2011 15th IEEE International Enterprise Distributed Object Computing Conference

Conformance Checking using Cost-Based Fitness Analysis

A. Adriansyah, B.F. van Dongen, W.M.P. van der Aalst
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
Email: {a.adriansyah,b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl

***Abstract*—The growing complexity of processes in many organizations stimulates the adoption of business process analysis techniques. Typically, such techniques are based on process models and assume that the operational processes in reality conform to these models. However, experience shows that reality often deviates from hand-made models. Therefore, the problem of checking to what extent the operational process conforms to the process model is important for process management, process improvement, and compliance.**

In this paper, we present a robust replay analysis technique that is able to measure the conformance of an event log for a given process model. The approach quantifies conformance and provides intuitive diagnostics (skipped and inserted activities).

Our technique has been implemented in the ProM 6 framework. Comparative evaluations show that the approach overcomes many of the limitations of existing conformance checking techniques.

***Keywords*-Business process management, fitness analysis, conformance checking, process mining, business process analysis**

I. Introduction

As business processes become more complex and change frequently, reliable process models become more important. Such models are used to document business processes or to configure the information system. Moreover, process models are used to analyze processes, e.g., to check conformance or to evaluate the performance of business process redesigns.

However, many studies show that models often deviate from reality (cf. [11]–[13]). Process models can be obsolete, outdated, idealized, or simply disconnected from reality [15]. Hence, before any sort of analysis is applied to process models, it is imperative to know how well reality conforms to the model and vice versa. Legislation such as the Sarbanes-Oxley Act, Basel II, and HIPAA, illustrate the importance of a good alignment between the real process and its model. Moreover, many organizations seek a balance between flexibility (people can deviate from the standard way of working) and control (deviations need to be monitored and acted upon if needed). Therefore, the importance of *conformance checking* is increasing.

Conformance checking measures how "good" a model of a process is with respect to an event log that records the executions of the process. In this paper, we focus on the *fitness* dimension of conformance [13], [17]. Fitness measures the extent process models capture the observed behavior as recorded in event logs. Given a process model and a sequence of activities from a log showing the execution of a process instance, the fitness of a trace is high (i.e. good) if the same sequence of activities (or a very similar one) is allowed by the model.

Event logs may not necessarily contain all the activities executed– logging everything might be costly and affect the performance of process executions. Still, the unlogged activities can influence process behavior. Identifying such *unobservable activities* in event logs is important in measuring fitness, as mistakes in doing so may lead to false-negative results.

Given a process model and an event log, deviations in the fitness dimension manifest as either *skipped* or *inserted* activities. *Skipped* activities refer to activities that should be performed according to the model, but do not occur in the log. In contrast, *inserted* activities refer to activities that occur in the log, but should not happen according to the model.

In reality, the severity of skipping/inserting activities may depend on characteristics of the activity, e.g., some activities may be skipped without severe problems while the insertion of an important activity may lead to significant problems. Take for example a typical process of handling insurance claims in an insurance company, shown as a Petri net in Figure 1. In cases where the amount of claim is relatively small, "check documents" or "check cause" activities are often skipped. Nevertheless, the severity of skipping these activities is less than the severity of skipping essential activities, such as "send money". Another example; in cases where the claimed amount is large, double checking on the

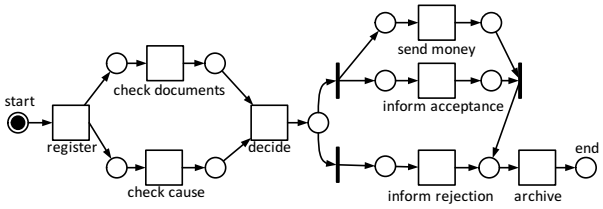

Figure 1. A Petri net describing an insurance claim handling process

1541-7719/11 $26.00 © 2011 IEEE
DOI 10.1109/EDOC.2011.12

IEEE computer society

Authorized licensed use limited to: Wuhan University. Downloaded on August 31,2022 at 08:27:32 UTC from IEEE Xplore. Restrictions apply.

claims is often performed, which leads to inserted executions of "check documents" or "check cause" not possible in the model. Although multiple executions of the two activities are also deviations, their severity to the overall process is small compared to multiple executions of "send money".

Classical techniques that measure fitness (e.g. [13]) penalize conformance for existence of either skipped or inserted activities. However, heuristics often result in incorrect estimations of fitness. Moreover, they do not consider the different severities of skipping/inserting the different activities. As a result, the fitness does not correspond to the perceived degree of conformance.

In this paper, given a process model and an event log, we propose a *cost-based replay technique* that measure fitness and taking into account the cost of skipping and inserting individual activities. The technique is based on the A^* algorithm and can be tailored to answer specific questions (e.g. Does the log conforms to the model? Which activities are often skipped? Are there any inserted activities in the log?). Section II introduces the basic concepts needed to understand this paper. The idea to measure fitness based on skipped and inserted activities is provided in Section III. We propose an A^*-based approach to measure the fitness in Section IV. Experimental results are discussed in Section VI, and related work is discussed in Section VII. Section VIII concludes the paper. For proofs of the theorems presented in this paper, as well as a more detailed analysis of the experiments, we refer to [2].

II. Basic Concepts

Our fitness calculation uses the A^* algorithm [6], an algorithm originally invented to find a shortest path between two nodes in a directed graph with arc costs. Hence, we formalize a graph with arc costs and related concepts.

$\mathbb{N}$ denotes the set of *natural numbers*. We write $\mathcal{A}$ for the *universe of activity names*, $A \subseteq \mathcal{A}$ for a set of activities, and $\tau \notin \mathcal{A}$ for unobservable activities (i.e., activities in the model not recorded in the event log).

Let T be a set. For (finite) *sequences* of elements over a set T we use the following notation: The empty sequence is denoted with ϵ; a non-empty sequence is given by listing its elements between angled brackets. A concatenation of sequences σ_1 and σ_2 is denoted with $\sigma_1 \cdot \sigma_2$ and we use $<$ to denote shorter sequences, i.e. $\sigma_1 < \sigma_2$ if and only if there is a sequence $\sigma_3 \neq \epsilon$ with $\sigma_2 = \sigma_1 \cdot \sigma_3$. T^* denotes the set of all finite sequences over T and $T^+ = T^* \setminus \{\epsilon\}$. We refer to the i-th element of a sequence σ as σ_i and we use $|\sigma|$ to represent the length of the sequence σ. The projection of a sequence $\sigma \in T^*$ on $Q \subseteq T$ is denoted as $\sigma_{\downarrow Q}$, e.g., $\langle a, a, b, c\rangle_{\downarrow\{a,c\}} = \langle a, a, c\rangle$. We say that σ is a *prefix* of σ' if and only if $\sigma < \sigma'$.

