Computing and Informatics, Vol. 39, 2020, 213–245, doi: 10.31577/cai_2020_1-2_213

ANALYSIS AND APPLICATION OF MIN-COST TRANSITION SYSTEMS TO BUSINESS PROCESS MANAGEMENT

Xiwen Feng, Dong Han

College of Energy and Mining Engineering
State Key Laboratory of Mining Disaster Prevention and Control
Co-founded by Shandong Province and the Ministry of Science and Technology
National Demonstration Center for Experimental Mining Engineering Education
Shandong University of Science and Technology
Qingdao 266590, China

Yinhua Tian*

Department of Information Engineering
Shandong University of Science and Technology
Taian 271000, China
e-mail: skdxxtyh@163.com

Abstract. To improve the efficiency of conformance checking in process mining, new alignment approaches are presented between event logs and process models based on the min-cost transition systems of Petri nets. An algorithm is presented to obtain the transition system with the minimum cost based on the product of the event net and process net. The min-cost transition system is a directed acyclic graph, where the paths from the initial node to the final node include all optimal alignments between the trace and the process model based on the given cost function. Two algorithms are proposed to calculate an optimal alignment and all optimal alignments, respectively. All algorithms are implemented in ProM platform. After a series of the simulation experiments, the feasibility and effectiveness of the proposed approaches are illustrated.

* Corresponding author

Keywords: Petri nets, event logs, process models, transition systems, business process management

Mathematics Subject Classification 2010: 68-Q05

1 INTRODUCTION

Business process management (BPM) aims to provide the unified modeling, running and monitoring environment for business processes from information technology and management technology [1]. In order to manage business processes better, the increasing enterprises and organizations utilize models to describe business processes. So, they can automatically implement processes, interact with participants and evaluate business processes [2]. Nowadays, most of the enterprises and organizations have established information management systems. With the continuous implementation of business processes, information systems will generate a large number of files on event logs. These files record massive data related to the execution processes, which are used to further analyze the performance of business processes in order to operate enterprises better [3].

Along with the increasing demand for business intelligence automatically extracted from event logs, process mining plays still more and more important role in business process management [4, 5, 6]. In enterprises, complete information management systems require high fitness between process models and event logs. However, there are always some deviations between the event logs recorded in the information system and the business process based models. Due to the deviations, event logs cannot be correctly replayed by the models. Because models are effective tools to identify and simulate the information systems, conformance checking becomes the necessary means to measure the compliance of process models and event logs.

At present, there are many conformance checking technologies between given models and event logs [7, 8, 9, 10, 11, 12, 13, 14, 15]. Alignment is one of the most advanced approaches. The main idea of alignment is to locate the deviations between process models and event logs. In general, the alignment results with the minimum deviations are considered to be the optimal alignments.

Through the analysis of various alignment approaches [16, 17, 18, 19, 20, 21, 22, 23], we find the existing problems of the current ones, mainly including: too large search space; high complexity of the search algorithms; unable to find the required and accurate optimal alignments; unable to find all the optimal alignments, and so on.

In our opinion, the alignment approaches can be divided into two steps:

1. generate a search space containing the optimal alignments according to traces and process models;
2. search for the optimal alignments in the search space based on the given cost function [24, 25].

The main framework of our approaches is shown in Figure 1.

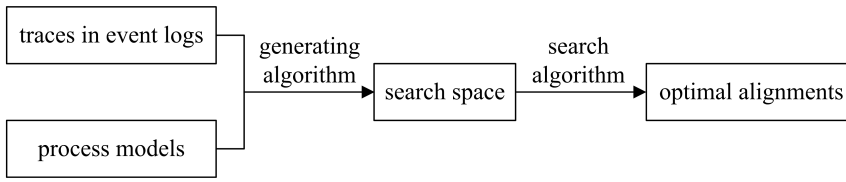


Figure 1. Framework of the alignment approaches

In the framework, the alignment approaches take the generating algorithm as preprocessing. When evaluating the performance of the alignment approaches, only the efficiency of the search algorithm is considered, including the mean computation time and the mean queued states. The search algorithms are widely used, well established and relatively easy to understand and implement. It is feasible to find and use an efficient search algorithm. Hence, to improve the efficiency of alignment approaches, the main means is to reduce the search space. In addition, if the search space is generated based on the trace and the process model but ignoring the cost function and other factors, the search space will include some redundant nodes which cannot reach to the optimal alignment. So, reducing the search space is the most effective approach to improve the efficiency of calculating optimal alignments.

In this paper, new alignment approaches are proposed, which can obtain the minimum space containing all the optimal alignments. The greatest advantage of our approaches is that the space contains only the useful nodes that can reach the optimal alignments, but no other redundant nodes. The main research objects of our approaches are traces in event logs and Petri nets-based models. Our approaches can generate a graph, in which a path from the initial node to the final node corresponds to an optimal alignment between traces and process models. The graph is called the min-cost transition system. By the means of the min-cost transition system, our approaches not only simplify the calculation procedures of optimal alignments, but also save the memory occupied by the search space.

The rest of this paper is organized as follows. Section 2 recalls some basic notions of Petri nets, event logs, alignment, and so on. The generating algorithm of the min-cost transition system is presented in Section 3. Section 4 proposes the approaches how to search for an optimal alignment and all optimal alignments in the min-cost transition system, respectively. Simulation experiments are done to illustrate the feasibility and effectiveness of our approaches in Section 5. Section 6 draws the conclusion and the future work.

2 PRELIMINARIES

In this section, we introduce the basic notations for multi-set, trace, event log, Petri nets, and so on.

A multi-set is a special set that allows multiple occurrences of the same element [26]. In a multi-set, only the number of occurrences of each element is concerned, and the order of occurrence of the elements is neglected.

Definition 1 (Multi-set)**.** Let S be a set. A *multi-set* S' over S is a mapping function $S' : S \to N^{0+}$.

N^{0+} refers to a set of zero and positive integers. Symbol $\emptyset$ denotes empty multi-set, and $\in$ denotes the inclusion relationship between elements and multi-sets. $\mathbb{B}(S)$ denotes the set of all multi-sets over a finite set S. $|S|$ is defined as the size of multi-set S.

Sequence is one of the most natural and appropriate ways to present traces in event logs [26].

Definition 2 (Sequence)**.** Let S be a set. σ is a finite *sequence* over S, written as $\sigma = \langle \sigma[1], \sigma[2], \sigma[3], \ldots, \sigma[n] \rangle$. σ is represented by listing its elements $\sigma[1]$, $\sigma[2]$, $\sigma[3]$, $\ldots$, $\sigma[n]$, where $\sigma[i] \in S (1 \leq i \leq n)$.

S^* denotes the set of all finite sequences over set S. $\langle\rangle$ denotes an empty sequence. Supposed that σ is a sequence over S, $\sigma[i]$ refers to the i^{th} element of σ. $\sigma[i] \in \sigma$ denotes the inclusion relationship, and $|\sigma|$ denotes the length of σ.

Let $x \in (S \times S)$ be a tuple of 2 elements(i.e., pair), $\pi_i(x)$ refers to the i^{th} element of x. For all $\sigma \in (S \times S)^*$, $\pi_i(\sigma) = \langle \pi_i(\sigma[1]), \pi_i(\sigma[2]), \pi_i(\sigma[3]), \ldots, \pi_i(\sigma[|\sigma|]) \rangle$. For all $Q \subseteq S$, $\sigma_{\downarrow Q}$ refers to the projection of $\sigma \in S^*$ on Q.

For any sequence σ over S, $\partial_{\text{set}}(\sigma) = \{\sigma[1], \sigma[2], \sigma[3], \ldots, \sigma[n]\}$, $\partial_{\text{multiset}}(\sigma) = [\sigma[1], \sigma[2], \sigma[3], \ldots, \sigma[n]]$. ∂_{set} converts a sequence into a set and $\partial_{\text{multiset}}$ converts a sequence into a multi-set. These conversions allow us to treat sequences as sets and multi-sets when needed.

A large number of events are recorded in the current information system and stored in the logs. An event log consists of cases and cases consist of events. The events for a case are represented in the form of a trace [15].

Definition 3 (Trace, Event log)**.** Let A be a set of activities. A *trace* $\sigma \in A^*$ is a process instance, i.e., a sequence of activities. An *event log* $L \in \mathbb{B}(A^*)$ is a multi-set of traces.

Petri nets are the most frequently used process modeling languages allowing for the modeling of concurrency [27]. The state of a Petri net is indicated by the distribution of tokens over places, and it is referred to as marking [28, 29, 30, 31, 32].

Transitions of Petri nets can be labeled with activities. Once the mapping relationship between transitions and activities is established, the transitions are related to the activities in the actual business process [15, 26].

Definition 4 (Labeled Petri net System)**.** Let A be a set of activities. A *labeled Petri net system* over A is a tuple $N = (P, T; F, \alpha, m_{\text{i}}, m_{\text{f}})$, where

1. P is the set of places;

2. T is the set of transitions, and $P \cup T \neq \emptyset$, $P \cap T = \emptyset$;
3. $F \subseteq (P \times T) \cup (T \times P)$ is an arc set between transitions and places, i.e., a flow relation;
4. $\alpha : T \to A^{\tau}$ is a function that maps transitions to labels, and τ denotes the invisible transition, $A^{\tau} = A \cup \{\tau\}$;
5. m_{i} and m_{f} are the initial marking and final marking, respectively.

For convenience, in the remainder of this paper, labeled Petri net system is abbreviated as Petri net.

Definition 5 (Per-set, Post-set)**.** Let $N = (P, T; F, \alpha, m_{\mathrm{i}}, m_{\mathrm{f}})$ be a Petri net. For $\forall x \in P \cup T$,

$$^{\bullet}x = \{y \mid y \in P \cup T \wedge (y, x) \in F\}$$

$$x^{\bullet} = \{y \mid y \in P \cup T \wedge (x, y) \in F\}$$

where $^{\bullet}x$ represents the pre-set of x, $x^{\bullet}$ represents the post-set of x.

We describe the transition firing rules by using the multi-sets of places. For any reachable state $m \in \mathbb{B}(P)$, the transition firing rules of Petri net $N = (P, T; F, \alpha, m_{\mathrm{i}}, m_{\mathrm{f}})$ are as follows:

1. For transition $t \in T$, if $^{\bullet}t \in m$, t is enabled denoted by $m[t >$; and
2. If $m[t >$, it means that the transition t can occur under the marking m, and after the transition t is fired, a new marking m' is generated, denoted by $m[t > m'$, where $m' = m \biguplus t^{\bullet} -^{\bullet} t$.

The event net of a trace is a Petri net with a linear structure, such that each transition in the net represents a unique activity occurrence in the trace. After traces are modeled as event nets, all possible movements are explicitly modeled by taking the product of two Petri nets, which are the event net and process net. The product of two Petri nets is the union of both nets with extra synchronous transitions, which are constructed by pairing transitions in event net with transitions in process net which have the same labels [15]. In the product of two Petri nets, all the places, transitions and arcs of the event net and process net are preserved in the product of two Petri nets.

An alignment between the process model and the trace is a movement sequence, and the move relates an event in the trace to an activity in the process model [15]. The synchronous move means that an event recorded in the trace is allowed according to the modeled behavior. The log move means that a recorded event is not allowed by the modeled behavior of process model. The model move means that an event which should have been recorded according to the modeled behavior is missed in the trace. The log moves and model moves indicate the deviations between traces and process models.

The symbol $\Gamma_{\sigma,N}$ denotes the set of all alignments between σ and N.

