Efficient Recovery of Missing Events

Jianmin Wang[$] Shaoxu Song[$] Xiaochen Zhu[$] Xuemin Lin[#]

[$]Tsinghua University, Beijing, China
{jimwang, sxsong}@tsinghua.edu.cn
zhu-xc10@mails.tsinghua.edu.cn
[#]University of New South Wales, Sydney, Australia
lxue@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-201311
May 2013

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

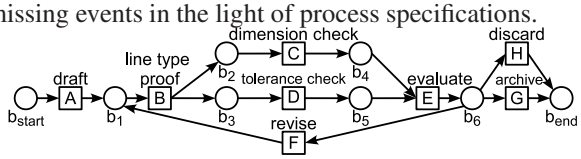
For various entering and transmission issues raised by human or system, missing events often occur in event data, which record execution logs of business processes. Without recovering these missing events, applications such as provenance analysis or complex event processing built upon event data are 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 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 is proved to be NP-hard. Nevertheless, we are able to concisely represent the space of event sequences in a branching framework. Advanced indexing and pruning techniques are developed to further improve the recovery efficiency. Our proposed efficient techniques make it possible to find top-k recoveries. The experimental results demonstrate that our minimum recovery approach achieves high accuracy, and significantly outperforms the state-of-the-art technique for up to 5 orders of magnitudes improvement in time performance.

1 Introduction

Business processes continuously generates huge volume of event data, ranging from traditional enterprise office automation systems or scientific workflows [14, 5] to recent Web services and online transactions [25]. To manage the event data, provenance analysis [27] identifies the sequence of steps leading to a data, and complex event processing [11] detects interesting event patterns from the data. While querying and mining upon event data are highlighted, 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% events are missed in the database. The missing events occur for various reasons, such as forgot to submit when manually recording event logs, suffered from system failures, or mess after collecting the events from heterogeneous execution environment.

Without addressing these missing events, the aforesaid applications and mining over event data are not reliable. Simply ignoring the missing events will yield incomplete provenance answers and lead to inaccurate event patterns. As indicated in [27], the provenance of a data item is the sequence of steps used to produce the data. Generally, it can be thought of as a graph which captures the causal dependencies between entities involved in processes, and queries of provenance as calculating transitive closures of dependencies. Owing to the incomplete event log, not only the missing events but also their corresponding prerequisites might be absent in the transitive closures of provenance. In this paper, we study the problem of recovering missing events, which can possibly provide a (set of candidates of) more complete provenance.

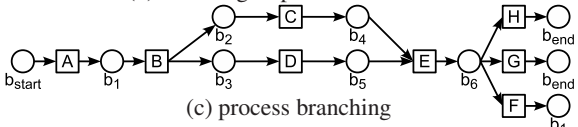
In general, it can hardly be speculated without any prior knowledge. Fortunately, most business events do not occur randomly. Instead, event data often follow certain business rules or constraints, such as process specifications [11]. Therefore, we focus on recovering missing events in the light of process specifications.



(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

Figure 1.1: Example of engineering drawing process

Example 1. *Consider a real process specification in Figure 1.1 (a) 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 denotes 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*

accomplished. Circles in the figure are choice nodes, called places*, which always appear between transitions. It indicates that only one of the flows going out a place can be executed. For instance, place* b_6 *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 $\mathsf{b}_{\mathsf{start}}$ *to the sink place* $\mathsf{b}_{\mathsf{end}}$ *exactly following the flow constraints in the specification. For example, the first sequence* `<ABCDEG>` *in Figure 1.1 (b) denotes a complete execution of engineering drawing including steps* `drafting`*,* `line type proofing`*,* `dimension checking`*,* `tolerance checking`*,* `evaluating`*,* `archiving` *from* $\mathsf{b}_{\mathsf{start}}$ *to* $\mathsf{b}_{\mathsf{end}}$*.*

In practice, 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 this provenance step. Moreover, if such data transmission problems occur frequently in the dimension checking database, an absurd event pattern without dimension checking step in engineering drawing will be mined.*

It is not surprising that multiple recoveries exist for an incomplete sequence. Previous studies on managing incomplete data are dedicated to representing all possible worlds of recoveries [1]. For event data, however, infinite sequences of events could be generated when loops exist in process specifications. For instance, to recover the third sequence `<ABCDG>` in Figure 1.1, the results could be `<ABCDEG>`, `<ABCDEFBCDEG>`, `<ABCDEFBCDEFBCDEG>`, ... Following the minimum change discipline in improving data quality [7, 21], we can also identify the optimal recovery of missing events that minimally differs from the original sequence. It is a rational assumption in improving data quality that people try to make the minimum mistakes, which is also applicable to missing events. The minimum recovery guarantees to conclude the minimum number of events that are missing, e.g., at least one event must be missing in the third sequence `<ABCDG>` in Figure 1.1. Without the minimum requirement, infinite results of possible recoveries may be returned when dealing with loops.

To find the minimum recovery, the existing alignment approach [9] studied in the business process management community enumerates all the valid sequences of events. It falls short of efficiency owing to the redundancy in all possible event sequences. For instance, to recover the sequence `<ABCEG>` in Example 1.1, 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`. As summarized below, we can explore opportunities abound both in indexing and pruning for improving the recovery efficiency.

Contributions Our main contributions in this paper are summarized as follows.

- 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 together with advanced pruning techniques are developed to accelerate recovery. The branching and pruning techniques are extended further to support loops.

Table 2.1: Frequently used notations

Symbol	Description	Place
$\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$	a net $\mathcal{N}$ with place set $\mathcal{P}$, transition set $\mathcal{T}$	Def.1
$\mathsf{b}_{\mathsf{start}}$,$\mathsf{b}_{\mathsf{end}}$	source place, sink place	Def.2
$\bullet x$	pre-set of a node x	
$x\bullet$	post-set of a node or firing sequence x	
b	place in process specification	
e	event (transition) in process specification	
σ, τ	sequence of events (transitions)	
$\delta(\mathsf{e}_i, \mathsf{e}_j)$	maximum distance of occurrence time	Def.5
t	transition in branching net	
p	place in branching net	
π	homomorphism mapping	Def.10
$\mathcal{T}_u(\mathsf{e})$	all transitions mapping to e	
$\mathsf{t}_1 \overset{\circ}{=} \mathsf{t}_2$	branching equivalence relation	Def.12
$\mathcal{T}^{EC}$	branching equivalence class	Def.13

- We employ the information of recovery size and event frequency to find a list of top-k recoveries.
- Finally, we report the extensive experimental evaluation on real and synthetic data.

The remaining of this paper are organized as follows. Section 2 illustrates the preliminaries and NP-hardness of the recovery problem. The backtracking algorithm and the verification of time constraints are presented in Section 3. We show the branching techniques in Section 4 and extend them to loops in Section 5. The top-k recoveries are introduced in Section 6. Section 7 reports the experimental evaluation. We discuss related work in Section 8 and the application of Petri nets in Section 9. Finally, Section 10 concludes this paper. *Due to the limitation of space, we present the proof details of theorems and lemmas in the full version technical report of this paper [30].*

2 Problem Statement

In this section, we present syntaxes, definitions and hardness analysis for the missing event recovery problem. Table 2.1 lists the frequently used notations.

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 set of directed arcs (flow relation).*

For any node $x \in \mathcal{P} \cup \mathcal{T}$, $\bullet x = \{y \mid (y, x) \in \mathcal{F}\}$ 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 is a Petri net $\mathcal{N}_s(\mathcal{P}_s, \mathcal{T}_s, \mathcal{F}_s)$, which has a unique source place $\mathsf{b}_{\mathsf{start}} \in \mathcal{P}_s$, $\bullet\mathsf{b}_{\mathsf{start}} = \emptyset$, and a unique sink place $\mathsf{b}_{\mathsf{end}} \in \mathcal{P}_s$, $\mathsf{b}_{\mathsf{end}}\bullet = \emptyset$. Each node $x \in \mathcal{P}_s \cup \mathcal{T}_s$ is on a path from $\mathsf{b}_{\mathsf{start}}$ to $\mathsf{b}_{\mathsf{end}}$.*

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

A sequence with missing events may not exactly obey the process specification constraint. To study whether an event sequence is logged completely, we introduce a notation of firing sequence.

Definition 3 (Firing sequence)**.** *A* firing sequence *of a process specification $\mathcal{N}_s(\mathcal{P}_s, \mathcal{T}_s, \mathcal{F}_s)$, and its post-set, 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, $\mathsf{e} \in \mathcal{T}_s$ is a transition (event), and $\bullet\mathsf{e} \subseteq \sigma\bullet$, then $\sigma\mathsf{e}$ is also a firing sequence, and $(\sigma\mathsf{e})\bullet = (\sigma\bullet) - (\bullet\mathsf{e}) + (\mathsf{e}\bullet)$.*

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

Example 2 (example 1 continued)**.** *Consider the process specification $\mathcal{N}_s$ in Figure 1.1 and a sequence $\sigma =$ <ABCDEG>. To investigate whether σ is a firing sequence with respect to $\mathcal{N}_s$, we start from the empty sequence ε with $\varepsilon\bullet = \{\mathsf{b}_{\mathsf{start}}\}$. Since the first event A has $\bullet A = \{\mathsf{b}_{\mathsf{start}}\} \subseteq \varepsilon\bullet$, the augmentation <A> is also a firing sequence with <A>$\bullet = \{\mathsf{b}_1\}$. It follows <AB>$\bullet = \{\mathsf{b}_2, \mathsf{b}_3\}$. For the next C, as $\bullet C = \{\mathsf{b}_2\} \subset$ <AB>$\bullet$, the firing sequence becomes <ABC> with post-set $\{\mathsf{b}_3, \mathsf{b}_4\}$. Similarly, we have <ABCD>$\bullet = \{\mathsf{b}_4, \mathsf{b}_5\}$ by appending D. As $\bullet E = \{\mathsf{b}_4, \mathsf{b}_5\}$ and $E\bullet = \{\mathsf{b}_6\}$, it leads to <ABCDE>$\bullet = \{\mathsf{b}_6\}$, and finally <ABCDEG>$\bullet = \{\mathsf{b}_{\mathsf{end}}\}$. Therefore, the sequence <ABCDEG> conforms to the specification.*

As mentioned, missing events may occur either in the start/middle of a sequence which prohibit it being a firing sequence, or at the end of the sequence having $\sigma\bullet \neq \{\mathsf{b}_{\mathsf{end}}\}$.

Definition 4 (Gap)**.** *Let σ 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*.*

A gap indicates that the previous firing sequence σ is successfully executed so far and it is impossible to execute the next event e further. In other words, $\sigma\mathsf{e}$ is not a firing sequence. There are some events missing between σ and e.

Example 3 (example 2 continued)**.** *Let us consider another sequence <ABCEG>. As illustrated, <ABC> is a firing sequence with <ABC>$\bullet = \{\mathsf{b}_3, \mathsf{b}_4\}$. For the next event E, however, we have $\bullet E = \{\mathsf{b}_4, \mathsf{b}_5\} \not\subseteq$ <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 <ABCDE>$\bullet = \{\mathsf{b}_6\} \neq \{\mathsf{b}_{\mathsf{end}}\}$. At least one event is missing at the end of the sequence.

