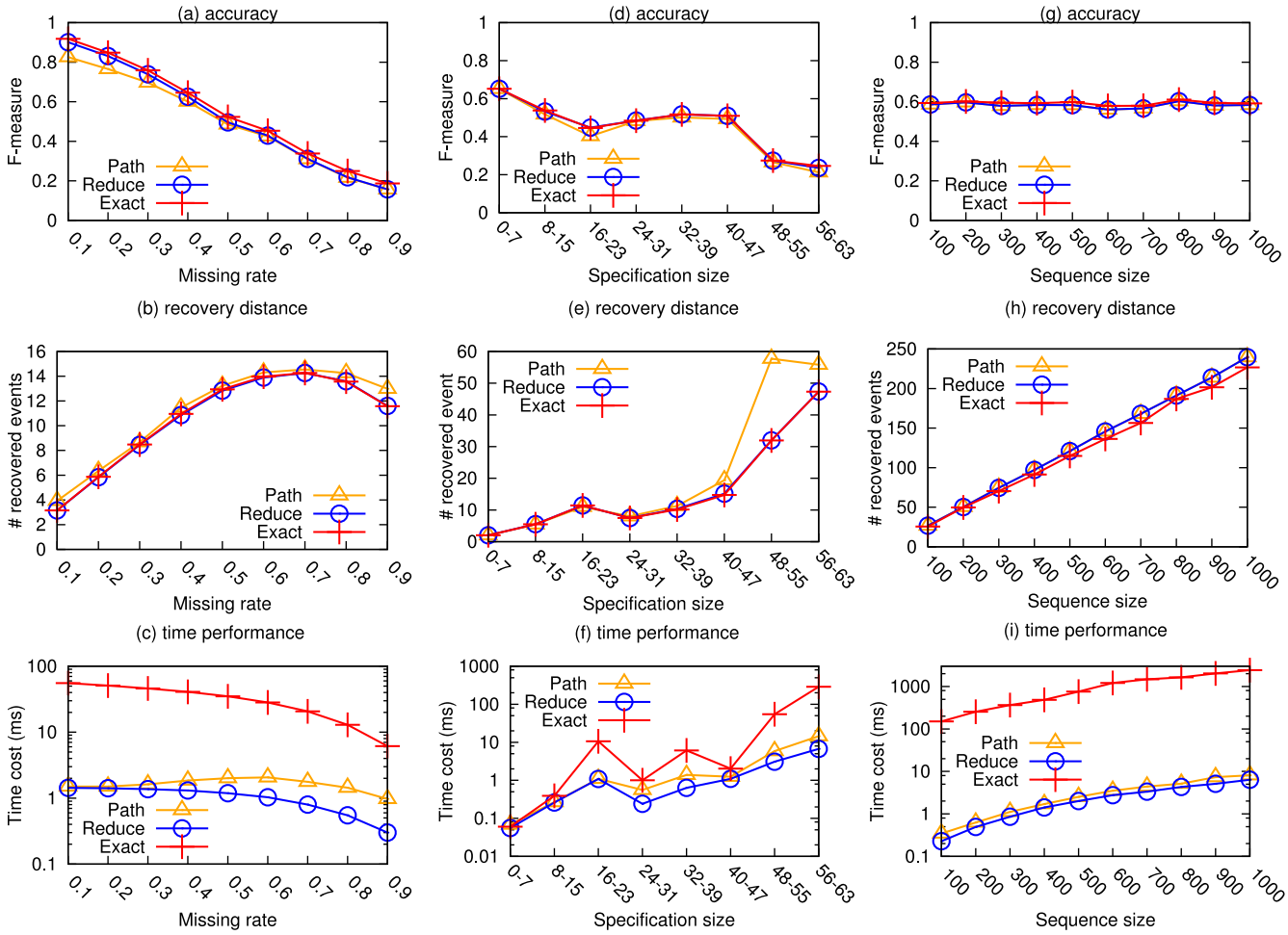


Fig. 14. Performance of heuristic over real dataset.

introduced in Section 5, the heuristic stores only the shortest paths or the minimum executions in process reduction between places and transitions in a specification, instead of caching the branching net. As shown in Fig. 15, Path/Reduce methods need only 10-30 percent space of branching (for Exact algorithm).

In short, the heuristic (with process reduction) shows significantly lower time and space costs, compared to the exact algorithm, without losing much accuracy.

6.3 Experiments on Systematic Errors

Instead of injecting random errors, we consider systematic errors, where some event is always missing after another.

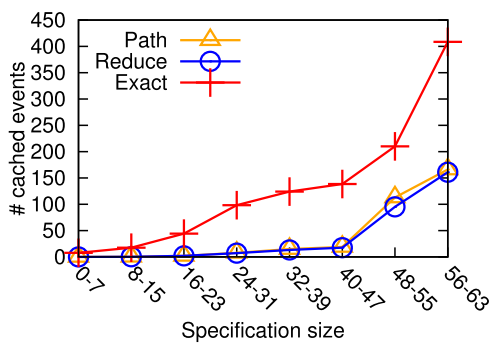


Fig. 15. Space costs over real dataset.