Given a trace and a Petri net model, there may be several different alignments that can be constructed. In order to get the most suitable alignments, a cost function $c((a,t))$ should assign a certain value to each move. According to the assigned cost function, the alignments with the least total cost are called optimal alignments.

In this paper, the standard likelihood cost function $lc()$ is used to assign the cost to the moves, i.e., the cost value of a synchronous move, log move and model move is 0, 1 and 1, respectively [15].

The symbol $\Gamma^{o}_{\sigma,N,lc}$ denotes the set of all optimal alignments between σ and N based on the function $lc()$.

3 GENERATION OF MIN-COST TRANSITION SYSTEMS

Alignments indicate the deviations between the process model and the trace in the event log. To express the idea of the approaches presented in this paper more clearly, the given process model and trace are taken as examples to illustrate.

3.1 Log Model and Process Model

Let $A = \{a, b, c, d, e\}$ be a set of activities. Given an event log $L = [\sigma^{10}]$, where $\sigma = \langle a, e, d\rangle$. The event net is shown in Figure 2, which is built according to the definition of event net [15]. We call the event net as log model, denoted as $N_{\mathrm{lm}} = (P_{\mathrm{lm}}, T_{\mathrm{lm}}; F_{\mathrm{lm}}, \alpha_{\mathrm{lm}}, m_{\mathrm{i,lm}}, m_{\mathrm{f,lm}})$.

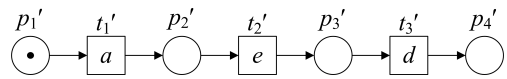


Figure 2. Log model N_{lm}

Given process model $N_{\mathrm{pm}} = (P_{\mathrm{pm}}, T_{\mathrm{pm}}; F_{\mathrm{pm}}, \alpha_{\mathrm{pm}}, m_{\mathrm{i,pm}}, m_{\mathrm{f,pm}})$, as shown in Figure 3.

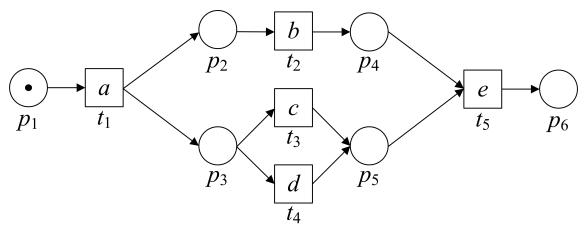


Figure 3. Process model N_{pm}

3.2 Product of Log Model and Process Model

The product model between the log model and the process model can be obtained. The product model consists of the log model, process model and synchronous transitions. The places, initial marking and final marking of the product model are the unions of the corresponding sets of the log model and the process model, respectively. Assuming that the log model $N_1 = (P_1, T_1; F_1, \alpha_1, m_{\mathrm{i},1}, m_{\mathrm{f},1})$ and the process model $N_2 = (P_2, T_2; F_2, \alpha_2, m_{\mathrm{i},2}, m_{\mathrm{f},2})$, the related information of the transitions in the product model is shown in Table 1. The arc relations can be established according to the pre-sets and the post-sets of transitions.

Transition	Type	Activity	Resource	Pre-Set	Post-Set
$(t_1, >>)$	log transition	$\alpha_1(t_1)$	$\{T_1\}$	${}^\bullet t_1$	$t_1^\bullet$
$(>>, t_2)$	model transition	$\alpha_2(t_2)$	$\{T_2 \mid \alpha(t_2) \neq \tau\}$	${}^\bullet t_2$	$t_2^\bullet$
(t_1, t_2)	synchronous transition	$\alpha_1(t_1) \setminus \alpha_2(t_2)$	$\{T_1 \times T_2 \mid \alpha(t_1) = \alpha(t_2) \neq \tau\}$	${}^\bullet t_1 \cup {}^\bullet t_2$	$t_1^\bullet \cup t_2^\bullet$
$(>>, t_2)$	invisible transition	$\alpha_2(t_2)$	$\{T_2 \mid \alpha(t_2) = \tau\}$	${}^\bullet t_2$	$t_2^\bullet$

Table 1. Transitions of the product of two Petri nets

Taking the log model N_{lm} and the process model N_{pm}, according to the conception of product of two Petri nets, the product model $N_{\mathrm{lm}*\mathrm{pm}} = (P_{\mathrm{lm}*\mathrm{pm}}, T_{\mathrm{lm}*\mathrm{pm}}; F_{\mathrm{lm}*\mathrm{pm}}, \alpha_{\mathrm{lm}*\mathrm{pm}}, m_{\mathrm{i,lm}*\mathrm{pm}}, m_{\mathrm{f,lm}*\mathrm{pm}})$ is built, as shown in Figure 4. According to Definition 4, the product model is also a Petri net.

3.3 Min-Cost Transition System of the Product Model

The transitions in the product model can be divided into four types: log transitions, model transitions, synchronous transitions and invisible transitions. Each transition is mapped to an activity, so the sort of transition also determines the sort of the activity. According to the standard likelihood cost function [15], we assign different weights to four types of transitions, and their corresponding relations are shown in Table 2.

Transition Type	Move Type	Weight Value
log transition	log move	1
model transition	model move	1
synchronous transition	synchronous move	0
invisible transition	invisible move	0

Table 2. Allocation of the weight value on transitions

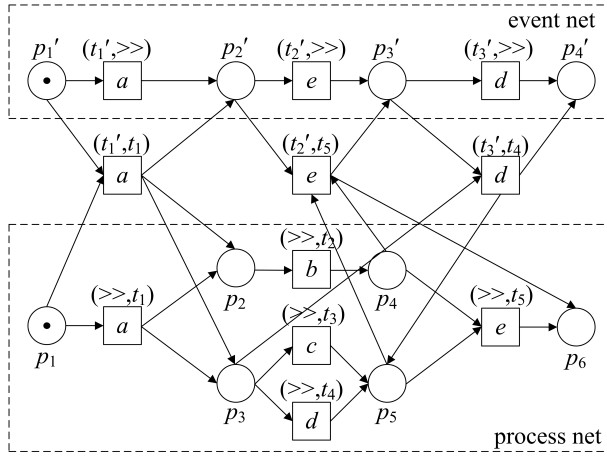


Figure 4. Product model $N_{\mathrm{lm}*\mathrm{pm}}$ between log model N_{lm} and process model N_{pm}

The product model is a Petri net, which can be performed for its reachable state graph. When calculating the reachable state graph, the weights of the transitions in the firing sequence are accumulated as the cost of the current reachable state. The min-cost reachable state graph can be obtained by counting the initial state, the minimum cost final state and all reachable states between them, which is called min-cost transition system. In the min-cost transition system, each node contains not only the current reachable marking but also its minimum cost.

Taking Petri net $N = (P, T; F, \alpha, m_{\mathrm{i}}, m_{\mathrm{f}})$ as an example, we illustrate the generation process of its min-cost transition system and how the cost of a transition system is minimized. The main idea to generate the min-cost transition system for Petri net N is as follows:

Step 1: Suppose the minimum cost of the transition system is $mincost = +\infty$, and the state queue is $\emptyset$. Consider the state $(m_{\mathrm{i}}, 0)$ as the initial state and be enqueued, where m_{i} is the initial marking of Petri net N and the value 0 is the current cost because there has not any transition fired.

Step 2: Choose the state (m_x, c_x) as the current state, where $\nexists(m_x', c_x') \to (c_x > c_x')$. For all $t_x \in T$ that $m_x[t_x >$, generate the new state (m_y, c_y), where $m_x[t_x > m_y$ and $c_y = c_x + lc(t_x)$ ($lc(t_x)$ is the weight value of transition t_x).

Step 3: Examine the new generated state (m_y, c_y):

Step 3.1: If $m_y = m_{\mathrm{f}}$ and $mincost > c_y$, then $mincost = c_y$.

Step 3.2: If there is the existing state (m_y', c_y'), $m_y' = m_y$ and $c_y' = c_y$, then share the existing state; $m_y' = m_y$ and $c_y' < c_y$, then discard the generated state; $m_y' = m_y$ and $c_y' > c_y$, then delete the state (m_y', c_y') and enqueue the state (m_y, c_y).

Step 3.3: If $c_y > mincost$, then discard the generated state.

Step 4: Examine all the visited states, and delete the states without children.

Step 5: Continue to execute Step 4, until all the visited states have children.

Step 6: If there are unvisited states in the queue, jump to Step 2; else, the min-cost transition system is generated.

In the procedure mentioned above, Step 3.1 ensures that the min-cost transition system has a minimum cost. The last remained state whose marking is m_{f} is considered as the final state of the min-cost transition system, and its cost is the minimum cost. Step 3.2 ensures that there are not two states with the same markings and different costs. Step 3.3 guarantees that there are no states with the greater cost than that of the final state. Steps 4 and 5 guarantee that there are no states that cannot arrive at the final state. Hence, the procedure can guarantee that the cost of a transition system is minimized.

Min-cost transition system $G_{\mathrm{lm}*\mathrm{pm}}$ of product model $N_{\mathrm{lm}*\mathrm{pm}}$ can be obtained by preserving the valid states and the connecting edges between them. The specific transition system $G_{\mathrm{lm}*\mathrm{pm}}$ is shown in Figure 5.

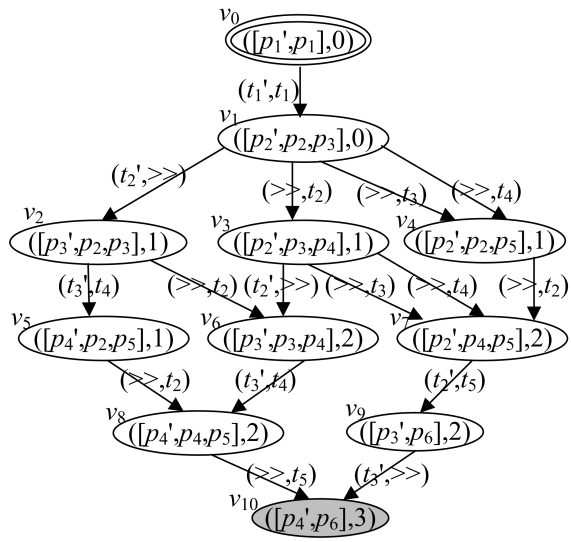


Figure 5. Min-cost transition system $G_{\mathrm{lm}*\mathrm{pm}}$

In the classic Petri nets, the marking represents the distribution of tokens in the places. However, in this paper, the state contains the marking as well as the cost in the min-cost transition system.

For arbitrary Petri nets, their structures may be very complicated and diverse. Here, we discuss a special structure for the Petri net and its influence on the min-cost

transition system. The Petri net contains cycles in which the cost of the transitions is 0. As a result, the min-cost transition system may contain cycles with cost 0, which results in countless paths between the initial node and the final node. In the context of practical application of our paper, the transitions with cost 0 are the invisible transitions in the Petri nets. The invisible transitions represent the activities that can never be observed, so the cycles containing only the invisible transitions have little meanings to the alignment results. In order to reach the final node from the initial node in a limited number of steps, we delete this kind of cycles from the min-cost transition system.

As shown in Figure 5, the min-cost transition system is a directed acyclic graph, which can also be called the min-cost reachability graph. In the graph, each node contains two attributes: one is the current marking of the product model that is represented by the multi-set of places; the other is the minimum cost of the state that is represented by a non-negative integer. The two attributes on the nodes are defined as the min-cost reachable state. In the graph, the label on the edge is the transition of the product model, which can be mapped to the move. There is one and only one node whose first attribute is the initial marking of the product net, which is called the min-cost initial state. There is one and only one node whose first attribute is the final marking of the product net, which is called the min-cost final state. Any node is on a path between the initial state and the final state.

