Data & Knowledge Engineering 154 (2024) 102353

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

# Reasoning on responsibilities for optimal process alignment computation

Matteo Baldoni, Cristina Baroglio, Elisa Marengo *, Roberto Micalizio

*University of Torino, Department of Computer Science, Corso Svizzera 185, 10149, Torino, Italy*

ARTICLE INFO

*Keywords:*
Process mining
Formal methods
Knowledge representation and reasoning

ABSTRACT

Process alignment aims at establishing a matching between a process model run and a log trace. To improve such a matching, process alignment techniques often exploit contextual conditions to enable computations that are more informed than the simple edit distance between model runs and log traces. The paper introduces a novel approach to process alignment which relies on contextual information expressed as *responsibilities*. The notion of responsibility is fundamental in business and organization models, but it is often overlooked. We show the computation of optimal alignments can take advantage of responsibilities. We leverage on them in two ways. First, responsibilities may sometimes justify deviations. In these cases, we consider them as correct behaviors rather than errors. Second, responsibilities can either be met or neglected in the execution of a trace. Thus, we prefer alignments where neglected responsibilities are minimized.

The paper proposes a formal framework for responsibilities in a process model, including the definition of cost functions for computing optimal alignments. We also propose a branch-and-bound algorithm for optimal alignment computation and exemplify its usage by way of two event logs from real executions.

## 1. Introduction

In process mining, an execution trace records the sequence of events that have been performed in executing a process. When the process that is being executed is also designed and represented by a process model specifying the expected executions, *conformance checking* techniques allow matching the execution trace with one of the possible model runs [1,2]. Among the existing conformance checking techniques, in this paper we consider *trace alignment*, which identifies the events in an execution trace matching the process model and events in the trace deviating from the model. The information resulting from trace alignment supports further investigations such as performance analysis [3] and diagnosis [4].

Several approaches have been proposed in the literature for computing the best alignment between a trace and a model [1]. One technique considers each possible model run, and compares it step by step with the events in the logged trace. In this way, for each possible run one can identify the mismatches between the run and the trace and evaluate each mismatch by means of a cost function of interest. The desired alignment is the *optimal* one, that is the alignment where the overall cost is minimized. In the classical approach, an alignment is evaluated by considering the number of mismatches between the trace and the model. As a consequence, the optimal alignment is the one that minimizes the number of mismatches between the two. The operation of counting all misalignments, however, does not consider possible relationships between activities. So, for instance, misalignments

* Corresponding author.
*E-mail addresses:* matteo.baldoni@unito.it (M. Baldoni), cristina.baroglio@unito.it (C. Baroglio), elisa.marengo@unito.it (E. Marengo), roberto.micalizio@unito.it (R. Micalizio).

https://doi.org/10.1016/j.datak.2024.102353
Received 8 February 2024; Received in revised form 31 July 2024; Accepted 4 September 2024
Available online 19 September 2024

on early activities are bound to have greater effects on the execution. Moreover, in many cases the lack of occurrence of some activity justifies the lack of other events, because it would be a mistake to carry out those activities in that context. More recent works (e.g., [5,6]), propose to compute the cost of an alignment by accounting for additional *context information*. In [5], early misalignments (i.e., those which occur closer to the beginning of a trace) have a higher cost. Instead, in [6], the evaluation of a mismatch depends on the activities that precede and follow the event at issue in the trace.

In this work, we focus on *responsibilities*. The Object Management Group Standards Development Organization (OMG SDO) places responsibilities at the specification level, in the *Business Motivation Model* (BMM) [7], stating that every *Business Process* must be under the responsibility of some *Organizational Unit*, in order for an enterprise to be able to justify its decisions. We propose to exploit responsibilities as *contextual information*. Responsibilities are often used for capturing task distribution with the expectation that the one who is responsible for a task will perform it at the right moment and with the right means. This is, for instance, the case of RACI matrices, which are indeed used to specify the different kinds of responsibilities that roles have on activities. As another example, in BPMN the specification of the role which is responsible for the execution of a process activity is modeled by means of lanes. Responsibilities can, however, be more complex. For instance, an activity might be adequately executed only if certain other activities occur conforming a given pattern (think of two activities to be performed in sequence because the former provides an input needed by the latter).