For instance, the Evaluation event is always missing after the Outsourcing event of passing tasks to outsourcing companies. Fig. 16 reports the results over various numbers of systematic errors. Specifications in the experiments have sizes varying from 11 to 41, and the sizes of event sequences are between 12 and 51.

The results are generally similar to those with random errors, in Figs. 13 and 14 under various missing rates. First, it is not surprising that the accuracy in Fig. 16a drops with the increase of systematic error number. All the approaches have similar accuracy, which is also observed in Figs. 13a and 14a w.r.t. random errors. Moreover, our proposed Local algorithm with both indexing and pruning shows significantly lower time cost compared to the existing Alignment. The heuristic algorithms Reduce and Path can further improve the time performance, as illustrated in Fig. 16b.

7 Related Work

Petri net are directly employed in a number of real applications. For example, YAWL, Yet Another Workflow Language based on Petri nets, is used by the European Defence Agency (EDA) for modelling and implementing personnel management processes.[2] Moreover, Petri net is a general notation for modeling workflows and has a well-developed

2. www.yawlfoundation.org/pages/impact/uptake.html

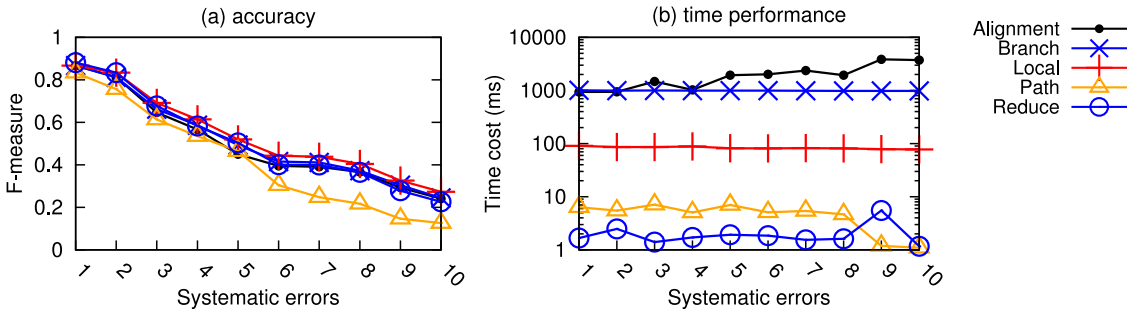


Fig. 16. Performance over various numbers of systematic errors.

mathematical theory for process analysis. Due to such a generality, other notations of industry standards, such as BPEL, BPMN and EPCs, are often translated to Petri nets, in order to perform advanced analysis and application [14]. For example, van der Aalst et al. [22] study the conformance checking for Web services, which are specified by Business Process Execution Language (BPEL, another process specification language). BPEL process definitions are translated into Petri nets and Petri net-based conformance checking techniques are applied. Therefore, in this study, we also employ Petri nets as process specifications in conformance checking, to inspect missing events.

In incomplete data management, previous studies focus on concise representation of possible recoveries [1]. Owing to the distinctness between integrity constraints and process specifications, techniques for imputing relational data [19] are not directly applicable to the event data in this study.

Event recovery works like a reverse engineering of the event summarization task. As presented in [12] and [9], the event summarization seeks concise summaries of event sequences. In contrast, the recovery performs in the other direction, i.e., filling the detailed events back to the (summarized) sequences. However, it is not suggested to perform the recovery over the event summaries. Instead, one may apply the recovery directly to the original event sequences, to fill potentially missing events, so that the event summarization over the recovered event sequences could be more precise. In this sense, event recovery is complementary to the event summarization task.

8 Conclusion

In this paper, we study the problem of finding minimum recoveries for missing events. The problem is first found to be NP-hard. To efficiently find the optimal recovery, we propose a backtracking idea to reduce the redundant sequences with respect to parallel events. A branching framework with branching index is then introduced, where each branch can apply the backtracking directly. Moreover, the local optimal method can identify groups of transitions that always share the same branching and thus only one of them needs to be computed. Finally, we propose efficient heuristic with precomputed path/execution. The experiment results demonstrate that the minimum recovery paradigm is able to effectively and efficiently retrieve the missing events.

Acknowledgments

This work is supported in part by the Tsinghua University Initiative Scientific Research Program; Tsinghua National Laboratory Special Fund for Big Data Science and Technology; National Key Technology Support Program No. 2015BAH14F02; China NSFC under Grants 61572272, 61325008, 61370055 and 61202008. A preliminary/conference version of this paper appears in [25]. Shaoxu Song is a corresponding author.

References

[1] L. Antova, C. Koch, and D. Olteanu, "10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information," in *Proc. IEEE 23rd Int. Conf. Data Eng.*, 2007, pp. 606–615.

[2] Z. Bao, S. C. Boulakia, S. B. Davidson, A. Eyal, and S. Khanna, "Differencing provenance in scientific workflows," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 808–819.

[3] P. Bohannon, M. Flaster, W. Fan, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 143–154.

[4] M. de Leoni, F. M. Maggi, and W. M. P. van der Aalst, "Aligning event logs and declarative process models for conformance checking," in *Proc. 10th Int. Conf. Business Process Manage.*, 2012, pp. 82–97.