A *bag* m over P is a mapping $m : P \to \mathbb{N}$. We use $+$ and $-$ for the sum and the difference of two bags and $=, <, >, \leq, \geq$ for comparison of bags, which are defined in the standard way. We overload the set notation, writing $\emptyset$ for the empty bag and $\in$ for the element inclusion. We write e.g. $m = 2[p] + [q]$ for a bag m with $m(p) = 2$, $m(q) = 1$, and $m(x) = 0$ for all $x \notin \{p, q\}$. As usual, $|m|$ stands for the total number of elements in bag m (e.g. $|2[p] + [q]| = 3$).

Definition II.1. (Directed graph with arc cost) A directed graph with arc costs is a tuple $G = (V, W, \zeta)$ where V is a set of nodes, $W \subseteq V \times V$ is a set of directed arcs, and $\zeta : W \to \mathbb{N}$ is a function assigning non-negative costs to arcs.

Let $v \in V$, we denote the set of *successor nodes* of node v as $v \overset{G}{\bullet} = \{v' \in V \mid (v, v') \in W\}$ and the *predecessor nodes* of node v as $\overset{G}{\bullet} v = \{v' \in V \mid (v', v) \in W\}$. We omit the superscript G if the context is clear.

A *path* from v to $v' \in V$ is a sequence of edges $\langle (v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\rangle \in W^+$ where $v_1 = v \wedge v_n = v'$ holds. By $v \overset{G}{\rightsquigarrow} v'$ we denote that a path from v to v' exists (we omit the superscript G if the context is clear). We say that G is an *acyclic graph* if $\forall_{v \in V}\ v \not\rightsquigarrow v$ holds. The set of all possible paths from v to v' is denoted by $\rhd(v, v') \subseteq W^*$. We also define a *path cost function* $\zeta_\rhd : W^* \to \mathbb{N}$ that returns the cost of a path where $\zeta_\rhd(\sigma) = \sum_{i=1}^{|\sigma|} \zeta(\sigma_i)$.

Definition II.2. (The A* algorithm) Let $G = (V, W, \zeta)$ be a directed graph with distances associated to arcs. Let $v_{src} \in V$ be a *source* node and let $V_{trg} \subseteq (V \setminus \{v_{src}\})$ be a set of *target* nodes in G. The A^* *algorithm* returns a path $pa \in \rhd(v_{src}, v_{trg})$ with the smallest distance from the source node to one of the target nodes where $v_{trg} \in V_{trg}$. Hence, $\forall_{v_{trg} \in V_{trg}}\ \nexists_{pa' \in (\rhd(v_{src}, v_{trg}) \setminus \{pa\})}\ \zeta_\rhd(pa') < \zeta_\rhd(pa)$ holds. The target node v_{trg} is also called the *preferred* target node.

The algorithm works by iteratively exploring successors of nodes, starting from the source node v_{src}. Let $v \in V$ be a node in G and let $pa \in \rhd(v_{src}, v)$ be the path with the smallest distance so far from v_{src} to v, the algorithm relies on an *evaluation* function $f(v) = g(v) + h(v)$, where $g : V \to \mathbb{N}$ is a function that returns the smallest distance of paths from the source node v_{src} to v (so far), and $h : V \to \mathbb{N}$ is a heuristic function that *underestimates* the distance of path from any node to its preferred target node $v_{trg} \in V_{trg}$.

To determine which visited node whose successors are going to be explored in the next iteration, the algorithm calculates the distance of every node whose successors haven't been explored yet, using an evaluation function $f : V \to \mathbb{N}$. Then, it selects the one that has the minimal distance. The iteration stops under two conditions: either a node that is a member of V_{trg} is selected as the node to be explored in the next iteration (implies that we get the solution path), or there is no other nodes to be explored (implies that no path to any target nodes exists).

As long as the heuristic function h returns a value that underestimates the distance of a path from a node

Authorized licensed use limited to: Wuhan University. Downloaded on August 31,2022 at 08:27:32 UTC from IEEE Xplore. Restrictions apply.

to its preferred target node and the evaluation function is increasing with the increasing number of visited nodes, the A^* algorithm is guaranteed to find a path with the smallest distance [6].

Measuring fitness requires an event log and a process model. We formalize event logs, process models, and related concepts as follows:

Definition II.3. (Event logs) An event log over a set of activities A is defined as $L_A = (E, C, \alpha, \beta, \succ)$, where:

- E is a finite set of events,
- C is a finite set of cases (process instances),
- $\alpha : E \rightarrow A$ is a function relating each event to an activity,
- $\beta : E \rightarrow C$ is a surjective function relating each event to a case.
- $\succ \subseteq E \times E$ imposes a total ordering on the events in E. We write $e_2 \succ e_1$ as a shorthand to $(e_2, e_1) \in \succ$.

In reality, cases are executed independently from each other. For example, in an insurance company, the way a claim is handled does not directly influence how other claims are handled. Therefore, events of a case are often treated independently from events of other cases. Let $c \in C$ be a case identifier. With E_c, we denote the *events of case* c, i.e. $E_c = \langle e_1, \ldots, e_{|E_c|} \rangle$ where $\forall_{e \in E}(\beta(e) = c) \Leftrightarrow (e \in E_c)$, furthermore $\forall_{1 \leq i < j \leq |E_c|}\ e_i \succ e_j$.

A process model typically describes a set of activities that have to be performed and their ordering. Many languages are used to model business process, like EPC and BPMN[1]. In this paper, we use Petri nets [10] to model processes, however, our approach is applicable to any kind of model, as long as it can be decided if a given execution is a valid one (there is no need to be able to decide which activities can occur in the future, only if a given activity could have occurred at a given point in time).

Definition II.4. (Petri net) A Petri net N over a set of activities A is a tuple $N = (P, T, F, \pi)$, where P and T is a set of places and transitions, respectively, $F \subseteq (T \times P) \cup (P \times T)$ is a set of directed arcs connecting places and transitions, $\pi : T \rightarrow A \cup \{\tau\}$ is a function mapping transitions to either activities or τ (unobservable activities), such that $\forall_{a \in A} \exists_{t \in T}\ \pi(t) = a$. Note that any net N is also a directed graph $(P \cup T, F, \zeta)$ where $\forall_{fw \in F}\ \zeta(fw) = 1$.

A marking m of N is a bag over P, indicating the state of N. A transition $t \in T$ can be executed (i.e. fired) from marking m if and only if $m \geq \bullet t$. Firing transition t resulted in a new marking m' where $m' \stackrel{\text{def}}{=} m - \bullet t + t\bullet$. We denote this relation between m and m' as $m \xrightarrow{t} m'$. A set of *runs* of N from marking m_0 is a set of all sequences of transitions $\mathfrak{R} \subseteq T^*$ where for all $\langle t_1, \ldots, t_n \rangle \in \mathfrak{R}$, $m_0 \xrightarrow{t_1} m_1, m_1 \xrightarrow{t_2} m_2, \ldots, m_{n-1} \xrightarrow{t_n} m_n$ hold.