It is worth noting that besides the logical controls specified by Petri nets, additional constraints could be further declared to restrict the occurrence of events.

Definition 5 (Time constraint)**.** *A time constraint of two consecutive events $\mathsf{e}_i, \mathsf{e}_j, \mathsf{e}_i \in \bullet(\bullet\mathsf{e}_j)$, denoted as $\delta(\mathsf{e}_i, \mathsf{e}_j)$, is the maximum distance of occurrence time of the events e_j and the most recent e_i that appear in any firing sequence.*

Consequently, a recovered firing sequence can be verified whether conforms to the process specification including the time constraints. For instance, consider a time constraint $\delta(\mathtt{A}, \mathtt{B})$ between consecutive events A and B in Figure 1.1 (a). Suppose that the occurrence time of event B is $ot(\mathtt{B})$ in the sequence $\sigma =$ <ABCDEG> in Example 2. For the most recent occurrence of event A in the sequence, the time constraint requires $ot(\mathtt{B}) - ot(\mathtt{A}) \leq \delta(\mathtt{A}, \mathtt{B})$.

2.2 Problem Definition

We aim at recovering the missing events in a sequence.

Definition 6. *A* recovery *of a sequence σ is also a sequence σ', such that $\sigma' \vDash \mathcal{N}$ and σ is a subsequence of σ'.*

The subsequence requirement implies that, for each i-th event $\sigma[i], 1 \leq i < |\sigma|$ in the sequence, there must exist a j-th event $\sigma'[j], j \geq i$ in the recovery having $\sigma[i] = \sigma'[j]$, and $\sigma[i+1] = \sigma'[j+k], k > 0$.

The distance between σ' and σ is given by $\Delta(\sigma', \sigma) = |\sigma'| - |\sigma|$. Following the convention of minimum changes in improving data quality [7], we can also find a recovery that minimally differs from the original sequence. As mentioned, without this minimum distance principle, infinite recoveries could be generated when loops exist in the process specification.

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

Example 4 (example 3 continued)**.** *To fill the gap ($\langle ABC\rangle\bullet, \bullet E$) in a sequence $\langle ABCEG\rangle$, 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 of $\langle ABC\rangle\bullet$. The firing sequence carries on with the new post-set $\langle ABCD\rangle\bullet = \{\mathsf{b}_4, \mathsf{b}_5\}$. It matches with $\bullet E$ and finally constitutes a recovery $\langle ABCDEG\rangle$ that conforms to the specification. A recovery $\langle ABCDEFBCDEG\rangle$ with a complete loop between $\langle ABC\rangle$ and E is not minimal. It is probably quite rare in practice that all the events in a loop are missing.*

For the sequence $\langle ABCDE\rangle$, 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 $\langle ABCDEG\rangle$ and $\langle ABCDEH\rangle$ could be returned as the minimum recovery.

Hardness analysis Owing to choices and parallelization of flows, there may have vast alternatives to enumerate in the recovery. de Leoni et. al. [9] propose an alignment based solution with exponential time complexity but fail to uncover the hardness of the problem.

As one of our major contributions in this paper, we find that generating the optimal recovery of missing events is indeed 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 $\mathcal{N}_s$ and a constant k, the problem is NP-complete to determine whether there exist a recovery σ' of σ such that $\sigma' \vDash \mathcal{N}_s$ and $\Delta(\sigma', \sigma) \leq k$.*

The NP-hardness of the problem can be proved by a reduction from the Set Cover problem. Given n sets over a universe of m elements, it is to construct a process specification for the transformation such that there is a set cover of size k if and only if an empty sequence has a recovery with size $n + k + 1$. The proof details can be found in the full version technical report of this paper [30].

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 [9] even fails to perform efficiently in this simple case.

Definition 7 (Causal net)**.** *A* causal net *is a Petri net* $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$*, such that for every* $\mathsf{b} \in \mathcal{P}, |\bullet\, \mathsf{b}| \leq 1$ *and* $|\mathsf{b}\, \bullet| \leq 1$.

According to the definition of process specifications, only the $\mathsf{b}_{\mathsf{start}}/\mathsf{b}_{\mathsf{start}}$ places can have empty pre/post-sets. The remaining places have exactly one in degree and one out degree, respectively. Consequently, a causal net can be equivalently represented as directed acyclic graph (DAG), where transitions (events) in $\mathcal{T}$ denote vertexes, and places with $|\bullet\, \mathsf{b}| = |\mathsf{b}\, \bullet| = 1$ are interpreted as edges.

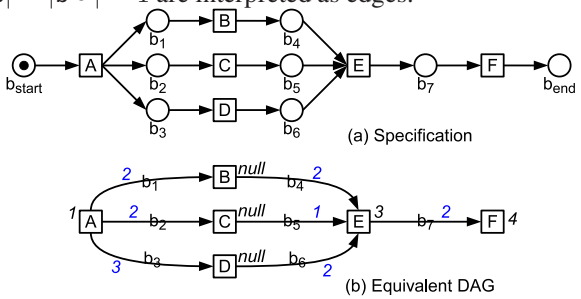


Figure 3.1: Example of causal net

Example 5. *Consider the process specification in Figure 3.1 (a), which is a causal net according to Definition 7. All the places* $\mathsf{b}_1, \ldots, \mathsf{b}_7$ *other than* $\mathsf{b}_{\mathsf{start}}/\mathsf{b}_{\mathsf{end}}$ *have* $|\bullet\, \mathsf{b}_i| = |\mathsf{b}\, \bullet_i| = 1$*, for instance,* $\bullet\mathsf{b}_1 = \{\mathtt{A}\}, \mathsf{b}_1\bullet = \{\mathtt{B}\}$*. It can be interpreted as an edge between* A *and* B *as illustrated in Figure 3.1 (b). Following the same line, the causal net can be equivalently represented as a DAG.*

Lemma 1. *Given a sequence* σ *over a causal net specification* $\mathcal{N}_s$*, the checking of conformance is equivalent to validate whether the sequence* σ *is one of the topological sorts on the DAG of* $\mathcal{N}_s$*.*

The lemma can be easily verified. First, according the definition of firing sequence, an event e can be executed only when all the places in $\bullet$e appear in the post-set of the current firing sequence, say $\sigma\bullet$. Since non-start place has $|\bullet\, \mathsf{p}| = 1$ in the causal net, each place in $\bullet$e corresponds to an event in σ. It follows $\bullet(\bullet\mathsf{e}) \subseteq \sigma$. 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 $\mathcal{T}_s$ should be performed. Therefore, each firing sequence with post-set $\{\mathsf{b}_{\mathsf{end}}\}$ corresponds to a topological sort in the DAG of $\mathcal{N}_s$.

Lemma 2. *For an incomplete sequence* σ *over a casual net* $\mathcal{N}_s$*, any topological sort* σ' *on the DAG of* $\mathcal{N}_s$ *such that* σ *is a subsequence of* σ' *is always the minimum recovery of* σ*.*

According to Lemma 1, any recovery of σ should be a topological sort. 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. Consequently, the recovery problem is indeed to find any topological sort that can contain the input sequence σ as a subsequence.

The existing alignment approach [9] 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 Figure 3.1 (a), the existing aligning approach enumerates all possible combinations of events in parallel, i.e.,* <ABCDEF>, <ABDCEF>, <ACBDEF>,

<ACDBEF>, <ADBCEF> and <ADCBEF>. According to Lemma 2, however, any topological sort should always be a minimum recovery, e.g., <ABCDEF> with the minimum distance 3 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, we propose a backtracking approach to find a topological sort as the optimal recovery. Let us first introduce how to fill a gap in a sequence. Then, a recovery can be found by checking possible gaps in one pass through the sequence.

Definition 8 (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,*
2. $\bullet\mathsf{e} \subseteq (\sigma\tau)\bullet$.

The GAP$(\sigma\bullet, \bullet\mathsf{e})$ function in Algorithm 1 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 firing 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 also 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* <A>$\bullet = \{\mathsf{b}_1, \mathsf{b}_2, \mathsf{b}_3\}$*. For each event in* $\bullet X = \{B, C, D\}$*, e.g., B, we fill the gap between the current firing sequence <A>*$\bullet$ *and* $\bullet B$*. It outputs a firing sequence <AB> with post-set <AB>*$\bullet = \{\mathsf{b}_4, \mathsf{b}_2, \mathsf{b}_3\}$*. Next, by inserting* $C \in \bullet X$*, we have <ABC>*$\bullet = \{\mathsf{b}_4, \mathsf{b}_5, \mathsf{b}_3\}$*. It follows <ABCD>*$\bullet = \{\mathsf{b}_4, \mathsf{b}_5, \mathsf{b}_6\}$*. Finally, the gap between <A> and E is filled by <BCD> such that* $\bullet E \subseteq$ <ABCD>$\bullet$.

Finally, we can 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' denotes the current firing sequence. Initially, σ' is empty, i.e., GAP$(\{\mathsf{b}_{\mathsf{start}}\}, \bullet\sigma[1])$, and at the end $\tau =$ 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 on 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 $O(|\mathcal{T}_s| + |\mathcal{P}_s|)$.

Verifying time constraints Consider a process specification where the time constraint $\delta(\mathsf{e}_i, \mathsf{e}_j)$ is given for each pair of consecutive events $\mathsf{e}_i, \mathsf{e}_j, \mathsf{e}_i \in \bullet(\bullet\mathsf{e}_j)$. Let σ' be a firing sequence where the original events are associated with a label time $ot(\mathsf{e})$ and the recovered events have $ot(\mathsf{e}) = null$. Intuitively, the time constraint validation problem aims to verifying whether any two events have the occurrence time distance no greater the (derived) time constraint.

Specifically, in the equivalent DAG derived from the causal net of σ', each arc corresponds to a weight $\delta(\mathsf{e}_i, \mathsf{e}_j)$ according to time constraints. For any connected event pair $\mathsf{e}_1, \mathsf{e}_2$, we can find a shortest weighted path with total arc weight $\hat{\delta}(\mathsf{e}_1, \mathsf{e}_2) = \sum \delta(\mathsf{e}_i, \mathsf{e}_j)$, $(\mathsf{e}_i, \mathsf{e}_j)$ in the shortest path. It is interpreted as the derived time constraint. The validation problem is indeed to verify whether $ot(\mathsf{e}_2) - ot(\mathsf{e}_1) \leq \hat{\delta}(\mathsf{e}_1, \mathsf{e}_2)$ for any two events with $ot(\mathsf{e}_1), ot(\mathsf{e}_2)$ defined, $ot(\mathsf{e}_1) \leq ot(\mathsf{e}_2)$. In light of computing the shortest path, the validation can be conducted by deriving a least bound of occurrence time $bound(\mathsf{e})$ for each e, as shown in Algorithm 2. If the actual occurrence time $ot(\mathsf{e})$ is later than $bound(\mathsf{e})$, the time constraint is violated and σ' is invalid, as shown in Line 14.