More in details, we propose a declarative responsibility specification which allows one to complement a procedural process model with explicit relationships between activities, that are not already captured by the process model itself. Such relationships include, but are not limited to, causal relationships, where an activity produces an outcome that is needed by another. So, for instance, in the case of a sequence of activities, the process model does not express necessarily possible causal dependencies. As a consequence, the classical approaches to alignment cannot but maximize the number of matches between the execution at issue and a process run. Thus, the case in which none of two consecutive activities is performed always counts as two mismatches with respect to the model. However, the absence of the second activity might be justified by the lack of proper input given the absence of the first activity.

We assume that the actors, playing roles in a business organization, do act to support the organization in reaching its goals, and hence they behave in accordance with their responsibilities. Under this assumption, we add responsibilities to the model description and we define a cost of alignment which depends both on the number of mismatches, and on respect of the responsibilities that are defined in the model. While neglected responsibilities always amount to costs, mismatches increase the cost only when they are not consistent with some responsibilities. Therefore, responsibilities not only enable a more informed search for an optimal alignment than just the control flow, but also represent a valuable source of information in the aftermath. In fact, the possibility of reasoning on the set of satisfied and neglected responsibilities allows a better understanding of what went wrong in the execution, and possibly pinpoint the root causes. This information could be used to improve the process, for instance, by regimenting some parts of the process or to make it more permissive.

This article extends the preliminary study of process alignment with responsibilities discussed in [8] as follows: (*i*) We extended the declarative formalism for responsibility representation, which supports the specification of partial orderings among activities, so as to support different weights associated with each responsibility. This is useful in some domains where not all the responsibilities have the same importance. (*ii*) Concerning the alignment strategy, that accounts for mismatches with the process model as well as responsibilities (which are either satisfied or neglected), the alignment cost function has been revised so to take into consideration the different weights associated with the responsibilities. (*iii*) An algorithm to compute all the optimal alignments and its implementation is presented in formal terms and thoroughly discussed (both were missing in the preliminary study). (*iv*) We extended the presentation of the Traffic Fine Management Process [9] reporting the complete process model extended with the responsibilities. To show the generalizability of the approach, we also modeled an example on a different domain: the Reimbursement Process for Domestic Declarations [10] (part of the BPI 2020 challenge). For both cases, we present a qualitative analysis of the alignments obtained with our implementation, and compare them with the solutions obtained in the classical approach.

The paper is organized as follows. Section 2 introduces the responsibility relations, their formalization and the formalization of the concept of progression of a responsibility. Section 3 formalizes the concept of process model extended with the responsibilities. Alignments are presented in Section 4 as well as the cost functions used to determine the optimal ones. The algorithm is presented in Section 5. In Sections 6 and 7 we report two examples showing and discussing the optimal alignments computed by the algorithm. Related works in Section 8 and Conclusions and Future Work in Section 9 end the paper.

## 2. Responsibility formalization and progression

The term responsibility is associated with multiple shades of meaning [11]. In Multi-Agent Organizations (MAO), for instance, agents take on responsibilities by committing to organizational norms, accepting to pursue certain organizational goals. Responsibilities are, thus, a tool by means of which MAOs realize distributed business processes [12]. In this paper, as well as in BMM and other business models, responsibility refers to a task assigned to an actor (*role responsibility* in the terminology by Vincent [11]). Indeed, in business organizations, responsibilities are often used to characterize *roles* (they provide a job description). An actor playing a role adopts all the responsibilities coming with it. Our aim is to use the events in a trace as contextual information for trace alignment, relying on the assumption that the role players will act so to respect as much as possible their responsibilities.

**Definition 1** (*Responsibility*)**.** A responsibility relation is formally denoted as $R(x, \mathfrak{u}, \mathfrak{v})$ where $x$ is a role, $\mathfrak{u}$ is a *context* condition, and $\mathfrak{v}$ is the *task* assigned to $x$.

Intuitively, $R(x, \mathsf{u}, \mathsf{v})$ states that any actor playing role $x$ will be receptive to the request/need of bringing about $\mathsf{v}$ if $\mathsf{u}$ holds.

Condition $\mathsf{u}$ and task $\mathsf{v}$ can both be simple activities or temporal patterns on activities executions.

*Example (Alignments and Responsibilities).* We consider a Fine Management Process. When a fine is created (Create Fine), then it can either be paid directly by the offender (Payment), or it is first sent to the offender (Send Fine) and, then, the notification is stored in the municipality system (Insert Fine Notification).