[1]Business Process Modeling Notation, see http://www.bpmn.org/

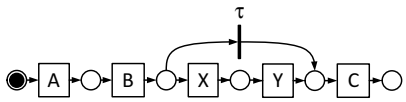

Figure 2. a Petri net for running example

III. Ideal cost-based replay

Given an event log $L_A = (E, C, \alpha, \beta, \succ)$ and a Petri net $N = (P, T, F, \pi)$ with initial marking m_0, events of case $c \in C$ fit the net if the sequence of activities that is constructed by mapping each event of the case c to an activity can also be constructed by mapping each transition of a run of the net to an activity. We define formally a fitting set of events of a case as follows:

Definition III.1. (Perfect fit) Let $N = (P, T, F, \pi)$ be a Petri net over a set of activities A with an initial marking m_0. Let $\mathfrak{R} \subseteq T^*$ be the set of all possible runs of N starting in m_0. Let $L_A = (E, C, \alpha, \beta, \succ)$ be an event log over A. Let $c \in C$ be a case and let E_c be the sequence of events of case c.

E_c *fits* N if there exists a run $\sigma \in \mathfrak{R}$ such that $\pi(\sigma)_{\downarrow A} = \alpha(E_c)$, where we lift the use of α and π to sequences[2].

Take for an example a Petri net in Fig. 2. Both cases $E_c^1 = \langle A, B, C \rangle$ and $E_c^2 = \langle A, B, X, Y, C \rangle$ perfectly fit the net according to Def. III.1. The case $E_c^3 = \langle A, B, X, C \rangle$, does not fit to the model, since any run in the model will contain Y after X.

Unfortunately, in practice, knowing whether a case is deviating or not is not useful for further analysis. In cases where deviations occur, such as the case E_c^3, it is more important to know *the extent* of the deviations, and furthermore *why* they occur.

To measure the extent of deviations, fitness is best measured on a per-event basis. From Section I, we know that there are two possible causes of deviations: *skipping* or *inserting* activities. Therefore, the ideal fitness calculation should penalize fitness value based on the existence of each of these two types of deviations. However, identifying such activities is not trivial. Consider the net in Fig. 2 and the case E_c^3. The case can be interpreted as one of the following: X is inserted in the event log, Y is skipped in the model, or C is inserted in the log (removing results in a "good" prefix).

As mentioned in Section I, the cost of skipping and inserting activities can be different for individual activities. Therefore, we assume the existence of cost functions that return the cost of skipping as well as inserting activities. We define cost function $\kappa^i : A \rightarrow \mathbb{N}$ that returns non-negative cost of inserting extra activities in the log, and another cost

[2]Let $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ be a sequence over some set S and let $f : S \rightarrow R$ be a function from S to some set R, then $f(\sigma) = \langle f(\sigma_1), \ldots, f(\sigma_n) \rangle$.

Authorized licensed use limited to: Wuhan University. Downloaded on August 31,2022 at 08:27:32 UTC from IEEE Xplore. Restrictions apply.

function $\kappa^s : A \cup \{\tau\} \rightarrow \mathbb{N}$ that returns non-negative cost of skipping transitions in the model (including the unlabeled ones). These functions need to be determined by process experts.

When identifying the deviations between cases in the log and processes, we assume that we are interested in the actual execution of which the costs of deviation are minimal. Thus, suppose that the cost functions for activities involved in the net in Fig. 2 are defined as follows: $\kappa^i(A) = \kappa^i(B) = \kappa^i(Y) = 1, \kappa^i(C) = 10, \kappa^i(X) = 5, \kappa^s(A) = \kappa^s(B) = \kappa^s(X) = 1, \kappa^s(Y) = 3, \kappa^s(C) = 10, \kappa^s(\tau) = 0$. For case E_c^3, the cost of either inserting X or inserting C is higher than the cost of skipping Y (i.e. $\kappa^i(X) > \kappa^s(Y)$ and $\kappa^i(C) > \kappa^s(Y)$), hence the deviation in the case most likely occurred because activity Y was skipped (hence the case should have been $\langle A, B, X, Y, C \rangle$).

Considering cost functions and both skipped and inserted activities, we define our fitness metric as follows:

Definition III.2. (Cost-based fitness metric) Let $N = (P, T, F, \pi)$ be a Petri net over a set of activities A with an initial marking m_0. Let $\mathfrak{R} \subseteq T^*$ be the set of all possible runs of N starting in m_0. Let $L_A = (E, C, \alpha, \beta, \succ)$ be an event log over A. Let $c \in C$ be a case and let E_c be the sequence of events of case c. Let κ^s and κ^i be the cost functions for skipping and inserting activities respectively.

Assuming that skipped activities in the case are identified as a bag A_s over $A \cup \{\tau\}$, and all inserted activities, manifested as events, are identified as a set $E_i \subseteq E_c$, we define a fitness f as one minus the ratio between the total cost of having inserted/skipped activities and the total cost of considering all events as inserted activities, i.e.

$$f = 1 - \frac{\sum_{a \in A_s} A_s(a) \times \kappa^s(a) + \sum_{e \in E_i} \kappa^i(\alpha(e))}{\sum_{e \in E_c} \kappa^i(\alpha(e))}$$

The intuition behind the metric is that fitness value should decrease as more activities are inserted/skipped. In the worst case, given a process model and a set of events of a case, all of the events can be considered as inserted activities. The cost of such extreme case is used to normalize our fitness metric. Note that due to the absence of information about skipped activities in the log, one can assume as many skipped activities as allowed by the model. In case there is a large number of skipped activities, (e.g., in a loop executed repeatedly) f may become negative. However, the "best run" has a value between 0 and 1.

The cost functions κ^s and κ^i defined above are an important contribution of this paper. By providing the user with options to change these parameters, the user can specify what he thinks is more problematic. For example, by giving relatively low costs to skipping activities, the user can specify that an activity that should be executed in the model but cannot be found in the log should be considered as being executed anyway, while a high value indicates that this transition indeed did not happen.

The fitness metric defined in Def. III.2 assumes that skipped and inserted activities are known in advance. In this paper, we are interested in finding inserted/skipped activities that give minimal cost such that the highest possible fitness value is obtained. We describe our approach to identify such activities in Section IV.

IV. Identify Skipped and Inserted Activities

A case in an event log fits a Petri net if each events performed in the case can be mimicked by firing a transition in the net that refers to the same activity as the event, either directly from the current state of the net or indirectly after firing sequence of τ-labeled transitions from the current state. Skipped activities should only be introduced when according to the model, a transition that should be able to mimic an event cannot be fired without firing non-τ-labeled transitions. Only in cases where the cost of skipping activities is higher than the cost to assume that an event under consideration is inserted, we consider the event to represent an inserted activity.