3.4 Definitions of Min-Cost Transition System

Next, we discuss some special states in the transition system of the product model. And then, the definition of min-cost transition system is given and its basic properties are proved. For convenience, in this section, we agree as follows:

1. Petri nets are products of two Petri nets, so the concrete expression of transition is a tuple. However, in order to simplify the description, the symbol t is still used to represent the transition when the specific Petri net is not involved.
2. Transitions in the product nets can be mapped to moves, while moves can be mapped to real set by the standard likelihood cost function.

Hence, we can map the transitions to real set. Based on the weight assignment on the transitions in Table 2, the standard likelihood cost function $lc()$ is directly applied to the transitions, and its function value remains unchanged, which is called transition cost function.

Definition 6 (Reachable state with cost)**.** Let A be a set of activities. $N = (P, T; F, \alpha, m_{\mathrm{i}}, m_{\mathrm{f}})$ is a Petri net over A. $lc()$ is a transition cost function. Supposing there is a transition firing sequence $t_1t_2t_3 \ldots t_n$ that makes $m_{\mathrm{i}}[t_1t_2t_3 \ldots t_n > m$, it is said that $(m, \sum_{\mathrm{i}=1}^{n} lc(t_i))$ is a *reachable state with cost*, denoted by m^c.

M^c is a set that includes all of the reachable states with cost, i.e., $m^c \in M^c$. Definition 6 shows that m^c is a 2-tuple, in which the first element is the reachable

marking of the Petri net, and the second one is the sum of the cost for each transition in the firing sequence that causes the Petri net from the initial state to the current state.

Definition 7 (Min-cost reachable state)**.** Let A be a set of activities. $N = (P, T; F, \alpha, m_\mathrm{i}, m_\mathrm{f})$ is a Petri net over A. $lc()$ is a transition cost function. Supposing there is a transition firing sequence $t_1 t_2 t_3 \dots t_n$ that makes $m_\mathrm{i}[t_1 t_2 t_3 \dots t_n > m$. $(m, \sum_{\mathrm{i}=1}^{n} lc(t_i))$ is a *min-cost reachable state* if and only if there is no transition firing sequence $t'_1 t'_2 t'_3 \dots t'_k$ that makes $m_\mathrm{i}[t'_1 t'_2 t'_3 \dots t'_k > m$ and $\sum_{\mathrm{i}=1}^{n} lc(t_i) > \sum_{j=1}^{k} lc(t'_j)$, denoted by $m^{\copyright}$.

$M^{\copyright}$ is a set that includes all of the min-cost reachable states, i.e., $m^{\copyright} \in M^{\copyright}$. In Petri nets, different transition sequences may reach the same reachable state. Each reachable state and its minimum cost constitute the min-cost reachable state. Obviously, $M^{\copyright} \subseteq M^{c}$.

For convenience, in the remainder of this paper, we abbreviate the min-cost reachable state as the reachable state.

Definition 8 (Min-cost initial state)**.** Let A be a set of activities. $N = (P, T; F, \alpha, m_\mathrm{i}, m_\mathrm{f})$ is a Petri net over A. $lc()$ is a transition cost function. Given $m^{\copyright}$ is a min-cost reachable state, $m^{\copyright}$ is called as the *min-cost initial state* if $\pi_1(m^{\copyright}) = m_\mathrm{i}$, denoted by $m_\mathrm{i}^{\copyright}$.

We abbreviate the min-cost initial state as the initial state. According to Definition 8, the first element of the initial state is the initial marking of the Petri net. Since no transition has been fired at present, the second element of the initial state is 0. Hence, $m_\mathrm{i}^{\copyright} = (m_\mathrm{i}, 0)$.

Definition 9 (Min-cost final state)**.** Let A be a set of activities. $N = (P, T; F, \alpha, m_\mathrm{i}, m_\mathrm{f})$ is a Petri net over A. $lc()$ is a transition cost function. Given $m^{\copyright}$ is a min-cost reachable state, $m^{\copyright}$ is called as the *min-cost final state* if $\pi_1(m^{\copyright}) = m_\mathrm{f}$, denoted by $m_\mathrm{f}^{\copyright}$.

We abbreviate the min-cost final state as the final state. According to Definition 9, the first element of the final state is the final marking of the Petri net. The second element of the final state is the minimum cost from the initial marking to the final marking for the Petri net. Hence, $m_\mathrm{f}^{\copyright} = (m_\mathrm{f}, \min(\{\sum_{\mathrm{i}=1}^{n} lc(t_i) \mid \forall t_1 t_2 t_3 \dots t_n \rightarrow m_\mathrm{i}[t_1 t_2 t_3 \dots t_n > m_\mathrm{f}\}))$, where $\min(S)$ is a function to find the minimum cost in the set S.

Definition 10 (Min-cost transition system)**.** Let A be a set of activities. $N = (P, T; F, \alpha, m_\mathrm{i}, m_\mathrm{f})$ is a Petri net over A. $lc()$ is a transition cost function. *Min-cost transition system* $G = (V, E)$ is a directed acyclic graph, where V is a finite node set and $E \subseteq (V \times V)$ is a finite set of directed edges between nodes. The graph satisfies the following conditions:

1. $V \subseteq M^{\copyright}$;

2. $\exists! v_{\mathrm{i}} \in V : (\forall v \in V : (v, v_{\mathrm{i}}) \notin E) \Rightarrow \left(v_{\mathrm{i}} = m_{\mathrm{i}}^{\copyright}\right)$;
3. $\exists! v_{\mathrm{f}} \in V : (\forall v \in V : (v_{\mathrm{f}}, v) \notin E) \Rightarrow \left(v_{\mathrm{f}} = m_{\mathrm{f}}^{\copyright}\right)$;
4. $\forall v \in V$: v is on the path from v_{i} to v_{f};
5. $\forall e \in E, w(e) : w(e) \in T$, where $w(e)$ is the weight of edge e.

Again, we abbreviate the min-cost transition system as the transition system. According to Definition 10, the transition system has such characteristics as follows:

1. Each node is labeled by the min-cost reachable state.
2. There is only one node that is the min-cost initial state in the graph, which is called as the initial node.
3. There is only one node that is the min-cost final state in the graph, which is called as the final node.
4. Any node in the graph is on the path from the initial node to the final node.
5. The weight of the edge in the graph is the name of the transition.

Next, we present Theorem 1 and Theorem 2 to illustrate the rationality of the min-cost transition system.

Theorem 1. Let $N = (P, T; F, \alpha, m_{\mathrm{i}}, m_{\mathrm{f}})$ be a Petri net and its min-cost transition system be $G = (V, E)$. Given $m_1^{\copyright} \in V$ and $m_2^{\copyright} \in V$, if $m_1^{\copyright} \neq m_2^{\copyright}$, $\pi_1\left(m_1^{\copyright}\right) \neq \pi_1\left(m_2^{\copyright}\right)$.

Proof. For $\forall m_1^{\copyright} \in V$ and $\forall m_2^{\copyright} \in V$, if $m_1^{\copyright} \neq m_2^{\copyright}$, one of the following cases holds:

1. $\pi_1\left(m_1^{\copyright}\right) \neq \pi_1\left(m_2^{\copyright}\right)$ and $\pi_2\left(m_1^{\copyright}\right) \neq \pi_2\left(m_2^{\copyright}\right)$;
2. $\pi_1\left(m_1^{\copyright}\right) \neq \pi_1\left(m_2^{\copyright}\right)$ and $\pi_2\left(m_1^{\copyright}\right) = \pi_2\left(m_2^{\copyright}\right)$;
3. $\pi_1\left(m_1^{\copyright}\right) = \pi_1\left(m_2^{\copyright}\right)$ and $\pi_2\left(m_1^{\copyright}\right) \neq \pi_2\left(m_2^{\copyright}\right)$.

If case 1 or case 2 holds, the conclusion is found. Under case 3, supposed $\pi_2\left(m_1^{\copyright}\right) > \pi_2\left(m_2^{\copyright}\right)$, according to Definition 7, it is impossible that $m_1^{\copyright}$ is the min-cost reachable state; vice versa, so case 3 will never happen.

Hence, if $m_1^{\copyright} \neq m_2^{\copyright}$, $\pi_1\left(m_1^{\copyright}\right) \neq \pi_1\left(m_2^{\copyright}\right)$. □

Theorem 1 shows that the reachable markings of any two reachable states are different in the transition system. Hence, according to Theorem 1, both the initial node and the final node are unique.

Theorem 2. Let $N = (P, T; F, \alpha, m_{\mathrm{i}}, m_{\mathrm{f}})$ be a Petri net and its min-cost transition system be $G = (V, E)$. For $\forall v \in V$, there must be a transition firing sequence $t_1 t_2 t_3 \ldots t_k$ that makes $m_{\mathrm{i}}[t_1 t_2 t_3 \ldots t_k > \pi_1(v)$. Similarly, there must be $t_{k+1} t_{k+2} t_{k+3} \ldots t_n$ that makes $\pi_1(v)[t_{k+1} t_{k+2} t_{k+3} \ldots t_n > m_{\mathrm{f}}$.

Proof. According to Definition 10, $V \subseteq M^{\textcircled{c}}$. For $\forall v \in V$, $v \in M^{\textcircled{c}}$. According to Definition 7, there must be a transition firing sequence $t_1t_2t_3 \ldots t_k$ that makes $m_{\mathrm{i}}[t_1t_2t_3 \ldots t_k > \pi_1(v)$.

We suppose that there is no transition firing sequence $t_{k+1}t_{k+2}t_{k+3} \ldots t_n$ that makes $\pi_1(v)[t_{k+1}t_{k+2}t_{k+3} \ldots t_n > m_{\mathrm{f}}$. If $\pi_2(v) > \pi_2\left(m_{\mathrm{f}}^{\textcircled{c}}\right)$, for $\forall t_i \in T$, $lc(t_i) \geq 0$, then it will never reach $m_{\mathrm{f}}^{\textcircled{c}}$ from v, so v can be deleted directly. This shows that $\pi_2(v) > \pi_2\left(m_{\mathrm{f}}^{\textcircled{c}}\right)$ is not founded. If $\pi_2(v) \leq \pi_2\left(m_{\mathrm{f}}^{\textcircled{c}}\right)$, we suppose that t_x can be fired under the reachable marking $\pi_1(v)$, and then $\pi_1(v)[t_x > v_x$. We consider all three cases:

1. Assume $v_x = m_{\mathrm{f}}^{\textcircled{c}}$, $\pi_1(v)[t_x > m_{\mathrm{f}}$ shows that the conclusion is rational.
2. Assume $\pi_2(v_x) > \pi_2\left(m_{\mathrm{f}}^{\textcircled{c}}\right)$, v_x will be deleted, and if v has no other child, v will also be deleted.
3. Assume $\pi_2(v_x) \leq \pi_2\left(m_{\mathrm{f}}^{\textcircled{c}}\right)$.

We consider v_x as v to continue the comparison process until there is no node v_n that makes $\pi_2(v_n) \leq \pi_2\left(m_{\mathrm{f}}^{\textcircled{c}}\right)$. Through the analysis, we can infer that for $\forall v \in V$, either v will be deleted because of the failure to reach the final node, or there will be a transition firing sequence $t_{k+1}t_{k+2}t_{k+3} \ldots t_n$ that makes $\pi_1(v)[t_{k+1}t_{k+2}t_{k+3} \ldots t_n > m_{\mathrm{f}}$.