Following this description, only two model runs are possible:

- $E1 = \langle$Create Fine, Payment$\rangle$
- $E2 = \langle$Create Fine, Send Fine, Insert Fine Notification$\rangle$

For brevity $\langle \mathsf{CF}, \mathsf{P}\rangle$ and $\langle \mathsf{CF}, \mathsf{SF}, \mathsf{IFN}\rangle$ respectively.

Let us consider the observed execution trace: $T = \langle \mathsf{CF}\rangle$; that is, only Create Fine is observed. The possible alignments with trace $T$ are the following, where $\gg$ represents a mismatch (i.e., a move where either the log trace or the model move one step).

$$A1 = \frac{\mathsf{CF} \quad \gg}{\mathsf{CF} \quad \mathsf{P}} \qquad A2 = \frac{\mathsf{CF} \quad \gg \quad \gg}{\mathsf{CF} \quad \mathsf{SF} \quad \mathsf{IFN}}$$

Classical approaches would conclude that $A1$ is the optimal alignment having one mismatch only, while $A2$ has two. Therefore, the model execution closer to trace $T$ is $E1$. Let us now assume that the model is complemented with an explicit representation of responsibilities, and that the employee is (always) responsible for inserting the notification of a fine *only after* the fine has been sent and *only in case* it is sent. Assessing the two alignments against such a responsibility allows us to observe that the lack of Insert Fine Notification (IFN) in $A2$ is justified by the fact that the fine was not sent. Therefore, while the absence of Send Fine (SF) is not justified by any responsibility (and thus has to be counted as a mismatch in the alignment), the absence of Insert Fine Notification is justified by the non-occurrence of Send Fine. As a result, the two alignments can be considered as equivalent in terms of number of mismatches.

Responsibilities provide, in a declarative manner, the expected context of an activity, which is precious for interpreting a logged trace in a way that goes beyond the syntactic distance between strings.

In principle, context and task conditions can be expressed by choosing one of the various alternatives proposed in literature, from LTL and its variants [13,14], to Declare and its extensions such as MP-Declare [15]. In this paper we adopt precedence logic defined in [16] due to its simplicity and to its notion of progression, that is well suited for our purposes. Specifically, given a responsibility $R(x, \mathsf{u}, \mathsf{v})$, we denote the conditions $\mathsf{u}$ and $\mathsf{v}$ as precedence logic expressions [16], defined over the set of symbols $\Sigma \cup \{0, \top\}$; here, $\Sigma$ is a set of activity symbols, 0 means false, and $\top$ means true. Section 2.1 provides a background on Precedence Logic as defined in [16].

### 2.1. Background on precedence logic

Precedence logic is an event-based linear temporal logic, obtained from propositional logic augmented with the temporal operator $(\cdot)$ *before*. Such an operator is used to express minimal ordering requirements between events. For instance, $a \cdot b$ expresses the requirement for event $a$ to occur before the occurrence of event $b$. Note that $a$ does not need to occur *strictly* before $b$, other events may occur between $a$ and $b$. Besides the before operator, the logic includes the $\vee$ (choice) and the $\wedge$ (interleaving) operators (capturing that two conditions need to be satisfied but there is no temporal requirement between them). Given a workflow $u$ expressed in precedence logic, the *residual* of $u$ against an event $e$, denoted as $u/e$, defines the evolution of $u$ after the occurrence of event $e$. The residual operator is defined by rules (1 – 8) below, defined in [16,17]. Here, $u$ is a given workflow, $e$ is an event or $\top$, its complement $\bar{e}$ represents the non-occurrence of $e$, and $\Gamma_u$ represents the set of literals in $u$ and their complements (e.g., $\Gamma_{a\cdot b} = \{a, \bar{a}, b, \bar{b}\}$). The residual $u/e$ is defined as:

(1) $0/e \doteq 0$
(2) $\top/e \doteq \top$
(3) $(u_1 \wedge u_2)/e \doteq ((u_1/e) \wedge (u_2/e))$
(4) $(u_1 \vee u_2)/e \doteq ((u_1/e) \vee (u_2/e))$
(5) $(e \cdot u_1)/e \doteq u_1$ if $e \notin \Gamma_{u_1}$
(6) $(u_1/e) \doteq u_1$ if $e \notin \Gamma_{u_1}$
(7) $(e' \cdot u_1)/e \doteq 0$ if $e \in \Gamma_{u_1}$
(8) $(\bar{e} \cdot u_1)/e \doteq 0$