Therefore, finding both skipped and inserted activities in a given case can be formulated as a problem of constructing the best *matching instance* of a given net based on the events of the case. A set of events of a case fits a Petri net (i.e. all events can be generated by the net) if and only if there is an instance of the net matching the events in which each non-τ transition instance represents an event and partial order between events is honored by transition instance. We use the standard definition of an *instance* of Petri net (sometimes referred to as *occurrence net* [9]) since a run of a net does not capture causal dependencies and concurrencies. This is a standard concept in Petri nets, therefore we simply use it without explaining it in detail.

Definition IV.1. (Instance of Petri net) Let $N = (P, T, F, \pi)$ be a Petri net over a set of activities A with an initial marking m_0. Let $I = (PI, TI, FI, \rho, \varrho, m_0')$ be a tuple where PI is a set of place instances, TI is a set of transition instances, $FI \subseteq (PI \times TI) \cup (TI \times PI)$ is a set of edge instances, $\rho : PI \rightarrow P$ is a function mapping place instances to places in P, $\varrho : TI \rightarrow T$ is a function mapping

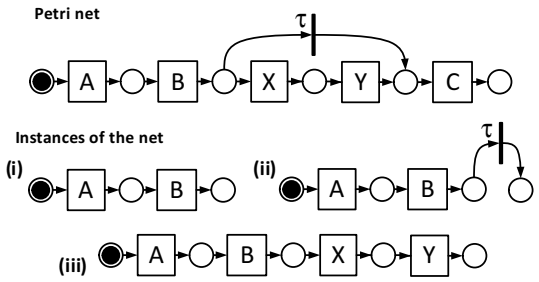

Figure 3. Examples of instances of a given Petri net

Authorized licensed use limited to: Wuhan University. Downloaded on August 31,2022 at 08:27:32 UTC from IEEE Xplore. Restrictions apply.

transition instances to transitions in T, and m'_0 is a bag of PI that indicates the initial marking of I.

I is an *instance* of N if and only if the following holds:

- For all $(x, y) \in FI$, if $x \in PI$ then $(\rho(x), \varrho(y)) \in F$, else $(\varrho(x), \rho(y)) \in F$,
- For all $ti \in TI$, ρ induces a bijection from $\bullet ti$ to $\bullet \varrho(ti)$ and from $ti\bullet$ to $\varrho(ti)\bullet$
- $\forall_{pi \in PI}\ m'_0(pi) \leq 1 \wedge |\bullet pi| \leq 1 \wedge |pi \bullet| \leq 1$
- For all $pi \in PI$, $\bullet pi = \emptyset$ if and only if $m'_0(pi) = 1$,
- $\forall_{p \in P}\ \sum_{pi \in PI, \rho(pi)=p} m'_0(pi) = m_0(p)$
- The transitive closure of FI is irreflexive, i.e. it is a partial order over $PI \cup TI$

Note that when we consider Petri net instances, we generally consider their equivalence classes, i.e. two instances are equivalent if their exists a graph isomorphism that respects the mapping functions. Furthermore, we use $\downarrow$ to denote the projection of an instance onto a subset of its transitions, i.e. $I \downarrow TI'$ with $TI' \subseteq TI$ is the same Petri net as I, but without the transition instances not in TI', without the arcs connected to these transitions and without any disconnected places.

Figure 3 shows some possible instances of the Petri net given in Fig. 2. As shown in Figure 3, there are no conflicts/choices in an instance. Places and transitions need to be on a path starting on one of the initially marked places.

When a set of events is replayed on a Petri net, we iteratively construct instances from its prefixes. For each pair consisting of a prefix and a constructed instance, we match events in the prefix (possibly partial) to transition instances that refer to the same activities. Matched events correspond to firings of transitions. Events in the prefix without any match indicate *inserted activities* because they happened in reality, but should not happen according to the net. Non-τ-labeled transition instances in the Petri net instance that are not associated with any events represent *skipped activities* as they should be performed according to the net, but were not performed in reality.

Given a prefix of a case, an instance of a Petri net with a function matching events in the prefix to transition instances is said to be *matching* the prefix.

Definition IV.2. (Instance matching a prefix) Let $L_A = (E, C, \alpha, \beta, \succ)$ be an event log over a set of activities A. Let $c \in C$ be a case identifier. Let E_c be the sequence of events of case c. Let E' be a prefix of E_c. Let $N = (P, T, F, \pi)$ be a Petri net over A with initial marking m_0, and let $I = (PI, TI, FI, \rho, \varrho, m'_0)$ be an instance of N. Let $\mu : E' \nrightarrow TI$ be a partial function mapping events to transition instances such that μ induces a bijection from its domain ($\mathrm{Dom}(\mu) \subseteq E'$) to its range ($\mathrm{Rng}(\mu) \subseteq TI$). We say I matches E' with match μ if and only if:

1) $\forall_{e_1, e_2 \in \mathrm{Dom}(\mu)}\ \mu(e_1) \overset{I}{\rightsquigarrow} \mu(e_2) \Rightarrow e_1 \succ e_2$, i.e. the ordering of events is in the prefix is respected in the instance, and

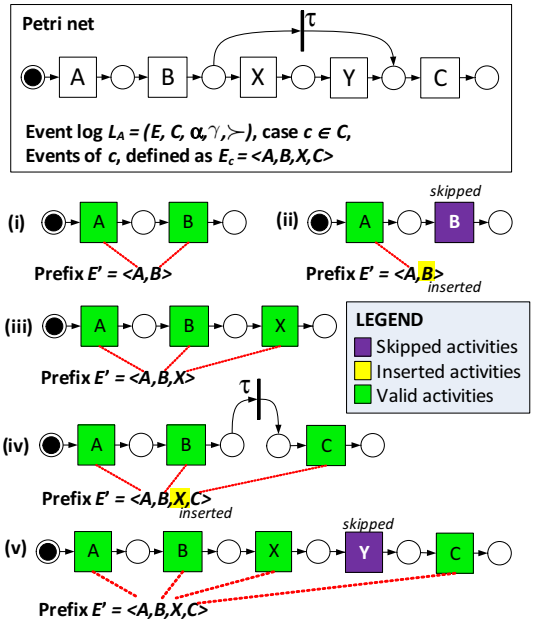

Figure 4. Example of matching instances, given a Petri net and a set of events of a case

2) $\forall_{e \in \mathrm{Dom}(\mu)}\ \varrho(\mu(e)) \in \{t \in T \mid \pi(t) = \alpha(e)\}$, i.e. each event mapped by μ in the prefix is mapped to a transition that corresponds to the activity represented by this event.