Algorithm 2 VALIDATETIME(σ)

Input: A firing sequence σ
Output: A boolean on whether σ satisfies the time constraints

```
1:  transform σ into a causal net N(P, T, F)
2:  for each e ∈ T do
3:      if ot(e) = null then
4:          bound(e) := infinity
5:      else
6:          bound(e) := ot(e)
7:  S := T
8:  while S is not empty do
9:      e_now := arg min_{e∈S} bound(e)
10:     S := S \ {e_now}
11:     for each e_next ∈ (e_now•)• do
12:         if ot(e_next) ≠ null and ot(t_next) > bound(e_now) + δ(e_now, e_next) then
13:             return false
14:         if ot(e_next) = null and bound(e_now) + δ(e_now, e_next) < bound(e_next) then
15:             bound(e_next) := bound(e_now) + δ(e_now, e_next)
16: return true
```

Example 8 (example 7 continued)**.** *Let the number on each edge in Figure 3.1 (b) denote the time constraint of two consecutive events as defined in Definition 5, e.g.,* $\delta(A, B) = 2$*. Each event in the input* σ *is associated with a label of occurrence time such as* $ot(A) = 1$*, while the recovered event has no occurrence time available, i.e.,* $ot(B) = null$*. According to the time constraint, it is required that the occurrence time of* B *should be no later than* $bound(B) = ot(A) + \delta(A, B) = 3$*. Similarly, we can derive that the largest occurrence time of* E *w.r.t.* B*, i.e.,* $ot(A)+\delta(A, B)+\delta(B, E) = 5$*. Considering all the paths from* A *to* E*, the shortest path* ACE *indicates that* E *should occur no later than* $bound(E) = ot(A) + \hat{\delta}(A, E) = ot(A) + \delta(A, C) + \delta(C, E) = 4$*. Since the observed occurrence time of* E *is* $ot(E) = 3 < bound(E)$*, the recovery* σ' *is valid w.r.t. time constraints.*

4 The Branching Framework

Now, we consider a general process specification with both choices and parallelization of flows. Different from causal net, there will be multiple choices of execution flows. A straightforward idea is to enumerate all possible flows in choice nodes by branching [12, 23], where each branch denotes a causal net without any choice. By applying the aforesaid GAP algorithm on causal net, we can find a minimal recovery for each branch, if exists. The minimum recovery can be find by traversing all the possible branches.

Branching idea Let us first introduce the idea of branching over a process specification.

Definition 9 (Occurrence net)**.** *An* occurrence net *is a Petri net $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$, such that for every $\mathsf{p} \in \mathcal{P}, |\bullet\,\mathsf{p}| \leq 1$.*

While a causal net requires both the in-degree $|\bullet\mathsf{p}| \leq 1$ and the out-degree $|\mathsf{p}\bullet| \leq 1$, an occurrence net only needs the in-degree to be $|\bullet\,\mathsf{p}| \leq 1$ for each place. In other words, there are only choice-splits but no choice-join in the process branching. To establish the relationship between process branching and specification, we introduce the following mapping.

Definition 10 (Homomorphism)**.** *A* homomorphism *from a net $\mathcal{N}_u(\mathcal{P}_u, \mathcal{T}_u, \mathcal{F}_u)$ to another net $\mathcal{N}_s(\mathcal{P}_s, \mathcal{T}_s, \mathcal{F}_s)$ is a mapping $\pi : \mathcal{P}_u \cup \mathcal{T}_u \to \mathcal{P}_s \cup \mathcal{T}_s$ such that*

1. *$\pi(\mathcal{P}_u) \subseteq \mathcal{P}_s$ and $\pi(\mathcal{T}_u) \subseteq \mathcal{T}_s$,*
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 process specification net $\mathcal{N}_s(\mathcal{P}_s, \mathcal{T}_s, \mathcal{F}_s)$ now can be unfolded, where each branch corresponds to a non-choice execution, i.e., a causal net.

Definition 11 (Process branching)**.** *A* process branching *of a specification $\mathcal{N}_s(\mathcal{P}_s, \mathcal{T}_s, \mathcal{F}_s)$ 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,*
2. *π is a homomorphism from $\mathcal{N}_u$ to $\mathcal{N}_s$, 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 $\mathsf{t}_1 = \mathsf{t}_2$.*

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, denoted as $AN(\mathsf{p}_i)$.

Proposition 1. *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 $\mathcal{P}_b \subseteq \mathcal{P}_u, \mathcal{T}_b \subseteq \mathcal{T}_u, \mathcal{F}_b \subseteq \mathcal{F}_u, \mathcal{P}_b \cup \mathcal{T}_b = AN(\mathsf{p}_i)$.*

A straightforward algorithm can apply the aforesaid topological sorting based approach to compute the recovery on each branch, and return the one with the minimum size among all the branches.

Example 9. *Consider the process specification $\mathcal{N}_s$ in Figure 4.1 (a). The corresponding process branching as illustrated in Figure 4.1 (b) is an occurrence net with only choice-split but no choice-join. 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 $\mathsf{p}_1 : \mathsf{b}_1, \mathsf{p}_1 \in \mathcal{P}_u, \mathsf{b}_1 \in \mathcal{P}_s$.*

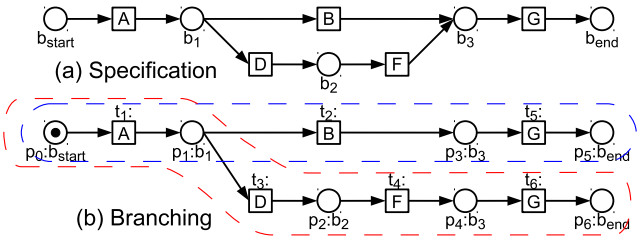


Figure 4.1: Example of branching

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 represented in dashed line in Figure 4.1 (b).*

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 Index

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.

Index on branches For any event $\mathsf{e} \in \mathcal{T}_s$, 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 11 of branching, duplicate events should not appear in a branch as well.[1]

Proposition 2. *No events appear twice in one branch.*

According to this proposition, each $\mathsf{t} \in \mathcal{T}_u(\mathsf{e})$ uniquely identifies all the distinct branches that may output a recovery containing e. As mentioned, the other branches can be safely 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 $\mathcal{N}_s$. There will be $O(d^{|\mathcal{P}_s|-1})$ branches, and the number of places $|\mathcal{P}_u|$in branching net is bounded by $O(d^{|\mathcal{P}_s|})$. According to occurrence net, each place can have at most one transition. Thereby, the number of transitions $|\mathcal{T}_u|$ in a branching net should not be greater than $|\mathcal{P}_u|$, i.e., bounded by $O(d^{|\mathcal{P}_s|})$ as well.

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, it can be transformed to a firing sequence $\sigma = \pi(\sigma_u)$ with respect to the specification.

Algorithm 3 presents the recovery with branching. Given a current firing sequence σ_u in branching net as illustrated in Figure 4.2, let $\sigma[k] : \mathsf{e}$ be the k-th (current) event in the input sequence. For each transition $\mathsf{t} \in \mathcal{T}_u(\mathsf{e})$, we call the $\textsc{gap}(\sigma_u\bullet, \bullet\mathsf{t})$ function

[1]The case of process specifications with loops needs further instruments to ensure non-duplicate events. See Section 5 for details.

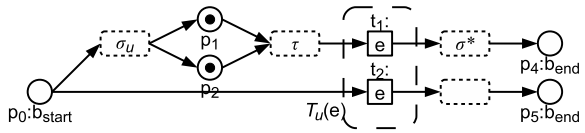


Figure 4.2: Index on branches

for causal net. It returns a fill τ between σ_u and t, if 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 transform the results $\sigma_{\min} =$ BRANCH$(\varepsilon, \sigma[1])$ to $\pi(\sigma_{\min})$.

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

Input: A firing sequence σ_u and k-th 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 visits at most $O(|\mathcal{T}_s| + |\mathcal{P}_s|)$ places and transitions, according to Proposition 2. Referring to the size analysis of branching net, the complexity of Algorithm 3 is $O((|\mathcal{T}_s| + |\mathcal{P}_s|) \cdot d^{|\mathcal{P}_s|})$, where d is the maximum out-degree of a place in $\mathcal{N}_s$.

Example 10 (example 9 continued)**.** *For a sequence $\sigma =$ <F>, we call* BRANCH$(\sigma_u, \sigma[1])$ *function, where $\sigma_u = \varepsilon, \sigma[1] =$ F. It first locates possible branches via $\mathcal{T}_u(\sigma[1]) = \{t_4\}$, i.e., only one branch containing F, as illustrated in Figure 4.1 (b). 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$(< \mathsf{t}_1\mathsf{t}_3\mathsf{t}_4 >, \sigma[2]) =< \mathsf{t}_5 >$ is returned. Finally, the minimum recovered sequence is $\sigma =< \mathsf{t}_1\mathsf{t}_3\mathsf{t}_4\mathsf{t}_5 >$ and transformed to <ADFG> as the minimum recovery.*

4.2 Pruning Branches

Although branching index significantly reduces irrelevant branches, there still have some branches that could not lead to any valid or minimum recovery. In the following, we focus on reducing the search space during the on-line computation of minimal recovery.

Path reachability pruning Intuitively, if all the places in $\sigma_u\bullet$ (e.g., $\{\mathsf{p}_1, \mathsf{p}_2\}$ in Figure 4.2) are not reachable to a candidate (e.g., t_2) in $\mathcal{T}_u(\mathsf{e})$, it is impossible to generate a fill between σ_u and t_2. That is, all the branches yielded by t_2 can be ignored. However, the branching containing t_2 is still considered by Algorithm 3.

To avoid unnecessary branching, we only need to consider those $\mathsf{t} \in \mathcal{T}_u(\mathsf{e})$ whose ancestor overlaps with the post-set of the current firing sequence $\sigma_u\bullet$. It is indeed to

decide whether there exists at least one place $\mathsf{p} \in \sigma_u\bullet$ such that p is reachable to t. By encoding on processes [14, 6], one can answer whether a place is reachable to a transition in constant time, with logarithmic encoding length.

We can modify the BRANCH($\sigma_u, \sigma[k]$) function by adding a reachability checking for each $\mathsf{t} \in \mathcal{T}_u(\sigma[k])$ in Algorithm 3. More precisely, for all places in $\sigma_u\bullet$, if none of them is reachable to t, then no fill τ exists between σ_u and t, and there is no need to execute GAP($\sigma_u\bullet, \bullet\mathsf{t}$) in Line 9.

Example 11 (example 9 continued)**.** *For a sequence $\sigma =$ <ADG>, let $\sigma_u =< \mathsf{t}_1\mathsf{t}_3 >$ be the current firing sequence with post-set $\{\mathsf{p}_2\}$ and G be the next event. According to branching index, both branches on t_5 and t_6 in $\mathcal{T}_u(G)$ will be considered as illustrated in Figure 4.1 (b). However, according to the reachability checking, all the places in $\sigma_u\bullet$, i.e., p_2, are not reachable to t_5. Therefore, the branch on t_5 leading to a sink place p_5 can be safely pruned.*

Branch and bound Next, we prune those branches that may lead to possible recoveries but cannot be the minimum one. The idea is to develop a lower bound of recovery sizes on these branches. Consequently, the branches whose lower bound of sizes is higher than the current minimum solution can be safely pruned without computing the remaining results of the recovery.