Since 0 amounts to false, and $\top$ to true, the residual operator can be used for assessing whether a workflow expression $u$ is satisfied by a given sequence of events $\sigma = \langle \sigma_1, \ldots, \sigma_m\rangle$ in $\Sigma$. Specifically, we denote as $u/\sigma$ the expression $(((u/\sigma_1)/\sigma_2)\ldots)/\sigma_m$. When $u/\sigma$ leads to $\top$, $\sigma$ is a possible execution run of $u$. When $u/\sigma$ leads to 0, $\sigma$ represents a trace not compliant with $u$. For instance, the formula $a \cdot b$ residuates to $\top$ with respect to the sequence $\sigma = \langle a, e, b\rangle$ as follows: $(a \cdot b)/\sigma = (a \cdot b)/\langle a, e, b\rangle \overset{(5)}{=} (b)/\langle e, b\rangle \overset{(6)}{=} (b)/\langle b\rangle \overset{(5)}{=} \top$. The same formula would residuate to 0 with respect to $\sigma' = \langle b, a\rangle$ as follows: $(a \cdot b)/\sigma' = (a \cdot b)/\langle b, a\rangle \overset{(7)}{=} (0)/\langle a\rangle \overset{(1)}{=} 0$. According to [16], it is assumed that (*i*) the events in $\sigma$ are non-repeating (timestamps can be used to differentiate multiple instances of the same event [16]), and (*ii*) an event $e$ and its complement $\bar{e}$ are mutually exclusive in every sequence $\sigma$.

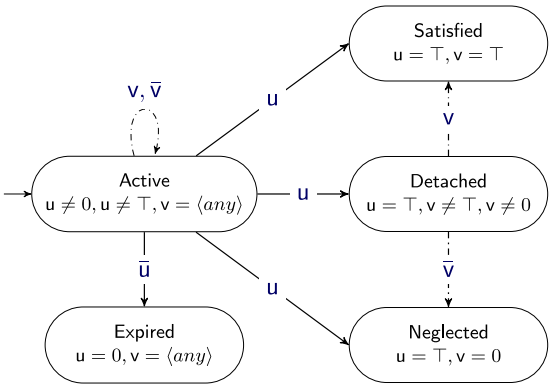


**Fig. 1.** The state transition of responsibility $R(x, \mathsf{u}, \mathsf{v})$. Changes of context condition u are marked as solid edges; changes of task condition v are marked as dash-dotted edges.

### 2.2. Responsibilities progression over a trace

Relying on precedence logic gives us two advantages: *generality*, since we can model both contexts and duties as workflows, and *semantics*, since we can make the state of a responsibility progress against a log trace, by relying on the residual operator. Specifically, given an execution trace $\sigma = \langle \sigma_1, \dots, \sigma_m \rangle$ of events over $\Sigma$, the state of responsibility $R(x, \mathsf{u}, \mathsf{v})$ is either (*i*) active, (*ii*) detached, (*iii*) expired, (*iv*) neglected, or (*v*) satisfied, as depicted in Fig. 1.

A responsibility is active when it exists according to the process model specification (as we will discuss in Section 3). Formally, given $\langle \sigma_1, \dots, \sigma_i \rangle$ a prefix of $\sigma$ events with $1 \leq i \leq m$, an active responsibility at step $i$ progresses to another state as follows:

- $R(x, \mathsf{u}, \mathsf{v})$ is *detached* at step $i$ (s.t. $i < m$), if $\mathsf{u}/\langle \sigma_1, \dots, \sigma_i \rangle = \top$ and $\mathsf{v}/\langle \sigma_1, \dots, \sigma_i \rangle$ is neither $\top$ nor 0;
- $R(x, \mathsf{u}, \mathsf{v})$ is *expired* at step $i$ if $\mathsf{u}/\langle \sigma_1, \dots, \sigma_i \rangle = 0$ (the residual of v is irrelevant);
- $R(x, \mathsf{u}, \mathsf{v})$ is *satisfied* at step $i$ if $\mathsf{u}/\langle \sigma_1, \dots, \sigma_i \rangle = \top$ and $\mathsf{v}/\langle \sigma_1, \dots, \sigma_i \rangle = \top$;
- $R(x, \mathsf{u}, \mathsf{v})$ is *neglected* at step $i$ if $\mathsf{u}/\langle \sigma_1, \dots, \sigma_i \rangle = \top$ and $\mathsf{v}/\langle \sigma_1, \dots, \sigma_i \rangle = 0$, or at step $m$ (the end of the execution) when $\mathsf{u}/\sigma = \top$ and $\mathsf{v}/\sigma$ is not $\top$.