We use $(I_{E'}, \mu)$ to denote a tuple consisting of an arbitrary instance I that matches prefix E' with match μ, and we use $\Im_{E'}$ to denote the (possibly infinite) set of all tuples that consists of an instance matching prefix E' and its match (i.e. $(I_{E'}, \mu) \in \Im_{E'}$). μ is a partial function. For convenience we write $\mu(e) = \perp$ to indicate that e is not in the domain of μ.

Take for example several instances of a Petri net that match a set of events of a case in Fig. 4, each with its own matching function μ that (partially) maps the events to transition instances. Transition instances that have events mapped to them are shaded. As shown by matching instances (i) and (ii), matching functions may partially map events to transition instances. In instance (i), both events in the prefix $\langle A, B \rangle$ are mapped to the transition instances A and B. In instance (ii) however, the second event is not mapped to the second transition instance, i.e. the event B is identified as being an inserted activity and the transition instance B is identified as a skipped activity.

Matching instances may reveal different reasons for deviations (if there are any). Matching instance (iii) shows no deviations for the current prefix, while in instance (iv), the whole case is considered in which event X is identified as being an inserted activity. In instance (v), again the whole case is considered, but now the transition instance Y is skipped, while X is no longer an inserted activity.

Authorized licensed use limited to: Wuhan University. Downloaded on August 31,2022 at 08:27:32 UTC from IEEE Xplore. Restrictions apply.

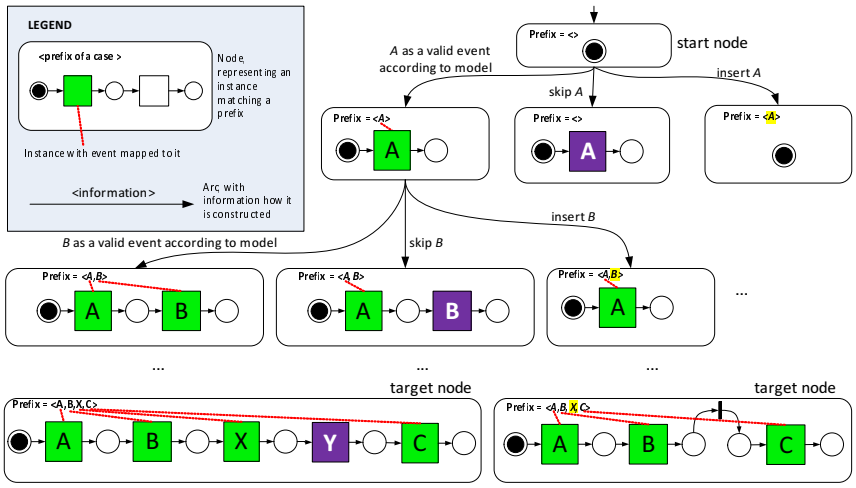

Figure 5. Example of a search by the A^* algorithm

To determine which of the instances is most likely describing the occurring deviations in a given case, we utilize the cost functions of skipping activities κ^s and inserting extra activities κ^i. The idea is that we construct an instance that has the least cost of deviations. For example, if the cost of skipping activity Y (i.e. $\kappa^s(Y)$) is less than inserting activity X (i.e. $\kappa^i(S)$), then instance (v) is the best fitting instance, otherwise instance (iv) is.

So far, we have defined the notion of fitness and we have shown both how to quantify fitness and how to locate deviations if we have an instance matching a prefix. In the next section, we present our A^* based algorithm for constructing the best matching instance.

V. Constructing the Best Matching Instance

Since our aim is to construct a matching instance for any given case, we first prove that such an instance always exists, not only for every instance, but also for every prefix of an instance.

Lemma V.1. (Matching instance exists for any prefix) Let $L_A = (E, C, \alpha, \beta, \succ)$ be an event log over set of activities A. Let $c \in C$ be a case identifier, let E_c be the sequence of events of case c, and let $E' \subseteq E_c$ be a prefix. Let $N = (P, T, F, \pi)$ be a Petri net over A with initial marking m_0, and let $I = (PI, TI, FI, \rho, \varrho, m_0')$ be an instance of net N. We show that $(I, \mu) \in \Im_{E'}$ for any μ with $\text{Dom}(\mu) = \emptyset$.

We identify a Petri net instance that best matches our case using the A^* approach. Before we introduce the formal technique, Fig. 5 illustrates our approach roughly in using the Petri net and the set of events example in Fig. 4.

We associate a directed graph in the A^* problem domain with an acyclic search space graph to seek the best matching instance, given a set of events of a case and a net. Nodes in the search space graph represent instances of the given net matching a prefix of the case.

In the A^* problem domain, the graph structure is typically known in advance, but in our technique, it is constructed during replay. Given a case E_c to be replayed on a net, we start by constructing the search space graph consisting of only an instance matching the empty set (i.e. the instance consisting only of places that are initially marked as indicated by the start node in Figure 5).

Based on the instance, we construct other instances as successors in the graph. A successor is again a matching instance such that:

- it has one matching transition instance more than its predecessor, in which case one transition instance is added to the net and the prefix is extended with a corresponding event or
- it has one inserted activity more, i.e., the net stays the same, but the prefix contains one more event, or
- it has one skipped activity more, i.e., the net has one more transition instance but the prefix remains the same. Note that more than one transition can be enabled, hence there may be more than one successor in the graph with an added skipped activity.

The steps above are repeated until one of the nodes in the graph with the shortest distance from the start node is actually a target node, i.e. a node of which the prefix is the entire case.

The distance between two adjacent nodes depends on the difference between these nodes in the number of events mapped by the matching function and the number of unmapped non-τ-labeled transitions. Furthermore, the heuristic function h required by the A^* algorithm, is defined as the number of events in the same case that do not belong to the current prefix.

In order to obtain the distances associated to the arcs, we use the notion of costs again, where costs are defined on nodes and the distance between two nodes is the difference

Authorized licensed use limited to: Wuhan University. Downloaded on August 31,2022 at 08:27:32 UTC from IEEE Xplore. Restrictions apply.

in costs.

Definition V.2. (Cost of a matching instance) Let $L_A = (E, C, \alpha, \beta, \succ)$ be an event log over a set of activities A. Let $c \in C$ be a case identifier, and let E_c be the sequence of events of case c. Let E' be a prefix of E_c. Let $N = (P, T, F, \pi)$ be a Petri net over A with initial marking m_0, and let $(I_{E'}, \mu) \in \Im_{E'}$ be a tuple of an instance of N matching E' with match μ. $I_{E'} = (PI, TI, FI, \rho, \varrho, m_0')$. Let κ^s and κ^i be cost functions for skipping and inserting activities respectively.