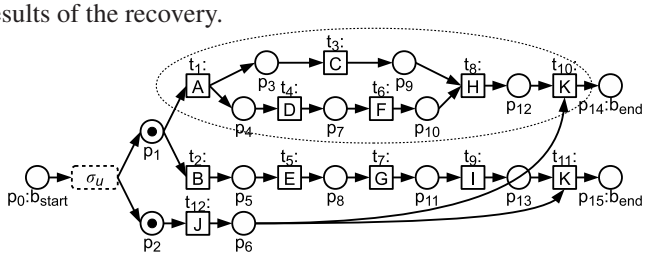


Figure 4.3: Branch and bound

The key issue is how to develop the lower bounding function. For each place in the post-set of the current firing sequence $\mathsf{p} \in \sigma_u\bullet$, it must lead to some sink places $\mathsf{p}' : \mathsf{b}_{\mathsf{end}}$ in order to form a possible recovery, e.g., from p_1 to p_{14} or p_{15} in Figure 4.3. Intuitively, the place $\mathsf{p} \in \sigma_u\bullet$ can be interpreted as a source place of a "sub-process" towards sink places p'. We can investigate the minimum branches (causal net) projected by p and p' in the branching net. More precisely, let $AN(\mathsf{p}')$ denote all the ancestors of p' and $DE(\mathsf{p})$ be all the descendants of p in the branching net. The projection of branch with respect to p and p' can be represented by $DE(\mathsf{p}) \cap AN(\mathsf{p}'), \pi(\mathsf{p}') = \mathsf{b}_{\mathsf{end}}$. The lower bound of a sequence that can lead the place p to a sink place could be defined as

$$LB(\mathsf{p}) = \min_{\mathsf{p}\in\sigma_u\bullet, \mathsf{p}'\in DE(\mathsf{p}), \pi(\mathsf{p}')=\mathsf{b}_{\mathsf{end}}} |DE(\mathsf{p}) \cap AN(\mathsf{p}') \cap \mathcal{T}_u|, \tag{4.1}$$

For example, in Figure 4.3, the "sub-process" projected by p_1 and p_{14} is illustrated in a dotted eclipse. There are 6 transitions in the projected causal net. Similarly, the number of transitions in the projected causal net of p_{15} is 5, which is smaller. According to formula (4.1), the lower bound is $LB(\mathsf{p}_1) = 5$.

It is worth noting that, other than the pruning by checking reachability on-line, the aforesaid lower bound on each place in $\mathcal{P}_u$ can be off-line computed when constructing the branching index. Considering all the places, the lower bound of a sequence that can lead the current firing sequence σ_u to a sink place is

$$LB(\sigma_u) = \max_{\mathsf{p}\in\sigma_u\bullet} LB(\mathsf{p}).$$

For the firing sequence σ_u with post-set $\{\mathtt{p}_1, \mathtt{p}_2\}$ in Figure 4.3, we have $LB(\mathtt{p}_1) = 5$ and $LB(\mathtt{p}_2) = 2$, according to formula (4.1). It follows $LB(\sigma_u) = 5$, that is, the sequence needs at least 5 more events to form a complete firing sequence ended with sink place.

Let $\sigma_{\min}$ be the minimum recovery in the currently computed recovery solutions. Then, all the branches on σ_u with a lower bound

$$LB(\sigma_u) \geq |\sigma_{\min}| - |\sigma_u| \tag{4.2}$$

can be safely pruned. This lower bound checking can be deployed at the beginning of BRANCH($\sigma_u, \sigma[k]$) in Algorithm 3.

Example 12 (example 9 continued)**.** *For an input sequence $\sigma =$ <AG>, let $\sigma_u =< \mathtt{t}_1 >$ be the current firing sequence with post-set $\{\mathtt{p}_1\}$ and G be the next event. According to branching index and reachability checking, both branches on $\mathtt{t}_5$ and $\mathtt{t}_6$ in $\mathcal{T}_u(G)$ will be considered as illustrated in Figure 4.1 (b). Suppose that $\mathtt{t}_5$ is considered in the first iteration, which returns a result $< \mathtt{t}_1\mathtt{t}_2\mathtt{t}_5 >$ of recovery <ABG>. For the second iteration on $\mathtt{t}_6$, according to formula (4.2), we can compute a lower bound $LB(\sigma_u) = 3$ of sequence that can lead σ_u to a sink place. Although it is also reachable to $\mathtt{p}_1$, the current recovery with size 3 is already smaller than the lower bound of minimum recoveries with respect to $\mathtt{t}_6$, i.e., $LB(\sigma_u) + |\sigma_u| = 4$. Consequently, the branch on $\mathtt{t}_6$ leading to a sink place $\mathtt{p}_6$ can be safely pruned.*

4.3 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 $\mathtt{t} \in \mathcal{T}_u(\sigma[k])$ that may lead to firing sequences with the same post-set, and prove that a local optimal result could be generated by only branching on one of the transitions.

Branching equivalence classes To define such groups of transitions in $\mathcal{T}_u(\sigma[k])$, we first introduce a binary relation on transitions, namely *branching equivalence relation*, denoted as $\stackrel{\circ}{=}$. Let σ_1 be a *minimum prefix firing sequence* of a transition $\mathtt{t}_1$, which only consists of transitions $AN(\mathtt{t}_1) \cap \mathcal{T}_u$ that are ancestor transitions of $\mathtt{t}_1$ and can form a new firing sequence $\sigma_1\mathtt{t}_1$. It is equivalent to the minimum fill of the gap between an empty firing sequence ε to $\mathtt{t}_1$, i.e., GAP($\varepsilon\bullet, \bullet\mathtt{t}_1$).

Definition 12 (Branching equivalence relation)**.** *For any two transitions $\mathtt{t}_1, \mathtt{t}_2 \in \mathcal{T}_u$, let σ_1 and σ_2 be the minimum prefix firing sequences of $\mathtt{t}_1$ and $\mathtt{t}_2$, respectively. Then, $\mathtt{t}_1$ and $\mathtt{t}_2$ are said* branching equivalent*, denoted by $\mathtt{t}_1 \stackrel{\circ}{=} \mathtt{t}_2$ iff:*

1. $\pi(\mathtt{t}_1) = \pi(\mathtt{t}_2)$;
2. $\pi((\sigma_1\mathtt{t}_1)\bullet) = \pi((\sigma_2\mathtt{t}_2)\bullet)$.

Obviously, relation $\stackrel{\circ}{=}$ is reflexive, symmetric and transitive. We define the branching equivalence classes as follows.

Definition 13 (Branching equivalence classes)**.** *For an event e, the transitions $\mathcal{T}_u(\mathtt{e})$ 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. $\mathsf{t}_1 \overset{\circ}{=} \mathsf{t}_2, \forall \mathsf{t}_1, \mathsf{t}_2 \in \mathcal{T}_i^{EC}$;
4. $\mathsf{t}_1 \overset{\not\circ}{=} \mathsf{t}_2, \forall \mathsf{t}_1 \in \mathcal{T}_i^{EC}, \forall \mathsf{t}_2 \in \mathcal{T}_j^{EC}$ *and* $i \neq j$.

Example 13 (example 9 continued)**.** *Consider the transitions* $\mathsf{t}_5, \mathsf{t}_6$ *in Figure 4.1 (b). 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* $\pi(\mathsf{t}_5) = \pi(\mathsf{t}_6) = G$ *and* $\pi((\sigma_1\mathsf{t}_5)\bullet) = \pi(\{\mathsf{p}_3\}) = \{\mathsf{b}_3\} = \pi(\{\mathsf{p}_4\}) = \pi((\sigma_2\mathsf{t}_6)\bullet)$*, we say* $\mathsf{t}_5, \mathsf{t}_6$ *in the same branching equivalence class having* $\mathsf{t}_5 \overset{\circ}{=} \mathsf{t}_6$.

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.

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

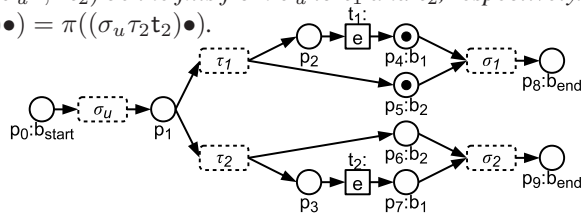


Figure 4.4: Local optimality

For example, in Figure 4.4, let τ_1 be a fill of $\text{GAP}(\sigma_u\bullet, \bullet\mathsf{t}_1)$, 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 $\mathsf{t}_1 \overset{\circ}{=} \mathsf{t}_2$ with $\pi(\mathsf{t}_1) = \pi(\mathsf{t}_2) = \mathsf{e}$. It always has $\pi((\sigma_u\tau_1\mathsf{t}_1)\bullet) = \pi((\sigma_u\tau_2\mathsf{t}_2)\bullet) = \{\mathsf{b}_1, \mathsf{b}_2\}$.

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 Figure 4.4. 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 2. *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 sequences σ_u. That is, we can pre-identify them off-line, and apply the local optimality directly.

Algorithm 4 LOCAL($\sigma_u, \sigma[k]$)