Intuitively, responsibilities are associated with process activities, so they come into existence (i.e., active) when the associated activity is performed; this aspect will be formalized in Section 3, where we explain how a process control flow is annotated with responsibilities. When the responsibility is detached there is an expectation on role $x$ to bring about v since the context condition u holds. When the responsibility is expired, instead, the context condition cannot hold along the given $\sigma$, and hence no expectation about v can be made. The responsibility is satisfied along $\sigma$ when both u and v progress to $\top$. Finally, a responsibility is neglected either when, at any execution step, the context condition u holds and the task v progressed to 0, or when, at the end of the trace, u holds and v has not progressed to $\top$, that is, the expectation created with u has not been met. Fig. 1 summarizes these state changes; each state reports the truth values for conditions u and v (the label *any* means that the actual truth value of v is irrelevant). Note that, as depicted in the figure, once a responsibility reaches a *satisfied*, *neglected* or *expired* state, it would not be possible to transit to another state. This is due to the fact that responsibilities are expressed and evaluated w.r.t. events occurrence: Once a sequence of events satisfying (resp. neglecting or expiring) a responsibility occur, no matter how the trace evolves, there will be no way to change the state of that responsibility from the one of satisfied (resp. neglected or expired).

*Example (Responsibilities)* Let us consider again the Fine Management Process, and the set $\Sigma$ of activity symbols $\{\mathsf{CF}, \mathsf{SF}, \mathsf{IFN}\}$. Consider a responsibility relation $R(x, \top, \mathsf{SF} \cdot \mathsf{IFN})$ expressing that the notification has to be inserted (IFN) only after the fine has been sent (SF). Let us consider the execution $\langle \mathsf{CF}, \mathsf{IFN} \rangle$ and apply the residual with respect to it. First, the event CF in the execution has to be considered. Since $\mathsf{CF} \notin \Gamma_{\mathsf{SF} \cdot \mathsf{IFN}}$, rule (6) applies: $\mathsf{SF} \cdot \mathsf{IFN}/\mathsf{CF} \overset{(6)}{=} \mathsf{SF} \cdot \mathsf{IFN}$. Then, the event in the trace following CF is IFN. In this case, rule (7) applies to $\mathsf{SF} \cdot \mathsf{IFN}/\mathsf{IFN}$ since $\mathsf{IFN} \in \Gamma_{\mathsf{SF} \cdot \mathsf{IFN}}$, bringing the responsibility to be neglected: $\mathsf{SF} \cdot \mathsf{IFN}/\mathsf{IFN} \overset{(7)}{=} 0$.

### 2.3. Responsibility weight

Responsibilities may not all have the same importance in an organization, when balanced against its business rules and the achievement of its business goals. For instance, the responsibility of sending a receipt only after the payment is more crucial for an organization than the responsibility of asking a customer to fill in a satisfaction questionnaire. To capture this aspect, we pair each responsibility with a positive number representing its *weight*, formally denoted as $\langle R(x, \mathsf{u}, \mathsf{v}), \mathcal{V} \rangle$. As a special case, one may consider all responsibilities equally important, and give them the same weight. Responsibility weights are an important piece of information that we use in the search for the optimal alignment. In fact, we consider the weight of a neglected responsibility as a penalty, and we look for alignments minimizing them.

## 3. Process model: Process net and responsibilities

In our approach a process model accounts both for the control flow, and for responsibility relations assigned to roles taking part to the process. We model the control flow as a *Process Net*, specified as a labeled Petri Net in Definition 2, and enrich it with a set of responsibilities, annotating the same net.

We define a process net as an extension of the process model given in [5] by including a set of roles and assigning them to the activities.