We denote transition instances that are not mapped to any event as $TI_s = TI \setminus \text{Rng}(\mu)$ and denote a set of events that are not mapped to any transitions as $E_i = E' \setminus \text{Dom}(\mu)$. We define a cost function $\delta_n : \Im_{E'} \to \mathbb{N}$ where

$$\delta_n((I_{E'}, \mu)) = \sum_{ti \in TI_s} \kappa^s(\pi(\varrho(ti))) + \sum_{e \in E_i} \kappa^i(\alpha(e))$$

The cost function δ_n has a close relation with the cost-based fitness metric defined in Def. III.2. The set of skipped transition instances is the same as the bag of skipped activities, hence δ_n defines the nominator of the fraction.

Using the costs of two matching instances, we can define the distance between them as follows.

Definition V.3. (Distance between matching instances) Let $L_A = (E, C, \alpha, \beta, \succ)$ be an event log over a set of activities A. Let $c \in C$ be a case identifier, and let E_c be the sequence of events of case c. Let E', E'' be two prefixes of E_c. Let $N = (P, T, F, \pi)$ be a Petri net over A with initial marking m_0, let I and I' be two instances of N, and let $(I, \mu) \in \Im_{E'}$ and $(I', \mu') \in \Im_{E''}$ be two tuples of instances matching E' with match μ and E'' with match μ' respectively.

We define the distance between (I, μ) and (I', μ') as:

$$\delta((I, \mu), (I', \mu')) = \delta_n((I', \mu')) - \delta_n((I, \mu)) + |E''| - |E'|$$

The number of events of each prefix is also added as part of the distance definition as the value of the function δ is influenced by the number of events in each prefix. Note that it is easy to see that the function δ is transitive.

Although we defined the distance between any two matching instances, our approach only considers instances that differ either by one transition, or by one event in the prefix. Therefore, we define a partial order on matching instances as follows:

Definition V.4. (Partial order between instances) Let $L_A = (E, C, \alpha, \beta, \succ)$ be an event log over a set of activities A. Let $c \in C$ be a case identifier, and let E_c be the sequence of events of case c. Let E_1, E_2 be two prefixes of E_c. Let $N = (P, T, F, \pi)$ be a Petri net over A with initial marking m_0, let $I_1 = (PI_1, TI_1, FI_1, \rho_1, \varrho_1, m_0^1)$ and $I_2 = (PI_2, TI_2, FI_2, \rho_2, \varrho_2, m_0^2)$ be two instances of N. Let $(I_1, \mu_1) \in \Im_{E_1}$ and $(I_2, \mu_2) \in \Im_{E_2}$ be two tuples of instances matching E_1 with match μ_1 and E_2 with match μ_2 respectively.

We define a partial order between matching instances (I_1, μ_1) and (I_2, μ_2), denoted by $(I_1, \mu_1) \blacktriangleright (I_2, \mu_2)$ if and only if

- $E_2 = E_1 \cdot e$ and there exist $ti \in TI_2 \setminus TI_1$ with $TI_2 = TI_1 \cup \{ti\}$, $\mu_2(e) = ti$, $\forall_{e' \in E_1}\ \mu_1(e') = \mu_2(e')$ and $I_1 = I_2 \downarrow TI_1$, i.e. a transition that corresponds to an event is added at the end of the instance, or
- $E_2 = E_1$ and there exist $ti \in TI_2 \setminus TI_1$ with $TI_2 = TI_1 \cup \{ti\}$, $\forall_{e \in E_2}\ \mu_1(e) = \mu_2(e)$ and $I_1 = I_2 \downarrow TI_1$, i.e. one skipped activity is added to the end of the instance, or
- $E_2 = E_1 \cdot e$ and $\forall_{e \in E_2}\ \mu_1(e) = \mu_2(e)$ and $\mu_1(e) = \perp$ and $I_1 = I_2$, i.e. one inserted activity is identified at the end of the instance.

Finally, we formalize our search space and heuristic function using this partial order on matching instances.

Definition V.5. (Search space graph and heuristic function) Let $L_A = (E, C, \alpha, \beta, \succ)$ be an event log over a set of activities A and let $N = (P, T, F, \pi)$ be a Petri net over A with initial marking m_0. Let $c \in C$ be a case, and let E_c be the events of case c. Let κ^s and κ^i be cost functions for skipping and inserting activities respectively. We define a search space graph of replaying E_c on N as $G = (V, W, \zeta)$ with heuristic function $h : V \to \mathbb{N}$ (see Def. II.2) as follows:

- V is a (possibly infinite) set nodes, defined as $V = \bigcup_{E' \leq E_c} \Im_{E'}$
- $W = \{(v_1, v_2) \in V \times V \mid v_1 \blacktriangleright v_2\}$ is a set of arcs.
- $h : V \to \mathbb{N}$ is a heuristic cost function that estimates the least cost of any paths from a node to its preferred target node. Let $E' \leq E_c$ be a prefix of events and let $v \in V$. We define $h(v) = |E_c| - |E'|$, i.e. the number of events in case c still not used to construct v.
- For all $(v_1, v_2) \in W$ holds that $\zeta((v_1, v_2)) = \delta(v_1, v_2)$.

The source $v_{src} \in V$ and target nodes $V_{trg} \subseteq V$ are defined as follows:

- $v_{src} = (((\{p \in m_0\}, \emptyset, \emptyset, \rho, \varrho, m_0'), \mu)$, i.e. $\Im_{\langle\rangle} = \{v_{src}\}$, and
- $V_{trg} = \Im_{E_c}$ is a set of all instances matching E_c and their matching functions .

To guarantee that the A^* algorithm will actually find one of the target nodes from the source node, we need to prove three things: (i) at least one target node is reachable, (ii) the heuristic function gives an underestimation of the distance to the target nodes, and (iii) the evaluation function is monotonously increasing. The first is trivial, as the target node where each event is considered a inserted activity is reachable by adding the inserted activities incrementally (i.e. following part 3 of Def. V.4). Furthermore, if the heuristic function underestimates the distance to the target nodes and all distances are non-negative then due to the transitivity of the function δ the evaluation function will be monotonously increasing.

Authorized licensed use limited to: Wuhan University. Downloaded on August 31,2022 at 08:27:32 UTC from IEEE Xplore. Restrictions apply.

Lemma V.6. (Non-negative cost of arc) Let $L_A = (E, C, \alpha, \beta, \succ)$ be an event log over a set of activities A, Let $c \in C$ be a case and let E_c be the events of case c. Let $G = (V, W, \zeta)$ be a search space and let $(v_1, v_2) \in W$ be an edge in the search space. We show that $\zeta((v_1, v_2)) \geq 0$, i.e. that $\delta(v_1, v_2) \geq 0$.

Theorem V.7. (Heuristic function gives underestimation) Let $L_A = (E, C, \alpha, \beta, \succ)$ be an event log over a set of activities A, Let $c \in C$ be a case and let E_c be the events of case c. Let $G = (V, W, \delta)$ be a search space and let $v', v_{trg} \in V$ be two nodes in the search space, such that $v' \rightsquigarrow v_{trg}$ and $v_{trg} \in V_{trg}$. We show that $h(v') \leq \zeta((v', v_{trg}))$.