Hence, $\forall v \in V : (\exists t_1t_2t_3 \ldots t_k \Rightarrow m_{\mathrm{i}}[t_1t_2t_3 \ldots t_k > \pi_1(v)) \wedge (\exists t_{k+1}t_{k+2}t_{k+3} \ldots t_n \Rightarrow \pi_1(v)[t_{k+1}t_{k+2}t_{k+3} \ldots t_n > m_{\mathrm{f}})$. □

Theorem 2 shows that any node is on the path from the initial node to the final node in the transition system, that is, any node is connected with the initial node and the final node.

3.5 Calculation of Min-Cost Transition Systems

After describing the generation process of the transition system through an example and presenting the definition of the transition system, a specific algorithm to realize the calculation of the transition system in this subsection, seen Algorithm 1.

Before giving the specific algorithm, in order to facilitate the explanation of the algorithm, the variables and functions used in the algorithm are introduced, as shown in Table 3 and Table 4, respectively.

Algorithm 1 The generation algorithm of min-cost transition systems (reachability graphs) of Petri nets according to the transition cost function

Input: Petri net model $N = (P, T; F, \alpha, m_{\mathrm{i}}, m_{\mathrm{f}})$, transition cost function $lc()$.
Output: Min-cost transition system $G = (V, E)$.
Initialize: $unvisitedSet \leftarrow \emptyset$, $cost \leftarrow +\infty$, $V \leftarrow \left\{m_{\mathrm{i}}^{\textcircled{c}}\right\}$, $E \leftarrow \emptyset$.

1: $unvisitedSet \leftarrow \left\{m_{\mathrm{i}}^{\textcircled{c}}\right\}$;

Variable	Data Type	Function Introduction
curnode	m^c	current node
newnode	m^c	new node
foundnode	m^c	existing node
cost	value	current minimum cost
unvisitedSet	set	to store the unvisited nodes
V	set	to store the nodes
E	set	to store the edges

Table 3. Variable declaration in Algorithm 1

Function	Parameter Type	Return Type	Function Introduction
Father(*node*)	*node* : m^c	m^c	to return the parent of *node*
Delete(*node*)	*node* : m^c	null	to delete the node without child, and check its ancestors recursively
AddNode(*node*)	*node* : m^c	null	to add the node *node*
AddEdge(*fathernode*, *node*)	*fathernode* : m^c *node* : m^c	null	to add the edge between node and its parent

Table 4. Function declaration in Algorithm 1

2: **while** ($unvisitedSet \neq \emptyset$) **do**
3: Choose the minimum cost node from *unvisitedSet* as current node *curnode*;
4: $unvisitedSet \leftarrow unvisitedSet - \{curnode\}$;
5: **for** (all ($t_i \in T$ **and** $\pi_1(curnode)[t_i >)$) **do**
6: $newnode \leftarrow (\pi_1(curnode)[t_i >, \pi_2(curnode) + lc(t_i))$;
7: **if** ($\pi_1(newnode) = m_\mathrm{f}$) **then**
8: **if** ($\pi_2(newnode) < cost$) **then**
9: **if** ($cost \neq +\infty$) **then**
10: Find the node $(m_\mathrm{f}, cost)$;
11: $Delete((m_\mathrm{f}, cost))$;
12: **end if**
13: $cost \leftarrow \pi_2(newnode)$;
14: $V \leftarrow V \cup \{newnode\}$;
15: $AddEdge(curnode, newnode)$;
16: **else**
17: **if** ($\pi_2(newnode) = cost$) **then**
18: Find the previous node $foundnode$ that is the same as *newnode*;
19: $AddEdge(curnode, foundnode)$;
20: **end if**
21: **end if**
22: **else**

```
23:         if (π2(newnode) > cost) then
24:           if (all transitions have been fired under π1(curnode) and curnode has
              no child) then
25:             Delete(curnode);
26:           end if
27:         else
28:           if (foundnode ∈ V and π1(foundnode) = π1(newnode)) then
29:             if (π2(foundnode) = π2(newnode)) then
30:               E ← E ∪ {(curnode, foundnode))};
31:             else
32:               if (π2(foundnode) < π2(newnode)) then
33:                 if (curnode has no child and all transitions have been fired under
                    π1(curnode)) then
34:                   Delete(curnode);
35:                 end if
36:               else
37:                 Delete(foundnode);
38:                 AddNode(newnode);
39:                 AddEdge(curnode, newnode);
40:               end if
41:             end if
42:           else
43:             AddNode(newnode);
44:             AddEdge(curnode, newnode);
45:           end if
46:         end if
47:       end if
48:     end for
49: end while
50: for (all cycles in G) do
51:     delete all the edges with cost 0;
52:     for (all nodes in the cycle ) do
53:       if (node has no out edge) then
54:         Delete(node);
55:       end if
56:     end for
57: end for
58: return  G = (V, E);
```

The computation complexity of the min-cost transition system of the Petri net is related to the number of the reachable states and that of the transitions fired by the Petri nets, which is a NP-hard problem. Although the min-cost transition system computed by this algorithm is a subgraph of the traditional reachable marking graph, its complexity is also very high. Especially when there are many transitions

with the concurrent relations in Petri nets, the number of reachable states increases exponentially, which even causes state space to explode.

Let $N = (P, T; F, \alpha, m_{\mathrm{i}}, m_{\mathrm{f}})$ be a Petri net. In this paper, N is considered to be sound if and only if $m_{\mathrm{f}} \in R(m_{\mathrm{i}})$, where $R(m_{\mathrm{i}})$ is the set which includes all the reachable markings from m_{i}.

The influence of the concurrent structures in the Petri net to its min-cost transition system is similar to the effect on its reachable marking graph. Due to the cost of the transitions, the transitions with the less cost will be fired in the choice structures and loop structures when generating the min-cost transition system. In this case, the scale of the min-cost transition system is mostly smaller than that of the reachable marking graph. However, in the sequence structures and concurrent structures, all the transitions should be fired, the min-cost transition system of the Petri net will be isomorphic to its reachable marking graph. Hence, if the Petri net is with the completely concurrent structures, the number of the state in the min-cost transition system will increase exponentially with the linear increase of the concurrent branches just as the reachable marking graph.

$m_{\mathrm{f}} \in R(m_{\mathrm{i}})$ is essential for Algorithm 1 to execute correctly. Too much concurrent branches in the Petri net maybe lead to state space explosion. Hence, in order to improve the availability of the algorithm, we only study the sound Petri nets with less concurrent transitions in this paper.

4 SEARCH ALGORITHM OF OPTIMAL ALIGNMENTS

The min-cost transition system of the product model can be obtained by Algorithm 1. In the transition system, the sequence of weights labeled on the directed edges of any path from the initial node to the final node corresponds to an optimal alignment between the trace and the model. Based on the min-cost transition system, two algorithms are presented in this section to calculate an optimal alignment and all optimal alignments, respectively.

4.1 Search Algorithm of an Optimal Alignment

In this subsection, we search for an optimal alignment in the min-cost transition system. In the generation process of the transition system, the states that cannot reach the final node are pruned, so all the nodes in the graph are valid.

In the transition system, as shown in Figure 5, the prefix alignment between the trace and the process model can be calculated according to the path from the initial node to any other node. Similarly, the optimal alignment between the trace and the process model can be inferred based on the path from the initial node to the final node in the graph.

In this paper, the path from the initial node to the final node is defined as the complete path, and the corresponding relationship between the complete path and the optimal alignment is proved.

Definition 11 (Complete path)**.** Let $G = (V, E)$ be a min-cost transition system. $m_{\mathrm{i}}^{\copyright}$ is the min-cost initial state and $m_{\mathrm{f}}^{\copyright}$ is the min-cost final state. A *complete path* is a sequence $\left\langle \left(m_{\mathrm{i}}^{\copyright}, (t'_1, t_1), m_2^{\copyright}\right), \ldots, \left(m_n^{\copyright}, (t'_n, t_n), m_{\mathrm{f}}^{\copyright}\right)\right\rangle$ of the min-cost reachable states, i.e., a path from $m_{\mathrm{i}}^{\copyright}$ to $m_{\mathrm{f}}^{\copyright}$, denoted by $m_{\mathrm{i}}^{\copyright} \Rightarrow m_{\mathrm{f}}^{\copyright}$.

Given a complete path, a complete movement sequence can be obtained through outputting all the weights labeled on the edges of the path and converting the weights into moves, referred to Definition 12.

Definition 12 (Complete movement sequence)**.** Let $G = (V, E)$ be a min-cost transition system. $\left\langle \left(m_{\mathrm{i}}^{\copyright}, (t'_1, t_1), m_2^{\copyright}\right), \ldots, \left(m_n^{\copyright}, (t'_n, t_n), m_{\mathrm{f}}^{\copyright}\right)\right\rangle$ is a complete path in G. A *complete movement sequence* is a sequence of successive moves corresponding to the weights on the edges of the complete path, denoted by λ.

Given a complete path, its corresponding movement sequence can be calculated, as detailed in Algorithm 2.

Algorithm 2 The algorithm to compute complete movement sequence λ of complete path $\left\langle \left(m_{\mathrm{i}}^{\copyright}, (t'_1, t_1), m_2^{\copyright}\right), \ldots, \left(m_n^{\copyright}, (t'_n, t_n), m_{\mathrm{f}}^{\copyright}\right)\right\rangle$.

Input: Complete path $\left\langle \left(m_{\mathrm{i}}^{\copyright}, (t'_1, t_1), m_2^{\copyright}\right), \ldots, \left(m_n^{\copyright}, (t'_n, t_n), m_{\mathrm{f}}^{\copyright}\right)\right\rangle$.
Output: Complete movement sequence λ.
Initialize: $\lambda \leftarrow \langle\rangle$.

1: Map complete path $\left\langle \left(m_{\mathrm{i}}^{\copyright}, (t'_1, t_1), m_2^{\copyright}\right), \ldots, \left(m_n^{\copyright}, (t'_n, t_n), m_{\mathrm{f}}^{\copyright}\right)\right\rangle$ to node path $\langle (v_1, e_1, v_2), \ldots, (v_n, e_n, v_{n+1})\rangle$;
2: $i \leftarrow 1$;
3: **while** $(i \neq n)$ **do**
4: $\quad e \leftarrow \pi_2((v_i, e_i, v_{i+1}))$;
5: $\quad t \leftarrow w(e)$;
6: $\quad$ **if** $(\pi_1(t) =$ " >> ") **then**
7: $\quad\quad x \leftarrow$ " >> ";
8: $\quad$ **else**
9: $\quad\quad x \leftarrow \alpha(t)$;
10: $\quad$ **end if**
11: $\quad y \leftarrow \pi_2(t)$;
12: $\quad \lambda \leftarrow \lambda \oplus < (x, y) >$;
13: $\quad i \leftarrow i + 1$;
14: **end while**
15: **return** λ;

Both the time complexity and space complexity of Algorithm 2 are related to the length of the complete path. Supposed that the length of the complete path is n in the transition system, both the time complexity and space complexity of Algorithm 2 are $O(n)$.