**Definition 2** (*Process Net*)**.** A Process Net is a Labeled Petri Net defined as a tuple $\mathsf{N} = \langle P, T, F, m_0, m_f, \Sigma, \lambda, Z, \zeta \rangle$, where $P$ is the set of places, $T$ is the set of transitions (with $P \cap T = \emptyset$), $F$ is the flow relation $F \subseteq (P \times T) \cup (T \times P)$, $m_0$ is the initial marking, $m_f$ is the final marking, $\Sigma$ is the set of activity symbols, $\lambda : T \to \Sigma \cup \{\tau\}$ labels every transition by an activity or as silent, $Z$ is the set of roles, and $\zeta : \Sigma \to Z$ assigns a role to every activity in $\Sigma$.

A process net N sets the scope of responsibility relations, since it specifies both the roles $Z$ and the activities $\Sigma$ over which a responsibility is defined.

Responsibilities are defined at design time, and are "attached" to activities: a responsibility gets *active* when the activity it is associated with is performed. This means that from this step on, the responsibility is considered in the computation of an optimal alignment. The rationale is that performing an activity may have consequences, and these consequences are captured via responsibilities. Considering the fine management scenario sketched above, responsibility $R(x,\ \mathsf{IFN},\ \mathsf{CF} \cdot \mathsf{SF} \cdot \mathsf{IFN})$ can be associated with activity Insert Fine Notification (IFN) meaning that when IFN is performed, the responsibility becomes detached; moreover, the responsibility is satisfied only when Create Fine (CF) and Send Fine (SF) have already occurred in the very same order, possibly interleaved with other events.

Definition 3 formally defines the responsibility labeling of a process net.

**Definition 3** (*Responsibility Labeling*)**.** Let N be a process net, and let $Z$ and $\Sigma$ be, respectively, the set of roles and activity symbols in N. A responsibility labeling over N is a function $\mathsf{R} : \Sigma \to \{r_1, \ldots, r_n\}$ where each $r_i$ is a pair $\langle R(x_i, \mathsf{u}_i, \mathsf{v}_i), \mathcal{V}_i \rangle$, such that: $x_i \in Z$, $\mathsf{u}_i$ and $\mathsf{v}_i$ are precedence logic expressions over $\Sigma \cup \{0, \top\}$, and $\mathcal{V}_i \in \mathbb{R}^+$ is the associated responsibility weight.

For the sake of readability, in the following we will omit the cost of a responsibility when not strictly necessary in the discussion.

In our approach a process model is a pair consisting of a process net and a responsibility labeling over that net, formally:

**Definition 4** (*Process Model*)**.** A process model is a tuple $\mathsf{M} = \langle \mathsf{N}, \mathsf{R} \rangle$ where N is the process net as in Definition 2, and R is a responsibility labeling as in Definition 3.

Given a process model $\mathsf{M} = \langle \mathsf{N}, \mathsf{R} \rangle$, we use the term *model run* for the sequence of activity symbols in $\Sigma$ produced by a full run of the process net N, where a Petri Net full run is a sequence of firings from the initial marking to the final one [5]. We also assume the process net N to be *easy sound* [1], that is, there exists at least one full run.

In addition, we expect that responsibilities in R are defined consistently with the flow model N they refer to. That is, for each full run of a model, the responsibilities detached along that run are also satisfied before the end of the very same run.

## 4. Flow and responsibility alignments

An alignment compares a process execution against an execution trace (i.e., a log trace). Generally, the objective is to find, among the possible ones, an alignment which is optimal w.r.t. a criterion of preference. Intuitively, an alignment proceeds step-by-step on the model and on the log: at each step, if the activity in the model and the one in the log match each other, a *synchronous move* is made, and both model and log advance one step. Otherwise, either the *model moves* and the log does not, or the other way around, the *log moves* and the model does not. Usually, to find an optimal matching, a cost function associated with mismatches (i.e., asynchronous moves) is defined. So, an optimal alignment is the one minimizing the cumulative cost of the mismatches. Among the existing approaches, the classical one is to assign the same cost to each mismatch, so optimality is reached by minimizing the number of asynchronous moves [1].

In our approach, an optimal alignment is determined by taking into account both the alignment between a log trace and a model run, and the involved responsibilities. We refer to the former as *flow alignment* and to the latter as *responsibility alignment*. Definition 5, adapted from [5], formally introduces the notion of *flow alignment*, capturing the alignment between a process net and an execution trace. The symbol $\gg$ represents a *no-move*, and is used for marking asynchronous moves. In the following, the term *log trace* refers to an actual execution. It is a finite sequence of activity symbols $\sigma \in \Sigma^*$ (i.e., the space of sequences defined over symbols in $\Sigma$).