This theorem shows that we can use an A^*-based approach for conformance checking. It is important to realize that the search space defined in Def. V.5 is infinite in case there are loops in the Petri net. If the costs of executing such a loop is not greater than 0, then the A^* algorithm has to consider the infinite search space, hence it is not guaranteed to terminate. Therefore, each loop should have an activity with positive costs for skipping. As our approach works based on costs, the selection of cost functions κ^i and κ^s is essential. We present experiments where we compare different values for these cost functions.

VI. Experiments

We implemented the proposed fitness calculation approach as a ProM 6 plug-in [3] and performed experiments using various cost parameter values for different goals. As the costs of skipping/inserting individual activities require specific knowledge about processes under consideration, in our experiments we assume fixed costs for skipping and inserting activities, i.e., we are not using domain knowledge to limit the number of parameters in our experiments. Furthermore, the costs of skipping a τ labeled transition are assumed to be 0, since we do not have loops of such transitions in the model.

[3] see http://www.processmining.org

A. Influence of cost ratios

Given a model and cases with different number of deviations, good fitness measurements should give cases with high number of deviations less fitness value than the ones with less number of deviations. To see which cost configuration gives good fitness measurement in the presence of deviations, we conducted a set of experiments where only one of the two types of deviations occur. We use a Petri net with unobservable activities, duplicate transitions (i.e. transitions representing the same activity), and complex control-flow patterns such as the multi-choice pattern (OR-split), milestone, and iterations. To get a controlled testing environment, experiments with deviating activities are performed with relatively small number of events. The set of events required for experiments with skipped activities is obtained by generating a relatively small number of 25 events from the model and delete the events randomly as many as the desired number of skipped activities. Similarly, the set of events to conduct experiments with inserted activities is obtained from 10 events, generated from the model, and then add activities as many as the number of inserted activities needed. For each experiment with a case, we also calculate the ratio between deviating events and the number of events in the case as baseline. The results of the experiments are shown in Fig. 6 and 7.

Our fitness measurement approach produces good results in the experiments where the cost of skipping activities is higher than the cost of inserting activities (e.g. experiments with $\kappa^i(a) : \kappa^s(a)$ equal to either $1:2$ or $1:5$). With such cost settings, fitness values are close to the baseline values and tend to decrease as the number of deviations increases. However, there are few exceptions on experiments with skipped activities, where fitness increases along with the increasing number of skipped activities (see Fig. 6). These exceptions occur because our measurement approach is guaranteed to find the biggest fitness value. Rather than identifying skipped activities that penalize fitness severely (due to cost settings), marking some events as inserted

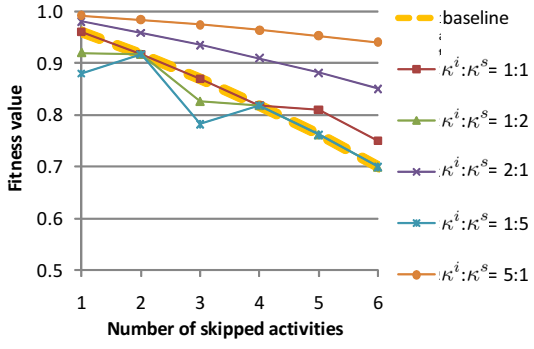

Figure 6. Experiment results showing that fitness values tend to decrease as the number of skipped activities increases

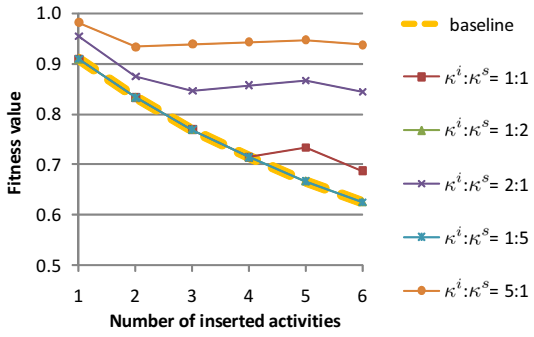

Figure 7. Experiment results showing that fitness values tend to decrease as the number of inserted activities increases

Authorized licensed use limited to: Wuhan University. Downloaded on August 31,2022 at 08:27:32 UTC from IEEE Xplore. Restrictions apply.

Table I
FITNESS VALUE FROM EXPERIMENTS ON REAL-LIFE LOGS AND MODELS

Log	#Cases	#Events	#Events per case			Process model		Fitness ($\kappa^i(a) : \kappa^s(a) = 1 : 3$)			$f_{classical}$ [13]
			min	max	avg	#places	#trans.	min	max	avg. fitness / case	
Bezwaar	65	683	6	20	10	15	27	0.00	1.00	0.76	0.86
BezwaarWOZ	1981	11278	1	17	5	12	18	0.00	1.00	0.55	0.74
Bouwverg.	714	9116	1	32	12	23	33	0.00	1.00	0.46	0.66
Afschriften	370	742	1	5	2	11	8	1.00	1.00	1.00	0.50

activities gives higher fitness in such exception cases.

Setting the costs such that inserting activities is more costly than skipping activities always give high fitness values, because it allows the approach to find skipped activities that enable unfireable transitions. The bigger the cost of skipping compare to the cost of inserting activities, the higher the fitness due to less penalty for skipping activities.

B. Real-life experiments

The second set of experiments is conducted to see whether the approach can handle real-life cases. We use four pairs of model and real-life log from a municipality in the Netherlands, describing process executions of handling two type of objections, handling building permission applications and giving out copies of documents. Experiment settings and results are shown in Table I. The results are compared to the results of the classical fitness calculation described in [13].

From our previous experiment, we know that setting the cost of inserting to be lower than the cost of skipping activities provides a better fitness metric. Therefore, we use the value 1:3 as the ratio between inserting and skipping activities to measure fitness and obtain results as shown in Table I. In most cases, our approach provides less fitness values than the ones provided by classical fitness measurements. Only in the experiments with "Afschriften" log that our fitness measurement gives higher values than the classical fitness measurement. This happens because our metric does not penalize fitness for not terminating properly. Thus, our fitness measurement is more suitable to measure fitness in cases where the completeness of process executions are not guaranteed. In addition, no performance issues were found as fitness calculations were done within seconds.