Taking transition system $G_{\text{lm}*\text{pm}}$ as an example, path $\langle(([p'_1, p_1], 0), (t'_1, t_1), ([p'_2, p_2, p_3], 0)), (([p'_2, p_2, p_3], 0), (t'_2, >>), ([p'_3, p_2, p_3], 1)), (([p'_3, p_2, p_3], 1), (t'_3, t_4), ([p'_4, p_2, p_5], 1)), (([p'_4, p_2, p_5], 1), (>>, t_2), ([p'_4, p_4, p_5], 2)), (([p'_4, p_4, p_5], 2), (>>, t_5), ([p'_4, p_6], 3))\rangle$ from $m_{\text{i}}^{\copyright}$ to $m_{\text{f}}^{\copyright}$ is a complete path. The weight sequence on the path is $\langle(t'_1, t_1), (t'_2, >>), (t'_3, t_4), (>>, t_2), (>>, t_5)\rangle$, and its corresponding movement sequence $\langle(a, t_1), (e, >>), (d, t_4), (>>, t_2), (>>, t_5)\rangle$ is a complete movement sequence. Obviously, this complete movement sequence is an optimal alignment between trace σ and process model $N_{\text{lm}*\text{pm}}$.

Theorem 3. Let $N_1 = (P_1, T_1; F_1, \alpha_1, m_{\text{i},1}, m_{\text{f},1})$ be a log model of trace σ and $N_2 = (P_2, T_2; F_2, \alpha_2, m_{\text{i},2}, m_{\text{f},2})$ be a process model. $N_3 = (P_3, T_3; F_3, \alpha_3, m_{\text{i},3}, m_{\text{f},3})$ is their product model and its transition system is $G = (V, E)$. λ is a complete movement sequence based on G. $\Gamma^o_{\sigma,N,lc}$ is the set of all optimal alignments between trace σ and model N_2. Then, $\lambda \in \Gamma^o_{\sigma,N,lc}$ is true.

Proof. We suppose that $\left\langle \left(m_{\text{i}}^{\copyright}, (t'_1, t_1), m_2^{\copyright}\right), \ldots, \left(m_n^{\copyright}, (t'_n, t_n), m_{n+1}^{\copyright}\right)\right\rangle$ is a complete path and its complete movement sequence is λ, where $m_1^{\copyright} = m_{\text{i}}^{\copyright}$, $m_{n+1}^{\copyright} = m_{\text{f}}^{\copyright}$. The transition sequence is $\rho = \left\langle w\left(m_1^{\copyright}, m_2^{\copyright}\right), w\left(m_2^{\copyright}, m_3^{\copyright}\right), \ldots, w\left(m_n^{\copyright}, m_{n+1}^{\copyright}\right)\right\rangle$. According to Algorithm 2, the mapping relationship between complete movement sequence λ and transition sequence ρ can be determined.

In the product of two Petri nets, the name of the transition meets the following conditions: $\pi_1(t_3) = t_1$ or $\pi_1(t_3) =>>$, $\pi_2(t_3) = t_2$ or $\pi_2(t_3) =>>$, where $t_1 \in T_1$, $t_2 \in T_2$, $t_3 \in T_3$. According to Algorithm 1,

1. $m_{\text{i},1} \xrightarrow{\pi_1(\rho)_{\downarrow T_1}} m_{\text{f},1}$;
2. $m_{\text{i},2} \xrightarrow{\pi_2(\rho)_{\downarrow T_2}} m_{\text{f},2}$.

According to Algorithm 2,

1. $\pi_1(\lambda)_{\downarrow A} = \sigma$;
2. $m_{\text{i},2} \xrightarrow{\pi_2(\lambda)_{\downarrow T_2}} m_{\text{f},2}$.

In addition, the final state $m_{\text{f}}^{\copyright}$ guarantees that $\forall\gamma \in \Gamma_{\sigma,N} : \pi_2\left(m_{\text{f}}^{\copyright}\right) \leq \sum_{(a,t)\in\gamma} lc((a, t))$ is true. Based on the transition cost function, $\pi_2\left(m_{\text{f}}^{\copyright}\right)$ is the number of the deviations in λ. According to the definitions of the alignment and optimal alignment [15], $\lambda \in \Gamma^o_{\sigma,N,lc}$. □

Theorem 3 shows that a complete path corresponds to an optimal alignment between the trace and the process model in the transition system. If we want to get an optimal alignment between the trace and the model, we only need to access any path from the initial node to the final node in the transition system. An optimal alignment can be obtained by recording the weights of the visited edges and mapping them to the moves.

Algorithm 3 is presented to describe the specific implementation steps of calculating an optimal alignment based on the transition system.

Algorithm 3 The search algorithm of an optimal alignment between trace σ and model N.

Input: Min-cost transition system $G = (V, E)$.
Output: Optimal alignment γ.
Initialize: $\gamma \leftarrow \langle\rangle$.

1: $curnode \leftarrow m_{\mathrm{i}}^{\textcircled{c}}$;
2: **while** $\left(curnode \neq m_{\mathrm{f}}^{\textcircled{c}}\right)$ **do**
3: Choose any out edge of the current node as *curedge*;
4: $t \leftarrow w(curedge)$;
5: Translate t to the corresponding move *curmove*;
6: $\gamma \leftarrow \gamma \oplus \langle curmove \rangle$;
7: Consider the end node of edge *curedge* as the current node *curnode*;
8: **end while**
9: **return** γ;

We can discuss the complexity of Algorithm 3 from the prespective of graph and alignment, respectively. Algorithm 3 is to traverse any path from the initial node to the final node in the min-cost transition system. Supposing that the number of the nodes is v, the time complexity and space complexity of Algorithm 3 are $O(v)$. However, the time complexity and space complexity of the algorithm are relatively low, which are related to the longest path between the initial node and the final node in the transition system. The path in the graph corresponds to the optimal alignment between the trace and the model, so the maximum length of the path is equal to that of the optimal alignment. Supposed that the maximum length of the optimal alignment between the trace and the model is n, both the time complexity and space complexity of Algorithm 3 are $O(n)$.

In transition system $G_{\mathrm{lm}*\mathrm{pm}}$, v_0 is the initial node and v_{10} is the final node. According to Algorithm 3, optimal alignment γ_1 between trace σ and model N_{pm} is obtained. The specific search procedures are shown in Figure 6.

In Figure 6, there are three types of connection lines between nodes: the solid lines represent the logical relationship between nodes; the dashed lines represent the access order between nodes according to Algorithm 3; the dotted lines represent the output sequence of the moves when calculating the optimal alignment.

4.2 Search Algorithm of All Optimal Alignments

In the transition system, a complete path from the initial node to the final node can be mapped to an optimal alignment, and then all complete paths correspond to all optimal alignments.

Theorem 4. Let $N_1 = (P_1, T_1; F_1, \alpha_1, m_{\mathrm{i},1}, m_{\mathrm{f},1})$ be a log model of trace σ and $N_2 = (P_2, T_2; F_2, \alpha_2, m_{\mathrm{i},2}, m_{\mathrm{f},2})$ be a process model. $N_3 = (P_3, T_3; F_3, \alpha_3, m_{\mathrm{i},3}, m_{\mathrm{f},3})$

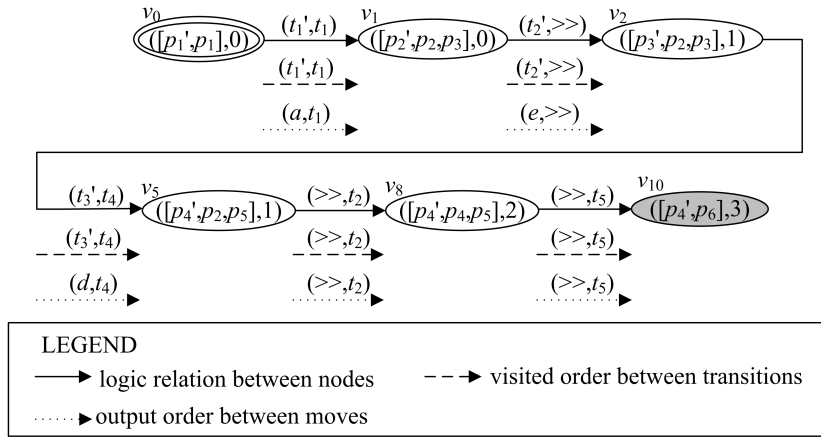


Figure 6. A search process of an optimal alignment in $G_{\text{lm}*\text{pm}}$

is their product model and its transition system is $G = (V, E)$. Λ is the set of all complete movement sequences based on G. $\Gamma^{o}_{\sigma,N,lc}$ is the set of all optimal alignments between trace σ and model N_2. Then, $\Lambda = \Gamma^{o}_{\sigma,N,lc}$ is true.

Proof. According to Theorem 3, $\forall\lambda \in \Lambda \Rightarrow \lambda \in \Gamma^{o}_{\sigma,N,lc}$. Then, $\Lambda \subseteq \Gamma^{o}_{\sigma,N,lc}$.

For $\gamma \in \Gamma^{o}_{\sigma,N,lc}$, there are three expressions that hold as follows:

1. $\pi_1(\gamma)_{\downarrow A} = \sigma$;
2. $m_{\text{i},2} \xrightarrow{\pi_2(\gamma)_{\downarrow T_2}} m_{\text{f},2}$;
3. given standard likelihood cost function $lc()$, $\forall\gamma' \in \Gamma_{\sigma,N,lc} : \sum_{(a,t)\in\gamma} lc((a,t)) \leq \sum_{(a',t')\in\gamma'} lc((a',t'))$.

According to the definition of the alignment and optimal alignment [15], optimal alignment γ is a movement sequence.

In the log model, $\alpha_1(t_{1,j}) = \sigma[j]$, where $t_{1,j} \in T_1$. So there is an inverse function $\alpha_1^{-1}(\sigma[j]) = t_{1,j}$, which maps optimal alignment γ to transition sequence ρ. In the product of two Petri nets, $\partial_{\text{set}}(\rho) \in T_3$. According to expression 1 mentioned above, $m_{\text{i},1} \xrightarrow{\pi_1(\rho)_{\downarrow T_1}} m_{\text{f},1}$; according to expression 2, $m_{\text{i},2} \xrightarrow{\pi_2(\rho)_{\downarrow T_2}} m_{\text{f},2}$. In the product of two Petri nets, $m_{\text{i},3} = m_{\text{i},1} \biguplus m_{\text{i},2}$, $m_{\text{f},3} = m_{\text{f},1} \biguplus m_{\text{f},2}$, $T_3 \subseteq (T_1^{>>} \times T_2^{>>})$, then $m_{\text{i},3} \xrightarrow{\rho} m_{\text{f},3}$. According to Algorithm 1, $\pi_1\left(m_{\text{i}}^{\copyright}\right) \xrightarrow{\rho} \pi_1\left(m_{\text{f}}^{\copyright}\right)$.

According to Algorithm 1, given the transition cost function, for $\forall\gamma' \in \Gamma_{\sigma,N,lc}$: $\pi_2\left(m_{\text{f}}^{\copyright}\right) \leq \sum_{(a',t')\in\gamma'} lc((a',t'))$. Combining with expression 3 mentioned above, $\sum_{(a,t)\in\gamma} lc((a,t)) = \pi_2\left(m_{\text{f}}^{\copyright}\right)$. So $\sum_{(t',t)\in\rho} lc((t',t)) = \pi_2\left(m_{\text{f}}^{\copyright}\right)$.