[5] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W.-P. Hsiung, and K. S. Candan, "Runtime semantic query optimization for event stream processing," in *Proc. IEEE 24th Int. Conf. Data Eng.*, 2008, pp. 676–685.

[6] J. Engelfriet, "Branching processes of Petri nets," *Acta Inf.*, vol. 28, no. 6, pp. 575–591, 1991.

[7] D. Fahland, et al., "Instantaneous soundness checking of industrial business process models," in *Proc. 7th Int. Conf. Business Process Manage.*, 2009, pp. 278–293.

[8] T. Heinis and G. Alonso, "Efficient lineage tracking for scientific workflows," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 1007–1018.

[9] Y. Jiang, C. Perng, and T. Li, "Natural event summarization. in *Proc. 20th ACM Int. Conf. Inf. Knowl. Manage.*, 2011, pp. 765–774.

[10] B. Kang, D. Kim, and S. Kang, "Periodic performance prediction for real-time business process monitoring," *Ind. Manage. Data Syst.*, vol. 112, no. 1, pp. 4–23, 2011.

[11] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York, NY, USA: Plenum, 1972, pp. 85–103.

[12] J. Kiernan and E. Terzi, "Constructing comprehensive summaries of large event sequences," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2008, pp. 417–425.

[13] Y. Kobayashi, A. Kishimoto, and O. Watanabe, "Evaluations of hash distributed a* in optimal sequence alignment," in *Proc. 22nd Int. Joint Conf. Artificial Intell.*, 2011, pp. 584–590.

[14] N. Lohmann, E. Verbeek, and R. M. Dijkman, "Petri net transformations for business processes-a survey," in *Transactions on Petri Nets and Other Models of Concurrency II*. Berlin, Germany: Springer, 2009, pp. 46–63.

[15] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining Knowl. Discovery*, vol. 1, no. 3, pp. 259–289, 1997.

[16] K. L. McMillan, "A technique of state space search based on unfolding," *Formal Methods Syst. Des.*, vol. 6, no. 1, pp. 45–65, 1995.

[17] H. R. M. Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah, "Event correlation for process discovery from web service interaction logs," *Int. J. Very Large Data Bases*, vol. 20, no. 3, pp. 417–444, 2011.
[18] S. Song, H. Cheng, J. X. Yu, and L. Chen, "Repairing vertex labels under neighborhood constraints. *Proc. VLDB Endowment*, vol. 7, no. 11, pp. 987–998, 2014.
[19] S. Song, A. Zhang, L. Chen, and J. Wang, "Enriching data imputation with extensive similarity neighbors," *Proc. VLDB Endowment*, vol. 8, no. 11, pp. 1286–1297, 2015.
[20] P. Sun, Z. Liu, S. B. Davidson, and Y. Chen, "Detecting and resolving unsound workflow views for correct provenance analysis," in *Proc. ACM SIGMOD Int. Conf. Manage. data*, 2009, pp. 549–562.
[21] W. M. P. van der Aalst, "Workflow verification: Finding control-flow errors using Petri-net-based techniques," in *Business Process Management*. Berlin, Germany: Springer, 2000, pp. 161–183.
[22] W. M. P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek, "Conformance checking of service behavior," *ACM Trans. Internet Techn.*, vol. 8, no. 3, 2008, Art. no. 13.
[23] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.
[24] W. vanderAalst and K. vanHee, *Workflow Management: Models, Methods, and Systems*. Cambridge, MA, USA: MIT Press, 2004.
[25] J. Wang, S. Song, X. Zhu, and X. Lin, "Efficient recovery of missing events," *Proc. VLDB Endowment*, vol. 6, no. 10, pp. 841–852, 2013.
[26] X. Yan, P. S. Yu, and J. Han, "Graph indexing: A frequent structure-based approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 335–346.
[27] X. Zhu, S. Song, X. Lian, J. Wang, and L. Zou, "Matching heterogeneous event data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1211–1222.
[28] X. Zhu, S. Song, J. Wang, P. S. Yu, and J. Sun, "Matching heterogeneous events with patterns. in *Proc. IEEE 30th Int. Conf. Data Eng.*, 2014, pp. 376–387.

Jianmin Wang is a professor in the School of Software, Tsinghua University. His current research interests include unstructured data management, workflow and BPM technology, benchmark for database system, information system security, and large-scale data analytics.

Shaoxu Song is an assistant professor in the School of Software, Tsinghua University, Beijing, China. His research interests include data quality and complex event processing.

Xiaochen Zhu is currently working toward the PhD degree in the School of Software, Tsinghua University, Beijing, China. His current research interests include event data management and schema matching.

Xuemin Lin is a professor in the School of Computer Science and Engineering, University of New South Wales. He has been the head of the Database Research Group, The University of New South Wales (UNSW), Australia, since 2002. He is a fellow of the IEEE.

Jiaguang Sun is a professor with Tsinghua University. He is dedicated to teaching and R&D activities in computer graphics, computer-aided design, formal verification of software, and system architecture. He is currently the head of the School of Information Science and Technology, Tsinghua University.