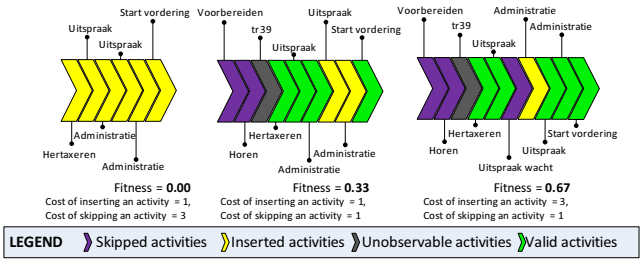

Figure 8. Analysis with different cost values to reveal the cause of deviations in a real-life case

We obtain better diagnostics for analysis of the cause of deviations than the classical fitness measurement that relies on manual analysis. For example, we take one of the cases that have extremely low fitness values (0.00) in log "BezwaarWOZ". The fitness value is provided when the cost of inserting:skipping activities is 1:3 (see Fig. 8). As shown in the left side of Fig. 8, all events in the case are identified as inserted activities. To see what causes such low fitness value, we decrease the cost of skipping an activity such that ratio between inserting:skipping activities becomes 1:1. With such ratio, two skipped activities are revealed: "Voorbereiden" and "Horen". These two activities are supposed to be executed early in the case. By increasing the cost of inserting an activity such that the ratio between the cost of inserting and skipping activities becomes 3:1, we identified one more activity that is skipped: "Uitspraak wacht". *By tuning the cost parameters such that the cost of skipping activities is less than the cost of inserting activities, one can obtain results that reveal possible causes of deviations and exploit the information for further analysis* (e.g. if an activity is often skipped, process model should also allow such skipping).

VII. RELATED WORK

The notion of conformance has appeared in many different contexts, such as business process compliance [14], auditing [16], security, and process mining. Conformance comprises of several orthogonal dimensions, such as *fitness*, *precision*, *generalization*, and *structural* [13], [17]. From all dimensions of conformance, *fitness* is one of the most important dimensions and therefore typically measured first. If fitness value between a given model and process execution is low, not much useful information can be obtained from measuring conformance on other dimensions.

Many of existing fitness metrics are created to evaluate process discovery techniques. There are already various metrics proposed related to the fitness dimension [1], [3], [5], [7], [8], [13]. A comprehensive lists of these metric are provided in [17]. Fitness is measured on different level of granularity. Our proposed metric penalize fitness only for individual execution of activity that is not fit.

We consider the work on the classical conformance checking with a metric based on the number of missing, remaining, produced, and consumed tokens, proposed by Rozinat et al. [13] as a benchmark. The approach have been tested against several real-life case studies (e.g. [11], [12]).

Authorized licensed use limited to: Wuhan University. Downloaded on August 31,2022 at 08:27:32 UTC from IEEE Xplore. Restrictions apply.

However, the proposed fitness is sensitive to the structure of the model. Furthermore, it is shown that in the presence of duplicate transitions/unobservable activities, heuristics may lead to incorrect results. Experiment results in Section VI show that our approach manage to provide correct results in the presence of duplicate transitions/unobservable activities.

Our work in this paper is closely related to the work of Cook et al. (see [3], [4]). To our knowledge, the work propose one of the most early fitness-related measurements that consider the severity of deviations based on event logs. Given a model and an execution of a process, the approach in [3] measures conformance by comparing an event stream generated by the model and an event stream that is derived from the execution. Fitness is measured in terms of string edit distance (*SSD*) and non-linear string distance metric (*NSD*) that can be weighted per-activity. The problem that is still unsolved in [3] is the selection of heuristic estimator that guarantee the minimum cost goal. In this paper, we solve the problem by proposing admissible heuristics that guarantees optimal result (see Thm. V.7). However, we do not cover fitness measurement in cases where process executions are assumed to be completed.

VIII. Conclusion

In this paper, we provide a robust cost-based technique to replay event logs on process models that is not only capable to deal with unobservable activities, but can also identify both skipped activities in process models and inserted activities in event logs.

As the work in this paper is based on a general framework using the A^* approach that guarantees to identify the best fit of a case in a Petri net, it provides a solid basis for benchmarking and further analysis based on replay, such as process conformance and performance analysis.

Finally, the benefit of always finding the best possible sequence of transitions to fire given the cost parameters comes at the cost of computational complexity, but real-life experiments show practical applicability.

Acknowledgements. This research is funded by the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 257593 (ACSI).

References

[1] A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Towards Robust Conformance Checking. In *Proceedings of the 6th Workshop on Business Process Intelligence (BPI 2010)*, 2010.

[2] A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Cost-Based Conformance Checking using the A* Algorithm. BPM Center Report BPM-11-11, BPMcenter.org, 2011.

[3] J.E. Cook, C. He, and C. Ma. Measuring Behavioral Correspondence to a Timed Concurrent Model. In *Software Maintenance, 2001. Proceedings. IEEE International Conference on*, pages 332 –341, 2001.

[4] J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 8:147–176, April 1999.

[5] A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: an Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14:245–304, 2007. 10.1007/s10618-006-0061-7.

[6] R. Dechter and J. Pearl. Generalized Best-first Search Strategies and the Optimality of A*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.

[7] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust Process Discovery with Artificial Negative Events. *The Journal of Machine Learning Research*, 10:1305–1340, 2009.

[8] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Discovering Expressive Process Models by Clustering Log Traces. *IEEE Trans. on Knowl. and Data Eng.*, 18:1010–1027, August 2006.

[9] K.L. McMillan and D.K. Probst. A Technique of State Space Search based on Unfolding. *Formal Methods in System Design*, 6:45–65, 1995. 10.1007/BF01384314.

[10] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, August 2002.

[11] A. Rozinat, I.S.M. de Jong, C.W. Günther, and W.M.P. van der Aalst. Conformance Analysis of ASML's Test Process. In *Proceedings of the Second International Workshop on Governance, Risk and Compliance (GRCIS'09)*, volume 459, pages 1–15. CEUR-WS.org, 2009.

[12] A. Rozinat, I.S.M. de Jong, C.W. Günther, and W.M.P. van der Aalst. Process Mining Applied to the Test Process of Wafer Steppers in ASML. *IEEE Transactions on Systems, Man and Cybernetics - Part C*, 39:474–479, 2009.

[13] A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33:64–95, March 2008.

[14] W.M.P. van der Aalst. Business Alignment: using Process Mining as a Tool for Delta Analysis and Conformance Testing. *Requirements Engineering*, 10:198–211, November 2005.

[15] W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Verlag, 2011. ISBN:978-3-642-19344-6.

[16] W.M.P. van der Aalst, K.M. van Hee, J.M. van der Werf, and M. Verdonk. Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. *Computer*, 43:90–93, March 2010.

[17] J. D. Weert, M. D. Backer, J. Vanthienen, and B. Baesens. A Critical Evaluation Study of Model-log Metrics in Process Discovery. In *Proceedings of the 6th Workshop on Business Process Intelligence (BPI 2010)*, 2010.

Authorized licensed use limited to: Wuhan University. Downloaded on August 31,2022 at 08:27:32 UTC from IEEE Xplore. Restrictions apply.