Hence, there is a complete path, just as $\left\langle m_1^{\copyright}, m_2^{\copyright}, \ldots, m_j^{\copyright}, \ldots, m_n^{\copyright} \right\rangle$, where $m_1^{\copyright} = m_{\text{i}}^{\copyright}$, $m_n^{\copyright} = m_{\text{f}}^{\copyright}$, $m_j^{\copyright} = \left(\pi_1\left(m_{j-1}^{\copyright}\right)[\rho[j-1]>, lc(\rho[j-1])\right)$ $(1 < j < n)$.

This complete path corresponds to a complete movement sequence, denoted by γ, which makes $\lambda = \gamma$ founded. So $\gamma \in \Lambda$, and then $\Gamma^o_{\sigma,N,lc} \subseteq \Lambda$.

In conclusion, $\Lambda = \Gamma^o_{\sigma,N,lc}$. □

Theorem 4 shows that the set of all complete movement sequences in the min-cost reachability graph is identical with the set of all optimal alignments between the trace and the model. All complete paths in the min-cost transition system correspond to all optimal alignments between the trace and the model. In other words, the complete movement sequence is equal to the optimal alignment, and they are legal movement sequences.

Next, Algorithm 4 is proposed to calculate all the optimal alignments between the trace and the model based on the standard likelihood cost function by the min-cost transition system. In Algorithm 4, two stacks are used. The description and operation of the stack are as follows:

- *nodestack*: node stack to store the visited nodes on the paths;
- *movestack*: move stack to store the weights of the directed edges between nodes in the node stack;
- *empty*(*stack*): a function to judge whether the stack is empty. If the stack is empty, it returns True; otherwise, it returns False;
- *gettop*(*stack*): get the top element of stack *stack*;
- *pop*(*stack*): pop up the top element of stack *stack*;
- *push*(*stack*, *node*): push element *node* into stack *stack*.

Algorithm 4 The search algorithm of all optimal alignments between trace σ and model N.

Input: Min-cost transition system $G = (V, E)$.
Output: Optimal alignment set $\Gamma^o_{\sigma,N,lc}$.
Initialize: $\Gamma^o_{\sigma,N,lc} \leftarrow \emptyset$, $nodestack \leftarrow \emptyset$, $movestack \leftarrow \emptyset$, $\gamma \leftarrow \langle\rangle$.

for (all ($edge \in E$)) **do**
  $flag(edge) \leftarrow 0$;
end for
$push\left(nodestack, m_{\mathrm{i}}^{\textcircled{c}}\right)$;
while (!$empty(nodestack)$) **do**
  $curnode \leftarrow gettop(nodestack)$;
  if ((each out edge *edge* of *curnode* has been visited) **or** $\left(curnode = m_{\mathrm{f}}^{\textcircled{c}}\right)$) **then**
    $pop(nodestack)$;
    if $\left(curnode \neq m_{\mathrm{i}}^{\textcircled{c}}\right)$ **then**
      $pop(movestack)$;
    end if
    $\gamma \leftarrow \gamma - \gamma[|\gamma|]$;
    if $\left(curnode = m_{\mathrm{f}}^{\textcircled{c}}\right)$ **then**
      $\Gamma^o_{\sigma,N,lc} \leftarrow \Gamma^o_{\sigma,N,lc} \cup \{\gamma\}$;

```
15:       else
16:         for (each out edge curedge of curnode) do
17:           flag(edge) ← 0;
18:         end for
19:       end if
20:     else
21:       flag(edge) ← 1;
22:       Consider the end node of edge as the current node curnode;
23:       push(nodestack, curnode);
24:       t ← w(edge);
25:       Translate t to the corresponding move move;
26:       push(movestack, move);
27:       γ ← γ ⊕ ⟨move⟩;
28:     end if
29: end while
30: return Γ^o_{σ,N,lc};
```

Similarly, we can discuss the complexity of Algorithm 4 from the prespective of graph and alignment, respectively. Algorithm 4 is to traverse all the paths from the initial node to the final node in the min-cost transition system. Because all paths between two nodes are required, every possibility must be examined, and backtracking is the only way. As for the complexity of this algorithm, it depends on the size of the graph and the number of the nodes. Supposing that the number of the nodes is v and the number of the edges is e, the time complexity and space complexity of Algorithm 4 are $O(ve)$.

However, the time complexity and space complexity of Algorithm 4 are related to the lengths of the complete paths and the number of complete paths between the initial node and the final node in the min-cost transition system. A complete path in the graph corresponds to an optimal alignment between the trace and the process model based on the given cost function, so the length of the complete path is equal to that of the optimal alignment. Meanwhile, the number of complete paths in the graph is equal to that of the optimal alignments. Supposed that the maximum length of optimal alignment is n and the number of optimal alignments is m, the time complexity and space complexity of Algorithm 4 are $O(mn)$.

According to Algorithm 4, the values of γ_1 to γ_7 are shown in Figure 7. There are seven different paths from the initial node to the final node in Figure 5, while there are seven different optimal alignments in Figure 7. In Figure 7, a vertical list represents a move in each optimal alignment. In order to explicitly compare the events in the trace with the activities mapped on the transitions in the model, the activities mapped on the transitions are also marked out in each optimal alignment.

According to Algorithm 4, the correspondence between complete paths, transition sequences and optimal alignments is shown in Table 5.

γ_1=

a	e	d	$>>$	$>>$
a	$>>$	d	b	e
t_1		t_4	t_2	t_5

γ_2=

a	e	$>>$	d	$>>$
a	$>>$	b	d	e
t_1		t_2	t_4	t_5

γ_3=

a	$>>$	e	d	$>>$
a	b	$>>$	d	e
t_1	t_2		t_4	t_5

γ_4=

a	$>>$	$>>$	e	d
a	b	c	e	$>>$
t_1	t_2	t_3	t_5	

γ_5=

a	$>>$	$>>$	e	d
a	b	d	e	$>>$
t_1	t_2	t_4	t_5	

γ_6=

a	$>>$	$>>$	e	d
a	c	b	e	$>>$
t_1	t_3	t_2	t_5	

γ_7=

a	$>>$	$>>$	e	d
a	d	b	e	$>>$
t_1	t_4	t_2	t_5	

Figure 7. All optimal alignments between trace σ and model $N_{\mathrm{lm}*\mathrm{pm}}$

Complete Path (Only Nodes)	Transition Sequence	Optimal Alignment
$(v_0, v_1, v_2, v_5, v_8, v_{10})$	$\langle (t'_1, t_1), (t'_2, >>), (t'_3, t_4), (>>, t_2), (>>, t_5) \rangle$	γ_1
$(v_0, v_1, v_2, v_6, v_8, v_{10})$	$\langle (t'_1, t_1), (t'_2, >>), (>>, t_2), (t'_3, t_4), (>>, t_5) \rangle$	γ_2
$(v_0, v_1, v_3, v_6, v_8, v_{10})$	$\langle (t'_1, t_1), (>>, t_2), (t'_2, >>), (t'_3, t_4), (>>, t_5) \rangle$	γ_3
$(v_0, v_1, v_3, v_7, v_9, v_{10})$	$\langle (t'_1, t_1), (>>, t_2), (>>, t_3), (t'_2, t_5), (t'_3, >>) \rangle$	γ_4
$(v_0, v_1, v_3, v_7, v_9, v_{10})$	$\langle (t'_1, t_1), (>>, t_2), (>>, t_4), (t'_2, t_5), (t'_3, >>) \rangle$	γ_5
$(v_0, v_1, v_4, v_7, v_9, v_{10})$	$\langle (t'_1, t_1), (>>, t_3), (>>, t_2), (t'_2, t_5), (t'_3, >>) \rangle$	γ_6
$(v_0, v_1, v_4, v_7, v_9, v_{10})$	$\langle (t'_1, t_1), (>>, t_4), (>>, t_2), (t'_2, t_5), (t'_3, >>) \rangle$	γ_7

Table 5. Mapping the optimal alignments to the paths in $G_{\mathrm{lm}*\mathrm{pm}}$

5 SIMULATION EXPERIMENTS

Given the process model, the trace and the standard likelihood cost function, Section 3 explains how to generate the min-cost transition system. In Section 4, we propose two algorithms to compute an optimal alignment and all optimal alignments according to the min-cost transition system, respectively. To facilitate the description of our approaches, we name the algorithm to compute an optimal alignment as *Min-cost algorithm-One*, and the algorithm to compute all optimal alignments as *Min-cost algorithm-All*.

This section presents several evaluations of simulation experiments about the proposed approaches to illustrate that our approaches can be finished in limited time by occupying limited space. The experiments are performed on a computer with Intel Core i7-6500U, 2.50 GHz CPU, 16.0 GB RAM, JDK 1.8, and Windows 7. We will compare the proposed approaches to illustrate their feasibility and effectiveness.

The approaches in this paper have been implemented as two plug-ins in the process mining framework ProM and are publicly available. The plug-ins are called "*Min-cost Algorithm-One*" and "*Min-cost Algorithm-All*". The first plug-in implements Algorithm 1, Algorithm 2 and Algorithm 3 presented in this paper and

its function is to compute an optimal alignment between the trace and the model based on the standard likelihood cost function. However, the second plug-in implements Algorithm 1, Algorithm 2 and Algorithm 4 presented in this paper and its function is to compute all optimal alignments between the trace and the model based on the standard likelihood cost function. Both plug-ins can be accessible at: `https://pan.baidu.com/s/11sAA_w5TBec08t7evHMyg`. The extraction code for downloading files is "57bg".

Through Algorithm 1, the min-cost transition system is obtained, which is the search space for the following search algorithms. So Algorithm 1 is the preparation. However, Algorithm 3 and Algorithm 4 do the search work in the search space. Algorithm 2 is invoked by Algorithm 3 and Algorithm 4 in the actual implementation. When we consider the performance of our approaches, Algorithms 3 and 4 are the main study objects.

Taking a business process from the inclined shaft in a coal mine as an example, the process model is constructed manually [32]. In order to enhance the safety of the transportation of the coal mine, a PLC-based distributed control system for the inclined shaft of coal mine is analyzed. And Petri nets are adopted to build the model of the system that simulates the process for building the route in the inclined shaft of the coal mine, shown in Figure 8 and denoted as N_{SE}. The meanings of the transitions in model N_{SE} are explained in Table 6.