Input: A firing sequence σ_u and k-th 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
```

Local algorithm Finally, we introduce the LOCAL algorithm by adapting the aforesaid BRANCH in Algorithm 3. As shown 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.

The proposed pruning techniques are complementary to local optimality. We can still apply the branch and bound at the beginning of Algorithm 4, and employ the reachability pruning in Line 11.

Example 14 (example 13 continued)**.** *Given a sequence <AG>, let* $\sigma_u =< \mathsf{t}_1 >$ *be the current firing sequence and* G *be the next event. As introduced,* $\mathsf{t}_5, \mathsf{t}_6$ *in Figure 4.1 are in the same branching equivalence class. Since the fill* $< \mathsf{t}_3\mathsf{t}_4 >$ *of* σ_u *and* t_6 *is larger than that of* t_4*, we have* $\tau_{\min} =< \mathsf{t}_2 >$ *in Line 14 in Algorithm 4. Consequently, only the branching on* $\sigma_u\mathsf{t}_2\mathsf{t}_5$ *is considered for the next event. The branching on the other* t_6 *in the same branching equivalence class is ignored.*

5 Extensions on Loops

Finally, we move to the general specifications with loops. In practice, there are a number of realistic process specifications involving loop semantics such as "redo". Branching nets could be infinitely generated on loops. To terminate the branching, we introduce the following *portal and shadow places*.

Definition 14 (Portal/shadow place)**.** *In a process branching* $(\mathcal{N}_u, \pi)$*, for two places* p, p'*, we call* p' *a* shadow *place of* portal *place* p *if*

1. $\pi(\mathsf{p}) = \pi(\mathsf{p}')$*;*
2. $\mathsf{p} \in AN(\mathsf{p}')$ *i.e.,* p *is an ancestor of* p'.

Each portal places may have multiple shadows, denoted by a shadow set $\mathcal{P}_{shadow}(\mathsf{p})$. According to the requirement of occurrence net, there should be only one transition in the pre-set of a shadow place, say $\bullet\mathsf{p}' = \{\mathsf{t}'\}$. The process branching is extended for loops as follows.

Definition 15 (Process branching for loops)**.** *A process branching* $(\mathcal{N}_u, \pi)$ *for a process specification* $\mathcal{N}_s$ *with loops is a process branching such that for each* $\mathsf{p}' \in \mathcal{P}_u, |\mathsf{p}' \bullet| = 0, \pi(\mathsf{p}') \neq \mathsf{b}_{\mathsf{end}}$,

1. $\exists \mathsf{p} \in AN(\mathsf{p}'), \pi(\mathsf{p}) = \pi(\mathsf{p}')$,
2. $\forall \mathsf{t} \in AN(\mathsf{p}') \setminus \{\mathsf{t}'\}, \pi(\mathsf{t}) \neq \pi(\mathsf{t}')$,

where t' *is the only transition in* $\bullet\mathsf{p}'$.

The first condition in Definition 15 indicates that the branching after shadow place p' can be ignored. The second condition ensures that no events appear twice in a branch, in order to keep the branching minimal.

Example 15. *Figure 5.1 shows a process specification with two loops and its branching.* p_6 *is a shadow place of the portal place* p_1 *having* $\pi(\mathsf{p}_1) = \pi(\mathsf{p}_6) = \mathsf{b}_1$. *We use a dashed line to denote the portal/shadow connection between* p_1 *and* p_6. *According to Definition 15, the duplicate branching on shadow place* p_6 *is cut off. Consequently, only the shadow and sink places can have empty post-sets, e.g.,* $\mathsf{p}_6, \mathsf{p}_7$ *and* $\mathsf{p}_5 : \mathsf{b}_{\mathsf{end}}$. *Moreover, no duplicate event in each branch is still guaranteed. It is exactly the reason why we stop branching on event* B *after* $\mathsf{p}_6 : \mathsf{b}_1$ *which already appears in the ancestor* $\mathsf{t}_2 : B$.

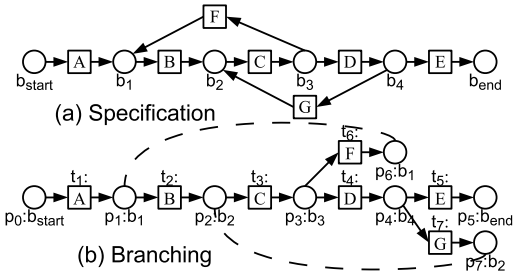


Figure 5.1: Example of branching net on loops

Algorithm extension We adapt the GAP function to support filling gaps with loops, namely GAP+ in Algorithm 5. Owing to the presence of portal/shadow places, there may exist recovered sequences with various lengths instead of topological sorting with unique size. According to the minimum fill principle, it is to find a recovered sequence with the minimum length between σ_u and t.

Lemma 4. *For a minimum fill* $\tau_{\min}$ *of a gap between* σ_u *and* t*, there does not exist any* t_1 *and* t_2 *in* $\tau_{\min}$ *such that* $\mathsf{t}_1 \bullet \cap \mathsf{t}_2 \bullet \neq \emptyset$.

The lemma states that during the generation of a minimum fill of a gap, any place can only appear at most once. Obviously, if a same place appears twice, there must exist a complete loop in the fill. According to the minimum requirement, such loop should be eliminated. We use a counter lc to record the the times of portal places p_i being looped, initially $lc(\mathsf{p}_i) = 0$.

The backtracking with loops should not only consider the places $\mathsf{p}_i \in \bullet\mathsf{t}$, but also the possible shadow places of p_i. Similar to the original GAP, Line 3 consider all places $\mathsf{p}_i \notin \sigma_u \bullet$. If p_i is a portal places and has loop counter $lc(\mathsf{p}_i) < k$, all the corresponding shadow places should be considered during backtracking, denoted as $\mathcal{P}_i := \{\mathsf{p}_i\} \cup \mathcal{P}_{shadow}(\mathsf{p}_i)$. Here, k denotes the maximum times that a portal place could be looped (k=1 when finding the minimum fill, i.e., at most once). Let Λ denote all the combinations on possible shadow places of each portal place. Each $X' \in \Lambda$ leads to a possible fill τ between σ_u and t. Following the same line of GAP function, the fill can be recursively computed by considering all the prerequisites of t with respect to the places in X' in Line 12. Finally, among all possible combinations of pre-sets, a minimum fill $\tau_{\min}$ is return.

Algorithm 5 GAP+$(\sigma_u\bullet, \bullet\mathsf{t}, lc)$

Input: A current firing sequence σ_u, a transition t and a counter lc recording the times each portal place being looped

Output: A fill of the gap between σ_u and t

1: τ_{min} := an infinite sequence
2: **for** each place $\mathsf{p}_i \in \bullet\mathsf{t}$ **do**
3: **if** $\mathsf{p}_i \notin \sigma_u\bullet$ **or** p_i is a portal place with $lc(\mathsf{p}_i) + 1 < k$ **then**
4: $\mathcal{P}_p^i$:=$\{\mathsf{p}_i\}$
5: **if** p_i is a portal place with $lc(\mathsf{p}_i) < k$ **then**
6: $\mathcal{P}_p^i$:=$\mathcal{P}_p^i \cup \mathcal{P}_{shadow}(\mathsf{p}_i)$
7: $lc(\mathsf{p}_i) := lc(\mathsf{p}_i) + 1$
8: $\Lambda := \Lambda \times \mathcal{P}_p^i$
9: **for** each $X' \in \Lambda$ **do**
10: $\tau := \varepsilon$
11: **for** each transition $\mathsf{t}_i \in \bullet X'$ **do**
12: $\tau' :=$ GAP+$((\sigma_u\tau)\bullet, \bullet\mathsf{t}_i, lc)$
13: $\tau := \tau\tau'\mathsf{t}_i$
14: **if** $|\tau| < |\tau_{min}|$ **then**
15: $\tau_{min} := \tau$
16: **return** τ_{min}

Once the minimum fill is computed, we can seamlessly apply the BRANCH algorithm to compute the minimum recovery, by calling $\tau :=$ GAP+$(\sigma_u\bullet, \{\mathsf{p}\}, lc)$ in Line 4 and $\tau :=$ GAP+$(\sigma_u\bullet, \bullet\mathsf{t}, lc)$ in Line 9 of Algorithm 3. During backtracking, we need to extend from portal places to shadow places, which may obstacle the pruning techniques performing. The reachability checking can only be effective on those transitions t, whose ancestors $AN(\mathsf{t})$ are not portal places. The branch and bound approach is directly applicable, as shadow places always lead to the execution of some transitions more than once and will not affect the lower bound by counting transitions (at most once) in the branching net.

For the minimum fill, GAP+ function visits a place or transition of the specification at most once with complexity $O(|\mathcal{T}_s| + |\mathcal{P}_s|)$. For each event, there are $O(d^{|\mathcal{P}_s|-1})$ branches to consider, where d is the maximum out degree of a place in $\mathcal{N}_s$. Considering branches over all events in the input sequence σ, the complexity of the branching recovery with loops is $O(|\sigma| \cdot (|\mathcal{T}_s| + |\mathcal{P}_s|) \cdot d^{|\mathcal{P}_s|-1})$.

Example 16 (example 15 continued)**.** *Consider a sequence* <ABCB>*. A gap will be detected when processing the last event* $\sigma[4]$: B*. Let* $\sigma_u =< \mathsf{t}_1\mathsf{t}_2\mathsf{t}_3 >$ *be the current sequence. It is to fill the gap between* σ_u *and* t_2 *with* $\pi(\mathsf{t}_2) =$ B*. By calling* GAP+$(\sigma_u\bullet, \bullet\mathsf{t}_2, lc)$*, a portal place* $\mathsf{p}_1 \in \bullet\mathsf{t}_2$ *is considered. Since this portal place is not visited according to* $lc(\mathsf{p}_1) = 0$ *in the initialization, the program backtracks both the portal place* p_1 *and the corresponding shadow place* p_6*. For the first case, by recursively calling* GAP+$(\sigma_u\bullet, \bullet\mathsf{t}_1, lc)$*, no valid fill could be generated. In the second case, considering* $\mathsf{t}_6 \in \bullet\mathsf{p}_6$*,* GAP+$(\sigma_u\bullet, \bullet\mathsf{t}_6, lc)$ *returns a fill of empty sequence, i.e.,* $\sigma_u\mathsf{t}_6$ *is a firing sequence with no gap. Consequently,* $< \mathsf{t}_6 >$ *is a fill returned in the second case and is also the minimum fill among all backtracking alternatives. The current firing sequence becomes* $\sigma_u =< \mathsf{t}_1\mathsf{t}_2\mathsf{t}_3\mathsf{t}_6\mathsf{t}_2 >$*. The* BRANCH *algorithm finally generates a* $\sigma_{\min} =< \mathsf{t}_1\mathsf{t}_2\mathsf{t}_3\mathsf{t}_6\mathsf{t}_2\mathsf{t}_3\mathsf{t}_4\mathsf{t}_5 >$*, which maps to a minimum recovery* $\pi(\sigma_{\min}) =$ <ABCFBCDE>*.*

6 On Top-k Recoveries