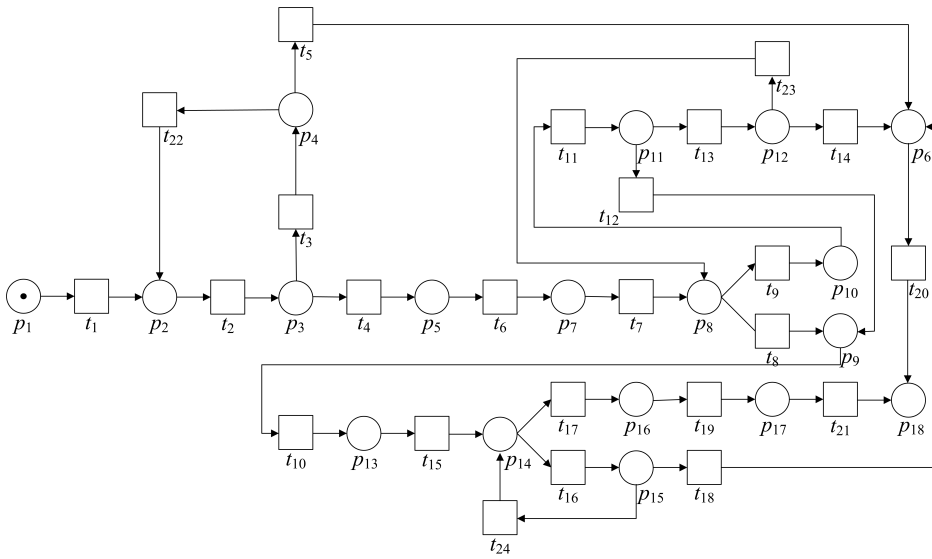


Figure 8. Petri net model N_{SE} for building the route in the inclined shaft of the coal mine

After the process model is introduced, the corresponding event logs can be obtained. We generate completely fit traces according to the process model with various lengths from 11 to 50 activities. Noise is introduced by randomly adding

Transition	Label	Meaning
t_1	a	build the route
t_2	b	obtain the target route state
t_3	c	fail to obtain the target route state
t_4	d	succeed in obtaining the target route state
t_5	e	report errors in obtaining the target route state
t_6	f	set the target route
t_7	g	check the consistency of turnouts
t_8	h	be consistent between turnout positions
t_9	i	be inconsistent between turnout positions
t_{10}	j	fix the turnout positions
t_{11}	k	switch the turnouts
t_{12}	l	succeed in switching the turnouts
t_{13}	m	fail to switch the turnouts
t_{14}	n	report errors in switching the turnouts
t_{15}	o	turn on the car stopper
t_{16}	p	fail to turn on the car stoppers
t_{17}	q	succeed in turning on the car stoppers
t_{18}	r	report errors in turning on the car stoppers
t_{19}	s	succeed in building the route
t_{20}	t	end abnormally and return
t_{21}	u	end normally and return
t_{22}	v	rebuild the route
t_{23}	w	recheck the consistency of turnouts
t_{24}	x	turn on the car stopper again

Table 6. The meanings of the transitions in model N_{SE}

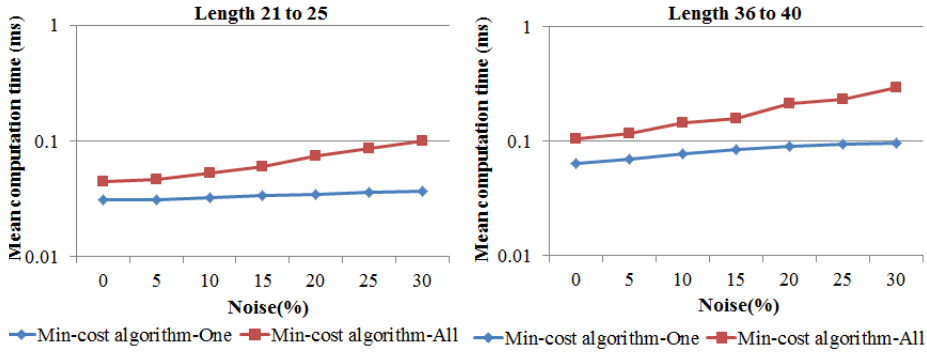
and/or deleting activities for every trace. The noise ratio is measured by the formula *noise = the number of the deviations/the length of the original fit trace*. Here, the number of the deviations is equal to that of the inserted activities and deleted activities.

For the traces with different noise ratios, the mean computation time and the mean queued states of constructing alignments are compared between both approaches. For the traces of lengths from 21 to 25 and 36 to 40, the computation time and the queued states for constructing alignments are also compared between the two approaches. For different trace lengths, we also compare the computation time and the queued states between the two approaches. In this experiment, two kinds of event logs are used. One includes the completely fit traces of various lengths and the other is the unfit traces of various lengths between the specified values. In this paper, each unfit trace has a noise ratio between 5 % and 30 %.

The experiment data in this paper are 28 event logs, and each event log includes 100 traces. Every result recorded in this paper is the average value of the same experiment repeated for 10 times.

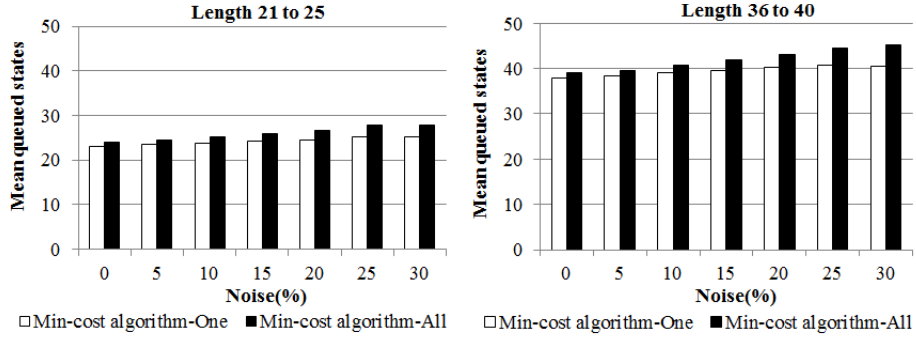
5.1 Noise Level

In this subsection, the mean computation time and the mean queued states are compared between Min-cost algorithm-One and Min-cost algorithm-All. The traces used in this experiment have various noise ratios. For the process model, two event log sets are generated, in which the trace lengths are from 21 to 25 and from 36 to 40, respectively. Each event log has the traces with the fixed noise ratio. The comparison results of the mean computation time are shown in Figure 9, and that of the mean queued states are shown in Figure 10. In Figure 9, y-axis is shown by a logarithmic scale.



a) b)

Figure 9. Comparison of computation time with different noise levels



a) b)

Figure 10. Comparison of queued states with different noise levels

Figure 9 shows that the mean computation time of constructing alignments increases as the length of traces and the noise ratios. When a trace length is from 21

to 25 in Figure 9 a), the mean computation time of the two approaches is less than 0.1 milliseconds, so it is very short. Min-cost algorithm-One needs the shorter time. In this experiment, the computation time of Min-cost algorithm-All is between 0.04 ms and 0.1 ms, and that of Min-cost algorithm-One is between 0.03 ms and 0.04 ms. The computation time of the two approaches increases exponentially as the noise ratios of the traces. However, the computation time of Min-cost algorithm-All increases faster than the other approach. In Figure 9 b), the growths of the computation time for the approaches are similar to the results shown in Figure 9 a). However, all data of the computation time shown in Figure 9 b) are higher than the results shown in Figure 9 a). In this experiment, the computation time of Min-cost algorithm-One lies in between 0.06 ms and 0.1 ms, and that of Min-cost algorithm-All lies in between 0.1 ms and 0.3 ms.

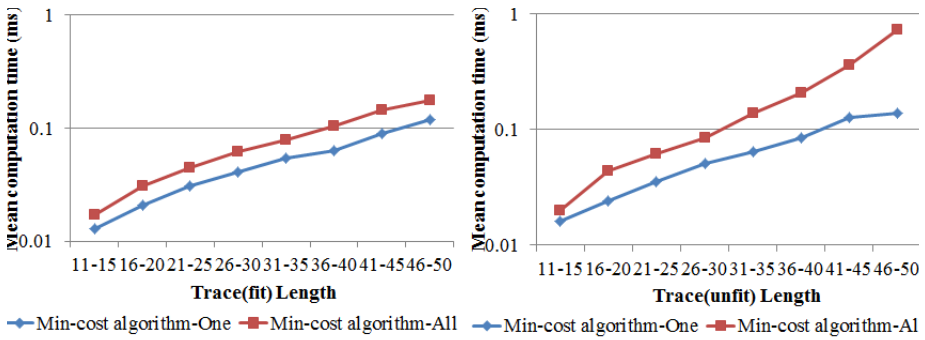
Whether in Figure 10 a) or Figure 10 b), the mean queued states of Min-cost algorithm-All are obviously higher than the other approach. However, the queued states of the approaches increase relatively slowly along with the growth of the noise ratios. Compared the data in Figure 10 a) with that in Figure 10 b), when the traces have the same noise ratios, the queued nodes of the traces with the lengths from 36 to 40 is almost 1.5 times as many as that of the traces with the lengths from 21 to 25 by the same approaches. On the study of the related data, the average length of the traces from 36 to 40 is also 1.5 times as that of the traces from 21 to 25, which is in accordance with the above-mentioned conclusion.

Hence, as the lengths of traces and the noise ratios increase, the time complexity and space complexity of Min-cost algorithm-One grow much slower than that of Min-cost algorithm-All.

5.2 Trace Length

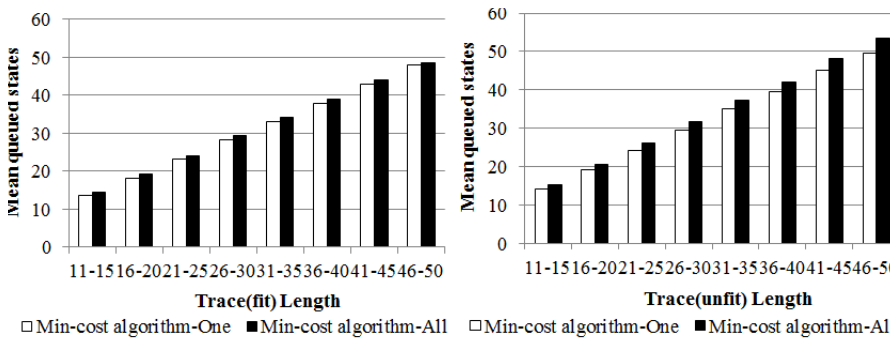
This experiment is conducted to illustrate the effect of the trace lengths. We aim to compare Min-cost algorithm-One and Min-cost algorithm-All for different trace lengths. For the process model, two kinds of event log sets are generated with different average lengths of the traces from 11 to 50. One includes the completely fit traces, and the other includes the unfit traces with different noise ratios. Here, the noise ratio of each trace is a random value from 5 % to 30 %. The comparison results of the mean computation time are shown in Figure 11, and that of the mean queued states are shown in Figure 12. In Figure 11, y-axis is shown by a logarithmic scale.

In Figure 11, no matter the traces are fit or unfit to process model N^{SE}, the mean computation time of Min-cost algorithm-One is lower than that of Min-cost algorithm-All. Figure 11 a) shows the computation time when the traces can be rightly replayed by the given model, and Figure 11 b) shows the computation time when the traces have noise ratios from 5 % to 30 %. In Figure 11 a), the growth rates of the computation time for the two approaches are almost equal. In Figure 11 b), the computation time of Min-cost algorithm-All grows the faster along with the growth of the trace lengths. When the lengths of the traces are larger than 45, the



a) b)

Figure 11. Comparison of computation time between different trace lengths



a) b)

Figure 12. Comparison of queued states between different trace lengths

computation time of the Min-cost algorithm-All has an obvious growth. However, the computation time of Min-cost algorithm-One increases relatively slow.

In Figure 12, the queued states of Min-cost algorithm-One are slightly fewer than that of Min-cost algorithm-All, but the two values are very close to each other. Compared the dots in Figure 12 a) with those in Figure 12 b), the queued states of each approach for the fit traces are less than those for the unfit traces. The queued states of the two approaches have a linear growth along with the increase of the trace lengths.

In conclusion, Min-cost algorithm-One outperforms Min-cost algorithm-All. Of course, Min-cost algorithm-One can only obtain the optimal alignment, but Min-cost algorithm-All can get all. In addition, we ignore the generation process of the min-cost transition system. The focus of these experiments is not to compare the two approaches, but to show that both approaches can be completed in limited time and space.

6 CONCLUSIONS

Conformance checking plays an increasingly important role in information management systems. Alignment is one of the most advanced and comprehensive conformance checking. By means of alignment approaches, the optimal alignments between traces and models based on the cost functions can be obtained. The results of optimal alignments can be applied to all aspects of process mining. However, the search space generated by some existing alignment approaches is so large that seriously affects the search efficiency of optimal alignments. In this paper, we propose new approaches that can align observed and modeled behaviors based on the min-cost transition systems. In the transition system, all paths from the initial node to the final node can be mapped to all the optimal alignments between the trace and the process model. All optimal alignments can be obtained and output by the traversing algorithm of graphs.

The alignment approaches proposed in this paper generate a min-cost transition system which includes all the optimal alignments between the trace and model based on the given cost function. In the min-cost transition system, the optimal alignments can be quickly found. Finally, all the algorithms in this paper are simulated on ProM. The simulation results show that the alignment approaches proposed in this paper are feasible and effective.

The alignment approaches presented in this paper are feasible and effective when dealing with the artificial logs and models. In the future work, we intend to mainly carry out the following research: Firstly, we can try to propose more efficient algorithms to generate the min-cost transition system for the Petri net. Secondly, the approaches presented in this paper will be simulated using more real-life cases to verify its robustness and stability. Then, it will be further to compare the min-cost transition systems in this paper with the classic reachability graphs of Petri nets, and find the differences between the optimal alignments under the different cost functions, as well as determine their own application areas. Finally, the idea of both products of two Petri nets and min-cost transition systems will be applied to process discovery as well as model repair and enhancement in order to improve the fitness between observed and modeled behaviors.

Acknowledgement

This work was supported in part by the Natural Science Foundation of China under the grant No. 61973180, in part by the Taishan Scholar Construction Project of Shandong Province, in part by the Key Research and Development Program of Shandong Province under the grant No. 2018GGX101011, and in part by the Natural Science Foundation of Shandong Province under the grants No. ZR2018MF001 and No. ZR2019MF033.

REFERENCES

[1] van der Aalst, W. M. P: Business Process Management: A Comprehensive Survey. ISRN Software Engineering, Vol. 2013, 2013, Art. No. 507984, 37 pp., doi: 10.1155/2013/507984.

[2] van der Aalst, W. M. P.—Adriansyah, A.—de Medeiros, A. K. A. et al.: Process Mining Manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (Eds.): Business Process Management Workshops (BPM 2012). Springer, Berlin, Heidelberg, Lecture Notes in Business Information Processing, Vol. 99, 2012, pp. 169–194, doi: 10.1007/978-3-642-28108-2_19.

[3] Li, C.—Reichert, M.—Wombacher, A.: Mining Business Process Variants: Challenges, Scenarios, Algorithms. Data and Knowledge Engineering, Vol. 70, 2011, No. 5, pp. 409–434, doi: 10.1016/j.datak.2011.01.005.

[4] van der Aalst, W. M. P.—Weijters, A. J. M. M.—Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering, Vol. 16, 2004, No. 9, pp. 1128–1142, doi: 10.1109/TKDE.2004.47.

[5] Weber, P.—Bordbar, B.—Tino, P.: A Framework for the Analysis of Process Mining Algorithms. IEEE Transactions on Systems Man and Cybernetics: Systems, Vol. 43, 2013, No. 2, pp. 303–317, doi: 10.1109/TSMCA.2012.2195169.

[6] van der Aalst, W. M. P.—Stahl, C.: Modeling Business Processes: A Petri Net Oriented Approach. The MIT Press, Cambridge, USA, 2011, doi: 10.7551/mitpress/8811.001.0001.

[7] Rozinat, A.—van der Aalst, W. M. P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems, Vol. 33, 2008, No. 1, pp. 64–95, doi: 10.1016/j.is.2007.07.001.

[8] Bose, R. P. J. C.—van der Aalst, W. M. P.: Process Diagnostics Using Trace Alignment: Opportunities, Issues, and Challenges. Information Systems, Vol. 37, 2012, No. 2, pp. 117–141, doi: 10.1016/j.is.2011.08.003.

[9] van der Aalst, W. M. P.—Adriansyah, A.—van Dongen, B. F.: Replaying History on Process Models for Conformance Checking and Performance Analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, Vol. 2, 2012, No. 2, pp. 182–192, doi: 10.1002/widm.1045.

[10] Wang, Y. Y.—Du, Y. Y.: Conformance Checking Based on Extended Footprint Matrix. Journal of Shandong University of Science and Technology (Natural Science), Vol. 37, 2018, No. 2, pp. 9–15 (in Chinese).

[11] Adriansyah, A.—van Dongen, B. F.—van der Aalst, W. M. P.: Towards Robust Conformance Checking. In: zur Muehlen, M., Su, J. (Eds.): Business Process Management Workshops (BPM 2010). Springer, Berlin, Heidelberg, Lecture Notes in Business Information Processing, Vol. 66, 2010, pp. 122–133, doi: 10.1007/978-3-642-20511-8_11.

[12] Rozinat, A.: Process Mining: Conformance and Extension. Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2010.

[13] de Leoni, M.—Maggi, F. M.—van der Aalst W. M. P.: Aligning Event Logs and Declarative Process Models for Conformance Checking. In: Barros, A., Gal, A., Kindler, E. (Eds.): Business Process Management (BPM 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7481, 2012, pp. 82–97, doi: 10.1007/978-3-642-32885-5_6.

[14] Adriansyah, A.—van Dongen, B. F.—van der Aalst, W. M. P.: Conformance Checking Using Cost-Based Fitness Analysis. Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference (EDOC'11), Helsinki, Finland, 2011, pp. 55–64, doi: 10.1109/EDOC.2011.12.

[15] Adriansyah, A.—van Dongen, B. F.—van der Aalst, W. M. P.: Memory-Efficient Alignment of Observed and Modeled Behavior. Technical report. BPMcenter.org, BPM Reports, Vol. 1303, 2013.

[16] van Eck, M. L.: Alignment-Based Process Model Repair and Its Application to the Evolutionary Tree Miner. Master Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2013.

[17] Fahland, D.—van der Aalst, W. M. P.: Model Repair – Aligning Process Models to Reality. Information Systems, Vol. 47, 2015, No. 1, pp. 220–243, doi: 10.1016/j.is.2013.12.007.

[18] Adriansyah, A.—MunozGama, J.—Carmona, J.—van Dongen, B. F.—van der Aalst, W. M. P.: Alignment Based Precision Checking. In: La Rosa, M., Soffer, P. (Eds.): Business Process Management Workshops (BPM 2012). Springer, Berlin, Heidelberg, Lecture Notes in Business Information Processing, Vol. 132, 2013, pp. 137–149, doi: 10.1007/978-3-642-36285-9_15.

[19] de Leoni, M.—Maggi, F. M.—van der Aalst, W. M. P.: An Alignment-Based Framework to Check the Conformance of Declarative Process Models and to Preprocess Event-Log Data. Information Systems, Vol. 47, 2015, No. 3, pp. 258–277, doi: 10.1016/j.is.2013.12.005.

[20] van Eck, M. L.—Buijs, J. C. A. M.—van Dongen, B. F.: Genetic Process Mining: Alignment-Based Process Model Mutation. In: Fournier, F., Mendling, J. (Eds.): Business Process Management Workshops (BPM 2014). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 202, 2014, pp. 291–303, doi: 10.1007/978-3-319-15895-2_25.

[21] Cook, J. E.—Wolf, A. L.: Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 8, 1999, No. 2, pp. 147–176, doi: 10.1145/304399.304401.

[22] Lu, X.—Fahland, D.—van der Aalst, W. M. P.: Conformance Checking Based on Partially Ordered Event Data. In: Fournier, F., Mendling, J. (Eds.): Business Process Management Workshops (BPM 2014). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 202, 2014, pp. 75–88, doi: 10.1007/978-3-319-15895-2_7.

[23] Wang, L.—Du, Y. Y.—Liu, W.: Aligning Observed and Modelled Behaviour Based on Workflow Decomposition. Enterprise Information Systems, Vol. 11, 2017, No. 8, pp. 1207–1227, doi: 10.1080/17517575.2016.1193633.

[24] Song, W.—Xia, X.—Jacobsen, H.-A.—Zhang, P.—Hu, H.: Efficient Alignment Between Event Logs and Process Models. IEEE Transactions on Services Computing, Vol. 10, 2017, No. 1, pp. 136–149, doi: 10.1109/TSC.2016.2601094.

[25] Tian, Y. H.—Du, Y. Y.—Li, M. Z.—Dong, H.—Hu, Q.: Reduced Alignment Based on Petri Nets. Concurrency and Computation: Practice and Experience, Vol. 30, 2018, No. 23, Art. No. e4411, doi: 10.1002/cpe.4411.

[26] van der Aalst, W. M. P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin, Heidelberg, 2011, doi: 10.1007/978-3-642-19345-3.

[27] Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceeding of the IEEE, Vol. 77, 1989, No. 4, pp. 541–580, doi: 10.1109/5.24143.

[28] Du, Y. Y.—Jiang, C. J.—Zhou, M. C.: A Petri Net-Based Model for Verification of Obligations and Accountability in Cooperative Systems. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, Vol. 39, 2009, No. 2, pp. 299–308, doi: 10.1109/TSMCA.2008.2010751.

[29] Ran, N.—Su, H. Y.—Wang, S. G.: An Improved Approach to Test Diagnosability of Bounded Petri Nets. IEEE/CAA Journal of Automatica Sinica, Vol. 4, 2017, No. 2, pp. 297–303, doi: 10.1109/JAS.2017.7510406.

[30] Hu, Q.—Liu, M. H.—Zhao, Z.—Du, J. W.: A Path Detecting Method to Analyze the Interactive Compatibility of Service Processes Based on WS-BPEL. Concurrency and Computation: Practice and Experience, Vol. 30, 2018, No. 19, Art. No. e4699, doi: 10.1002/cpe.4699.

[31] Liu, G. J.: Complexity of the Deadlock Problem for Petri Nets Modeling Resource Allocation Systems. Information Sciences, Vol. 363, 2016, pp. 190–197, doi: 10.1016/j.ins.2015.11.025.

[32] Zhang, W. W.—Liu, W. D.: Research of Inclined Shaft Monitoring and Control System of Coal Mine Based on Petri Net. Computer Engineering and Application, Vol. 48, 2012, No. 20, pp. 240–243 (in Chinese).

Xiwen Feng received his Ph.D. degree from Shandong University of Science and Technology, Qingdao, China. He is currently Professor at the College of Mining and Safety, Shandong University of Science and Technology, Qingdao, China. He has long been engaged in mining system engineering, industrial engineering and logistics, logistics system simulation and optimization of teaching and research work. He presided over 30 vertical and horizontal research topics such as provincial and ministerial-level key planning projects, natural science fund projects, and corporate commissions, of which 8 results reached the international advanced level and won 10 provincial and ministerial science and technology awards. He has published 60 scientific and technological papers and 1 monograph.

Dong Han received his B.Sc. degree from Weifang College, Weifang, China, in 2004. He received his M.Sc. degree from Shandong University of Science and Technology, Qingdao, in 2007, where he is currently pursuing the Ph.D. degree with College of Mining and Safety. His current research interests include resource management, Petri nets, and workflow. He is currently Lecturer of computer science and technology with Shandong University of Science and Technology.

Yinhua Tian received her B.Sc. degree in computer science and technology, the M.Sc. degree and the Ph.D. degree in computer software and theory from Shandong University of Science and Technology, Qingdao, China, in 2004, 2007 and 2018, respectively. She is currently Lecturer of computer science and technology with Shandong University of Science and Technology. She has authored over 10 technical papers in journals and conference proceedings. Her current research interests include Petri nets, process mining, and optimization algorithms.