It is notable that we cannot always automatically make correct recovery of missing events by only using specification constraints. The rationale is that multiple valid alternatives exist due to the choice structure of process specifications. For example, consider the specification in Figure 5.1 (a) with loops. A sequence σ<ABCDE> contains the semantics of a blank action between event C and D. More precisely, it may be the case that some events FBC do occur and are missed between C and D, i.e., <ABCFBCDE>. Or it is also possible that the log data is correct that such events FBC do not occur. The automatic recovery approach cannot tell which case is the real occurred one. Further knowledge is necessary for recovering such cases. A method concerned is to return a list of k candidate recoveries instead of one. For instance, the recovery candidates for the above example σ can be <ABCDE>, <ABCFBCDE>, ..., ranked by sizes.

Moreover, a recovery with events, which appear more frequently in the event log, may have a larger chance of being the correct recovery. For each event $\mathsf{e} \in \mathcal{T}_s$, let $f(\mathsf{e})$ denote the frequency of event e occurring in the event log. We define the frequency score of a recovered result σ' as $s(\sigma') = \sum_{k=1}^{n} f(\sigma'[k])$, where $n = |\sigma'|$ denotes the length of the recovered firing sequence. Consequently, for any recoveries σ_1, σ_2 with the same recovery distances $\Delta(\sigma_1, \sigma), \Delta(\sigma_2, \sigma)$ to the original input σ, i.e., having $|\sigma_1| = |\sigma_2|$, we can further rank the results according to their frequency scores $s(\sigma_1), s(\sigma_2)$.

Finally, instead of only one answer, the results consist of a list of k candidate recoveries $\sigma_1, \ldots, \sigma_k$, such that 1) for any σ_i, σ_{i+1}, having $|\sigma_i| \leq |\sigma_{i+1}|$; and 2) if$|\sigma_i| = |\sigma_{i+1}|$, then $s(\sigma_i) \geq s(\sigma_{i+1})$. That is, a recovery with smaller distance is preferred, and if some candidate recoveries share the same distances, the one with high frequency events is favored in ranking. In Algorithm 3, let $\sigma_{\min}$ denote the k-th minimum recoveries that have been found. The BRANCH algorithm directly returns top-k answers.

7 Experiment

In this section, we report the experimental evaluation by comparing our proposed branching approaches with the state-of-the-art technique Alignment [9]. The programs are implemented in Java and all the experiments were performed on a computer with Intel(R) Core(TM) i7-2600 3.40GHz CPU and 8 GB memory.

Data set We employ a real data set collected from a train manufacturer[1]. There are 149 process specifications considered 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). As illustrated in Table 7.1, 25 specifications are causal nets without any choice of flows; 57 specifications contain choices but no loops; and the remaining 67 specifications involve loops. The event logs are extracted from the company's ERP systems. A total of 4470 event sequences were collected from execution logs of the specifications. Among them, 3513 sequences have at least one event missed. The minimum recovery gives the indication that at least 47.66% events are missing in these sequences.

To build branching index, we need to materialize the branching nets of process specifications. Figure 7.1 reports both the transition and place sizes of branching nets.

[1] www.tangche.com

Table 7.1: Statistics on process specifications

Type	#	%
causal net	25	16.78
with choices (no loop)	57	38.25
with loops	67	44.97

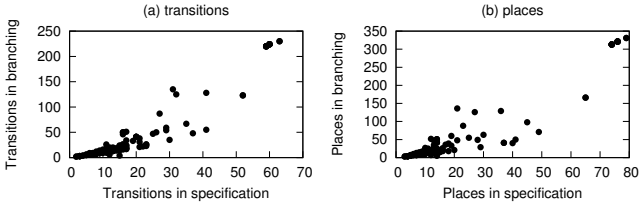


Figure 7.1: Statistics on branching nets

As illustrated, the size of branching net is about 3 times larger than the original specification net. The sizes of transitions and places are quite similar. Below, we use the size of transitions to distinguish process specifications.

In order to evaluate the scalability of our algorithms and the performance over different workflow patterns (i.e., typical structures such as sequential, choice and parallel in specifications), we generate four larger synthetic data sets, including sequential, parallel, choice and complex. The Sequential testbed contains specifications that only have sequential structure, i.e., the in/out degree of each place/transition is at most 1. The Parallel testbed consists of specifications containing only parallel flows, while the specifications in Choice testbed only have structures of flow choices. Finally, the Complex testbed contains specifications each of which consists of the same amount of parallel, choice and sequential structures. We generate event sequences on four synthetic testbeds by using a log generator BeehiveZ [16]. Each generated sequence have 20% random missing events. In order to obtain event sequences with larger sizes, we employ a loop structure from the end to the beginning of each specification.

Criteria Besides time performance, we also verify the effectiveness of minimum recovery. Specifically, 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 sequences that are removed between two events, and recovered be the set of recovered sequences between two events. We use the F-measure of precision and recall to evaluate the accuracy, given by $\textit{precision} = \frac{|\mathsf{removed} \cap \mathsf{recovered}|}{|\mathsf{recovered}|}$, $\textit{recall} = \frac{|\mathsf{removed} \cap \mathsf{recovered}|}{|\mathsf{removed}|}$, and $\textit{F-measure}= 2 \cdot \frac{precision \cdot recall}{precision + recall}$. A larger F-measure denotes a higher recovery accuracy.

Exp. on specifications of causal nets We first study the comparison on specifications of causal nets. Since our indexing and pruning techniques work on different branches and have no effect on causal net (as one branch), the comparison focuses on Alignment and our Branch with the backtracking technique. We report accuracy and time performance on various missing rates of events in Figures 7.2 (a) and (b). The accuracy results of two approaches are similar, while our Branch shows about 5 orders of magnitudes improvement in time costs. According to our analysis in Section 4, the complexity of branching recovery relates to the size of branching net (i.e., equivalent to specification of causal net). We also observe the performance under various specification sizes in Figures 7.2 (c) and (d). As illustrated, time costs of our branching approach increase slowly as the size of specifications, while time costs of Alignment are high and unstable (affected by the structure of specifications).

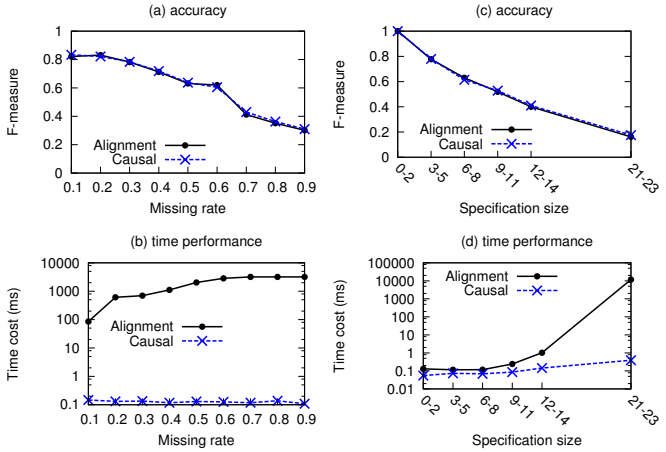


Figure 7.2: Performance on causal net specifications

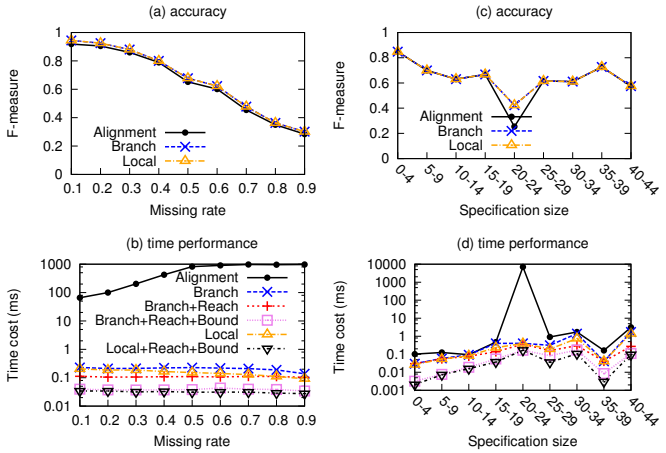


Figure 7.3: Performance on specifications with choices

Exp. on specifications with choices The second experiment in Figures 7.3 (a) and (b) reports the average accuracies and time costs on all 57 specifications with choices by varying missing rates of events. We also observe the results on different specification sizes in Figures 7.3 (c) and (d). The results of specifications with sizes 20-24 in Figures 7.3 (c) and (d) are interesting. We find that these specifications contain a large number of parallel flows (i.e., the degree of transitions is large). Thereby, our Branch approach can show significant lower time cost (as the case of causal net) than Alignment. The corresponding accuracies are lower partially because of the criteria, which reports a success only when the recovered sequence exactly matches with the randomly removed one between two events. As discussed, a swapping of two parallel events may still be correct but identified as a fault in the evaluation criteria.

For the remaining specifications in Figures 7.3 (c) and (d), the improvement of branching is not significant, since these specifications contain much less parallel events. Nevertheless, our Branch approach is at least no worse than Alignment in all tests, and show significant improvement in average as presented in Figures 7.3 (a) and (b). Moreover, our advanced approach Local+Reach+Bound can further improve the time performance (indeed the best one) in Figure 7.3 (d).

Exp. on specifications with loops For the general case of specifications with loops, the performance relies on not only the specifications but also the sizes of sequences.

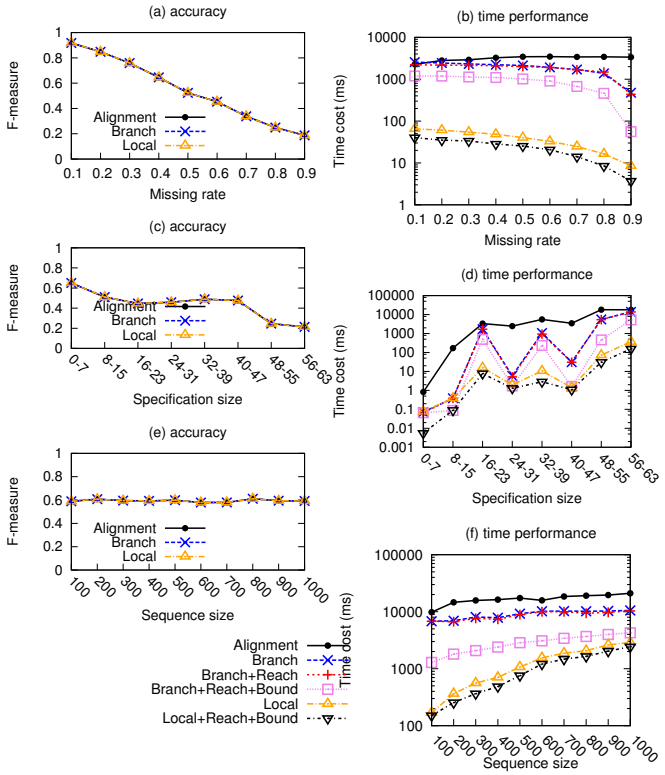


Figure 7.4: Performance on general specifications with loops

Figure 7.4 presents the comparison on accuracies and times costs by varying missing rates, specification sizes and sequence sizes.

First, the reachability checking does not function as good as it in Figure 7.3 (specifications without loops), since it works only when there is no portal place between two nodes. The branch and bound technique can show improvement together with both the original Branch and the revised Local approach. Our advanced approach Local+Reach+Bound can always achieve the lowest time cost and show several orders of magnitudes improvement compared with Alignment.

Again the results on different specifications may be unstable in Figures 7.4 (c) and (d), as the sizes of branching depends on the structure of specifications (e.g., degree of places). The longer a sequence is, more loops it may contain. Therefore, time costs increase as sequence sizes in Figure 7.4 (f). It is also interesting that the time costs drop as the increase of missing rates in Figure 7.4 (b). The rationale is that each backtracking may search possible sequences that cannot lead to any valid fill. When most events are missing, the number of backtracking operations decreases and consequently the searching on invalid sequences reduces. As mentioned, reachability checking, which is employed to avoid searching on invalid sequences, does not function well in loops. Thereby, the time cost decreasing trends are more significant in Figure 7.4 (b) than that of Figure 7.3 (b) without loops.

Exp. on synthetic data sets To evaluate the impact of different workflow patterns, we employ four synthetic testbeds, including Sequential, Parallel, Choice and Complex cases. For each testbed, 10 specifications are generated with sizes ranging from

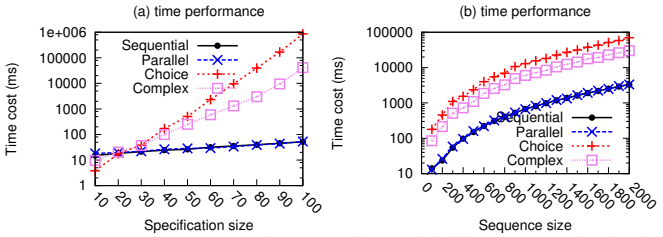


Figure 7.5: Scalability on synthetic data

10 transitions to 100 transitions.[2] We study the performance of our most advanced approach Local+Reach+Bound in Figure 7.5 (a). The time costs on Sequence and Parallel testbeds increase slowly with the rise of specification sizes, which is similar to the results in Figure 7.2 (d). However, the time cost of Choice testbed grows heavily, since the size of branching index could be very large with the growth of choice structures. The time cost of Complex data set also grows significantly, but not as fast as Choice because the Complex specifications contain less choice structures with some other sequential and parallel cases.

We also evaluate the scalability over larger sizes of event sequences, up to 2000 events,[3] using the specification with 40 transitions in each testbed. As shown in Figure 7.5 (b), the time costs show a trend of increase similar to the results on real data, e.g., in Figure 7.4 (d). Again, the Choice testbed has the highest time cost owing to the large size of branching index. The Sequential and Parallel testbeds as well as the Complex case with some sequential and parallel structures show lower time costs.

An interesting result is that Sequential and Parallel testbeds have almost the same time costs in both Figures 7.5 (a) and (b). The rationale is that specifications with only sequential or parallel structures are exactly causal nets. The recovery approach in Section 3 is directly performed on both cases by backtracking causal nets. Thereby, the time costs on Sequential and Parallel are quite similar.

Exp. on top-k recoveries Finally, we report the results on top-k recoveries. Figure 7.6 presents the highest F-measure of all the k results. First, as illustrated, the frequency information can improve the ranking of recoveries, i.e., top-1 recoveries (with frequency consideration) show higher accuracy than that of top-1 results without frequency. Moreover, the top-2 and top-3 recoveries can further improve the accuracy compared with the top-1. However, by keeping on enlarging k, e.g., top-10, it can hardly increase the accuracy further. Therefore, it is sufficient to explore the top-3 recoveries of missing events. The corresponding time costs of top-3 tests are slightly higher than that of top-1.

8 Related Work

In event data management, a series of interesting tasks have been raised. For example, provenance queries [27] answer the sequence of steps leading to a queried data. Complex event processing [11] detects interesting event patterns from the event database.

[2]Real process specifications, however, often have sizes bounded by about 60, according to the recent survey [29]. Indeed, referring to the process modeling guidelines [24], process specifications should be decomposed if they have more than 50 elements, so that they are easier to read and understand.

[3]According to our statistics on real data, the execution sequences often have sizes within 1000. A recent study on detecting anomaly in event logs [10] employs synthetic data with maximum sequence size 78 (21.19 on average).

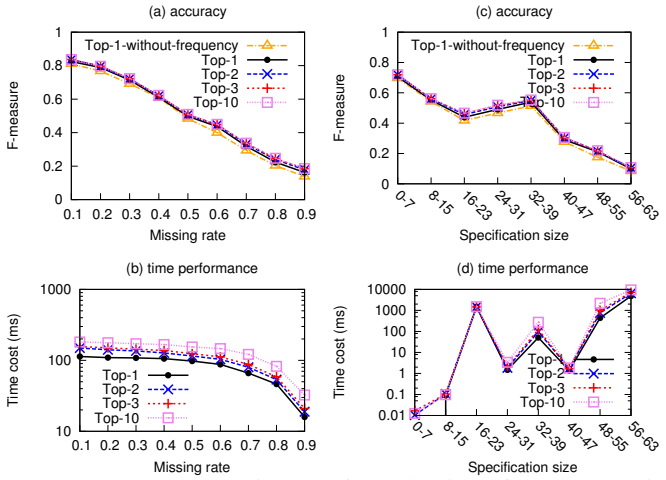


Figure 7.6: Evaluation of top-k extension

Nezhad et al. study the correlation of events for mining process specifications [25]. All these applications rely on a source of accurate and complete event data.

In incomplete data management, previous studies focus on concise representation of possible recoveries. To capture incompleteness and non-determinism in design, planning and scheduling specifications, data models such as AND/OR trees [15] are proposed by facilitating factorized representations. Antova et al. [1] propose world-set decompositions for finite sets of worlds, as well as relational algebra queries on world-sets [2]. However, these two techniques, AND/OR trees [15] and world-set decompositions [1], dedicated to representing finite sets of worlds are not directly applicable to event data studied in this paper, where sequences could be infinite due to the existence of loops. Thereby, we employ process branching techniques [12] which can represent the case of loops.

The minimal recovery of missing events is also studied as optimal sequence alignment [18], where A^* algorithm is employed [9]. The basic idea [9] is to enumerate all the valid combinations of events as possible sequences, and apply the A^* algorithm to search the one with the minimum cost. As mentioned in the introduction, the alignment approach considers a search space involving redundant sequences with respect to parallel events. Our proposed approaches can successfully avoid such inefficient scenario and show significantly lower time cost in the experiments.

9 Utility of Petri Nets

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.[1] In bioinformatics, Will and Heiner [31] report a comprehensive survey of Petri nets in biology, chemistry, and medicine. The research group lead by Monika Heiner has conducted a series of studies on applying Petri nets in bioinformatics, e.g., application of Petri net for modelling and validation of the sucrose breakdown pathway in potato tuber [19]. Their representative results include the Snoopy system: a unifying Petri net framework to investigate biomolecular networks [26], STEPP: a tool for Petri net-

[1] http://www.yawlfoundation.org/pages/impact/uptake.html

based path analysis in biochemical networks [20], and so on. Moreover, Gambin et al. [13] introduce Nested Relational Calculus (NRC) and Petri nets as a formal model for expressing bioinformatics workows.

Moreover, Petri net is a general notation for modeling workflows and has a well-developed 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 [22]. For example, van der Aalst et al. [28] 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. The translation guarantees that the original BPEL definition and the translated Petri net specification have exactly the same space of all possible sequences. Moreover, a sequence of events conforms to the BPEL definition, if and only if the sequence conforms to the corresponding specification translated into Petri net. Following this principle, a minimum recovery of a sequence w.r.t. Petri net must be a minimum recovery over the corresponding BPEL definition as well. Therefore, our proposed techniques can be applied to processes specified by BPEL. Similarly, our techniques may also be applicable to BPMN and EPC.

Unfortunately, our proposed techniques are not directly applicable to simpler workflow models defined by context-free grammars, such as SEAM [4]. In particular, the branching approach relies on the unfolding of Petri net, which is context-sensitive.

10 Conclusion and Discussion

In this paper, we study the problem of finding minimum recoveries for missing events. The problem is first proved 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 is then introduced, where each branch can apply the backtracking directly. We construct a branching index, and develop reachability checking and lower bounds of recovery distances to further accelerate the computation. 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. The proposed techniques are then adapted to support processes with loops. Finally, we can naturally extend the approach to answer top-k recoveries. The experiment results demonstrate that the minimum recovery paradigm is able to effectively and efficiently retrieve the missing events.

As with the other automatic recovery techniques over relational data, our automatic recovery techniques also cannot guarantee to always return the true results without involving the executor of each individual event. In order to improve the accuracy of recovery, learning ranking functions for top-k recoveries is a possible future work.

Moreover, besides suggesting a top-k list of minimum recoveries as studied in this paper, it is also interesting to consider other recovery scheme. For example, 1) relying on the knowledge of constraint and data, it is to return a sub-sequence of "core" events that always appear in all the possible recoveries, as known as the consistent query answering problem [3] in data cleaning. Or 2) we can rely on the knowledge of people. A user may ask why the two occurrences of event `F` (conducted by the user) do not appear in σ<ABCDE>. It is also studies as the why-not problem [8].

Bibliography

[1] Lyublena Antova, Christoph Koch, and Dan Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, pages 606–615, 2007.

[2] Lyublena Antova, Christoph Koch, and Dan Olteanu. From complete to incomplete information and back. In *SIGMOD Conference*, pages 713–724, 2007.

[3] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[4] Akhilesh Bajaj and Sudha Ram. Seam: A state-entity-activity-model for a well-defined workflow development methodology. *IEEE Trans. Knowl. Data Eng.*, 14(2):415–431, 2002.

[5] Zhuowei Bao, Sarah Cohen Boulakia, Susan B. Davidson, Anat Eyal, and Sanjeev Khanna. Differencing provenance in scientific workflows. In *ICDE*, pages 808–819, 2009.

[6] Zhuowei Bao, Susan B. Davidson, Sanjeev Khanna, and Sudeepa Roy. An optimal labeling scheme for workflow provenance using skeleton labels. In *SIGMOD Conference*, pages 711–722, 2010.

[7] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD Conference*, pages 143–154, 2005.

[8] Adriane Chapman and H. V. Jagadish. Why not? In *SIGMOD Conference*, pages 523–534, 2009.

[9] Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. Aligning event logs and declarative process models for conformance checking. In *BPM*, pages 82–97, 2012.

[10] Fábio de Lima Bezerra and Jacques Wainer. Algorithms for anomaly detection of traces in logs of process aware information systems. *Inf. Syst.*, 38(1):33–44, 2013.

[11] Luping Ding, Songting Chen, Elke A. Rundensteiner, Jun'ichi Tatemura, Wang-Pin Hsiung, and K. Selçuk Candan. Runtime semantic query optimization for event stream processing. In *ICDE*, pages 676–685, 2008.

[12] Joost Engelfriet. Branching processes of petri nets. *Acta Inf.*, 28(6):575–591, 1991.

[13] Anna Gambin, Jan Hidders, Natalia Kwasnikowska, S^{a}awomir Lasota, Jacek Sroka, Jerzy Tyszkiewicz, and Jan Van den Bussche. Nrc as a formal model for expressing bioinformatics workflows. *Poster at ISMB*, 2005.

[14] Thomas Heinis and Gustavo Alonso. Efficient lineage tracking for scientific workflows. In *SIGMOD Conference*, pages 1007–1018, 2008.

[15] Tomasz Imielinski, Shamim A. Naqvi, and Kumar V. Vadaparty. Incomplete objects - a data model for design and planning applications. In *SIGMOD Conference*, pages 288–297, 1991.

[16] Tao Jin, Jianmin Wang, and Lijie Wen. Efficiently querying business process models with beehivez. In *BPM (Demos)*, 2011.

[17] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[18] Yoshikazu Kobayashi, Akihiro Kishimoto, and Osamu Watanabe. Evaluations of hash distributed a* in optimal sequence alignment. In *IJCAI*, pages 584–590, 2011.

[19] Ina Koch, Björn H. Junker, and Monika Heiner. Application of petri net theory for modelling and validation of the sucrose breakdown pathway in the potato tuber. *Bioinformatics*, 21(7):1219–1226, 2005.

[20] Ina Koch, Markus Schüler, and Monika Heiner. Stepp - search tool for exploration of petri net paths: A new tool for petri net-based path analysis in biochemical networks. *In Silico Biology*, 5, 2004.

[21] Solmaz Kolahi and Laks V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.

[22] Niels Lohmann, Eric Verbeek, and Remco M. Dijkman. Petri net transformations for business processes - a survey. *T. Petri Nets and Other Models of Concurrency*, 2:46–63, 2009.

[23] Kenneth L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[24] Jan Mendling, Hajo A. Reijers, and Wil M. P. van der Aalst. Seven process modeling guidelines (7pmg). *Information & Software Technology*, 52(2):127–136, 2010.

[25] Hamid R. Motahari Nezhad, Régis Saint-Paul, Fabio Casati, and Boualem Benatallah. Event correlation for process discovery from web service interaction logs. *VLDB J.*, 20(3):417–444, 2011.

[26] Christian Rohr, Wolfgang Marwan, and Monika Heiner. Snoopy - a unifying petri net framework to investigate biomolecular networks. *Bioinformatics*, 26(7):974–975, 2010.

[27] Peng Sun, Ziyang Liu, Susan B. Davidson, and Yi Chen. Detecting and resolving unsound workflow views for correct provenance analysis. In *SIGMOD Conference*, pages 549–562, 2009.

[28] Wil M. P. van der Aalst, Marlon Dumas, Chun Ouyang, Anne Rozinat, and Eric Verbeek. Conformance checking of service behavior. *ACM Trans. Internet Techn.*, 8(3), 2008.

[29] Jianmin Wang, Tao Jin, RaymondK. Wong, and Lijie Wen. Querying business process model repositories. *World Wide Web*, 2013, to appear.

[30] Jianmin Wang, Shaoxu Song, Xiaochen Zhu, and Xuemin Lin. Efficient recovery of missing events. *UNSW Computer Science and Engineering Technical Report no. UNSW-CSE-TR-201311*, ftp://ftp.cse.unsw.edu.au/pub/doc/papers/UNSW/201311.pdf.

[31] Jürgen Will and Monika Heiner. *Petri Nets in Biology, Chemistry, and Medicine: Bibliography*. BTU, Inst. of Computer Science, 2002.

Proof

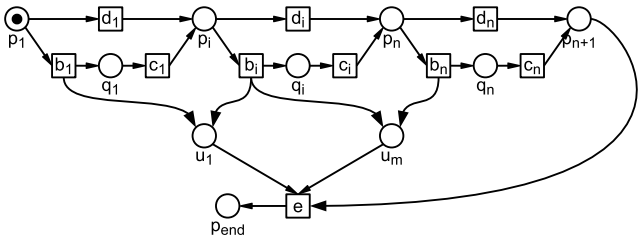


Figure 1: Reduction

Transformation To prove the NP-completeness of the repairing problem, we show a reduction from the set cover problem, which is one of Karp's 21 NP-complete problems [17].

Given a set of m elements $\mathcal{U} = \{u_1, \ldots, u_m\}, m = |\mathcal{U}|$ and n sets $\mathcal{S} = \{s_1, \ldots, s_n\}, n = |\mathcal{S}|$ such that $s_i \subseteq \mathcal{U}$ and $\cup_i s_i = \mathcal{U}$. A set cover is a $\mathcal{C} \subseteq \mathcal{S}$ of sets whose union is still $\mathcal{U}$. The minimum set cover problem is to identify the smallest number of sets whose union still contains all elements in $\mathcal{U}$.

We construct a process specification $\mathcal{N}(\mathcal{P}, \mathcal{T}, \mathcal{F})$ for the transformation, as illustrated in Figure 1. Let p_1 be the source place. For each $s_i \in \mathcal{S}, i = 1, \ldots, n$, we introduce two places $\mathsf{p}_i, \mathsf{q}_i$ and three transitions $\mathsf{b}_i, \mathsf{c}_i, \mathsf{d}_i$, such that $(\mathsf{p}_i, \mathsf{b}_i), (\mathsf{b}_i, \mathsf{q}_i), (\mathsf{q}_i, \mathsf{c}_i), (\mathsf{c}_i, \mathsf{p}_{i+1})$, $(\mathsf{p}_i, \mathsf{d}_i), (\mathsf{d}_i, \mathsf{p}_{i+1}) \in \mathcal{F}$. Moreover, for each element $u_j \in \mathcal{U}, j = 1, \ldots, m$, we insert a place u_j in $\mathcal{P}$.

Next, for each element u_j belonging to set s_i, we put an arc $(\mathsf{b}_i, \mathsf{u}_j) \in \mathcal{F}$. Finally, all the u_j together with p_{n+1} point to a transition $\mathsf{e} \in \mathcal{T}$, which leads to the sink place $\mathsf{p}_{\mathsf{end}}$.

Proof of Theorem 1 Given an empty sequence of σ, we can show that there is a set cover of size k if and only if the sequence has a recovery with cost $n + l + 1$.

First, let $\mathcal{C}$ be a set cover cover of size l. According to Figure 1, the execution cannot terminate until the flow on p_{n+1} is processed. For each set $s_i \in \mathcal{C}$, the corresponding p_i will choose b_i to execute (recover); otherwise, the flow w.r.t. d_i is selected. Referring to the requirement of set cover, each u_j corresponding to element u_j must has a flow executed leading from some b_i such that $u_j \in s_i$. Consequently, all the flows before transition e is processed. By filling e, the sequence is recovered. Note that for each p_i, either b_i or d_i should be processed, with total n recovered transitions. If a set s_i is included in the cover $\mathcal{C}$, i.e., s_i is processed, an additional c_i will be filled. Considering all l sets in $\mathcal{C}$, the total recovery cost is $n + l + 1$.

Conversely, assume that we have a recovery σ' with cost $\Delta(\sigma', \sigma) = n + l + 1 < n + l^* + 1$, where l^* is the size of a minimum set cover. According to the specification, there are exactly n transitions in the recovery belonging to either b_i or d_i. Moreover, in order to form a firing sequence leading to the sink place, the transition e must be executed. Thereby, the number of occurrence of c_i (as well as b_i) is exactly l. Note

that e can be processed only when all the flows on $\mathsf{u_i}$ and $\mathsf{p_{n+1}}$ are conducted. In other words, the l transitions of recovered $\mathsf{b_i}$ forms a set cover with size $l < l^*$, which is is a contradiction.

Proof of Lemma 3 Let n the length of σ_u. We prove the lemma inductively. For $n = 0$, i.e., $\sigma_u = \varepsilon$, we get $\tau_1 = \text{GAP}(\varepsilon\bullet, \bullet\mathsf{t}_1)$ and $\tau_2 = \text{GAP}(\varepsilon\bullet, \bullet\mathsf{t}_2)$. According to the definition of branching equivalence classes, it has $\pi((\sigma_u\tau_1\mathsf{t}_1)\bullet) = \pi((\sigma_u\tau_2\mathsf{t}_2)\bullet)$.

Assume that the lemma is true for all the length-k firing sequence σ_u^k having $\pi((\sigma_u^k\tau_1\mathsf{t}_1)\bullet) = \pi((\sigma_u^k\tau_2\mathsf{t}_2)\bullet)$, where $\tau_1 = \text{GAP}(\sigma_u^k\bullet, \bullet\mathsf{t}_1)$ and $\tau_2 = \text{GAP}(\sigma_u^k\bullet, \bullet\mathsf{t}_2)$. For $n = k + 1$, σ_u^{k+1} can be represented by a length-k firing sequence σ_u^k followed by a transition t. Let $\tau_1' = \text{GAP}(\sigma_u^k t\bullet, \bullet\mathsf{t}_1)$ and $\tau_2' = \text{GAP}(\sigma_u^k t\bullet, \bullet\mathsf{t}_2)$. We can show that $\pi((\sigma_u^k t\tau_1'\mathsf{t}_1)\bullet) = \pi((\sigma_u^k t\tau_2'\mathsf{t}_2)\bullet)$. There are two cases to consider, $t \in \tau_1 \cap \tau_2$ and $t \notin \tau_1 \cup \tau_2$.

Case 1. We define $S(\sigma_u) = \{t | t \in \sigma_u\}$. According to Definition 3, the post-set of a firing sequence $\sigma\bullet$ can be denoted as $\varepsilon \bullet - \cup_{t\in S(\sigma)} \bullet t + \cup_{t\in S(\sigma)} t\bullet$. Two firing sequences σ_1 and σ_2 with the same $S(\sigma_1) = S(\sigma_2)$ must have $\pi(\sigma_1\bullet) = \pi(\sigma_2\bullet)$. Since $t \in \tau_1 \cap \tau_2$, we have $S(t\tau_1') = S(\tau_1)$ and $S(t\tau_2') = S(\tau_2)$, it follows $\pi((\sigma_u^k t\tau_1'\mathsf{t}_1)\bullet) = \pi((\sigma_u^k\tau_1\mathsf{t}_1)\bullet) = \pi((\sigma_u^k\tau_2\mathsf{t}_2)\bullet) = \pi((\sigma_u^k t\tau_2'\mathsf{t}_2)\bullet)$.

Case 2. For $t \notin \tau_1 \cup \tau_2$, it always has $S(\tau_1') = S(\tau_1)$ and $S(\tau_2') = S(\tau_2)$, followed by $\pi((\sigma_u^k t\tau_1'\mathsf{t}_1)\bullet) = \pi((\sigma_u^k t\tau_1\mathsf{t}_1)\bullet) = \pi((\sigma_u^k t\tau_2\mathsf{t}_2)\bullet) = \pi((\sigma_u^k t\tau_2'\mathsf{t}_2)\bullet)$. The lemma holds for $n = k + 1$.

Proof of Theorem 2 Let $\tau_{pre}^* = \text{GAP}(\sigma_u\bullet, \bullet\mathsf{t}^*)$ denote the minimum fill, and $\tau_{post}^* = \text{GAP}(\sigma_u\tau_{pre}^*\mathsf{t}^*\bullet, \mathsf{p}')$ be the minimal recovery leading to a sink place p' among all the branches w.r.t. $\mathcal{T}_i^{EC}$. The entire recovered sequence is $\tau_{lo} = \sigma_u\tau_{pre}^*\mathsf{t}^*\tau_{post}^*$. Assume that there exists a local optimal recovery $\tau_{go} = \sigma_u\tau_{pre}'\mathsf{t}'\tau_{post}'$ such that $|\tau_{go}| < |\tau_{lo}|$, $\mathsf{t}' \in \mathcal{T}_i^{EC}$ and $|\tau_{pre}'| > |\tau_{pre}^*|$. That is, although τ_{go} is a local optimal solution, it's not local minimal. According to Lemma 3, we have $\pi((\sigma_u\tau_{pre}^*\mathsf{t}^*)\bullet) = \pi((\sigma_u\tau_{pre}'\mathsf{t}')\bullet)$, i.e., $\tau_{go}' = \sigma_u\tau_{pre}^*\mathsf{t}^*\tau_{post}'$ is a firing sequence. Thereby, τ_{go}' could be also a valid recovery. However, we now have $|\tau_{go}'| = |\sigma_u| + |\tau_{pre}^*| + |\tau_{post}'| + 2 < |\sigma_u| + |\tau_{pre}'| + |\tau_{post}'| + 2 = |\tau_{go}|$. It contradicts the assumption that τ_{go} is a local optimal solution.