**Definition 5** (*Flow Alignment*)**.** Let $\sigma = \langle \sigma_1, \ldots, \sigma_m \rangle$ be a log trace in $\Sigma^*$, and $\mathsf{N} = \langle P, T, F, m_0, m_f, \Sigma, \lambda, Z, \zeta \rangle$ a process net. An alignment of $\sigma$ with the process net N is a finite sequence $\varphi = \langle (\sigma'_1, u'_1), \ldots, (\sigma'_p, u'_p) \rangle$ of moves such that:

- each move is either: a synchronous move $(a, t) \in \Sigma \times T$ with $a = \lambda(t)$, a log move $(a, \gg)$, or a model move $(\gg, t)$,
- dropping the $\gg$ symbols from the left projection $(\sigma'_1, \ldots, \sigma'_p)$ of $\varphi$, yields $\sigma$, this is also dubbed *trace projection*,

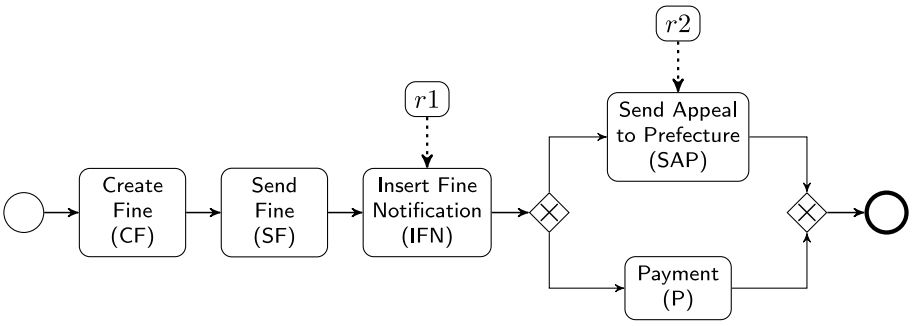


**Fig. 2.** A simplified version of the Fine Management process.

- dropping the $\gg$ symbols from the right, or model, projection $(u'_1, \ldots, u'_p)$ of $\varphi$, yields a full run $u$ of $\mathsf{N}$, this is also dubbed *model projection*.

To evaluate the best model approximation of a log trace, we also consider the responsibility relations. Intuitively, we collect all the responsibilities attached to the activities of the model projection of the alignment (i.e., the active responsibilities that should be satisfied along a possible, expected execution), and progress them against the log trace (to check whether they are satisfied, or not). The cost of an alignment, thus, takes also into account the cost of neglected responsibilities. More importantly, the responsibilities that are active along a model run give us a context for assessing whether a model move (i.e., a "skip" on the log side) actually represents an execution error, or it corresponds to a proper behavior being justified by some responsibility.

Given a flow alignment $\varphi$, its *responsibility set* is the set of responsibilities (together with their corresponding weights) attached to the activities in the model run given by the right projection of $\varphi$ (i.e., the model projection). In general, a responsibility set can be computed for any non-empty prefix of $\varphi$ by considering the alignment up to a given step $j$.

**Definition 6** (*Responsibility Set*). Let $\varphi = \langle (\sigma'_1, u'_1), \ldots, (\sigma'_p, u'_p) \rangle$ be a flow alignment between a process model $\mathsf{M} = \langle \mathsf{N}, \mathsf{R} \rangle$ and a log trace $\sigma \in \Sigma^*$, the responsibility set $\mathscr{R}^{\varphi,j}$ for the alignment $\varphi$ at step $j$ $(1 \leq j \leq p)$ is defined as $\mathscr{R}^{\varphi,j} = \cup_{i=1}^{j} \mathsf{R}(\lambda(u'_i))$.

It holds $\mathsf{R}(\gg) = \emptyset$. As a shortcut, we denote as $\mathscr{R}^{\varphi}$ the set $\mathscr{R}^{\varphi,p}$, that is the set of responsibilities computed considering all the steps in the alignment $\varphi$. Summing up, a responsibility set $\mathscr{R}^{\varphi}$ collects the active responsibilities for a given model run in the alignment $\varphi$ (i.e., model projection of $\varphi$). These responsibilities are actually satisfied or neglected depending on the activities that are included in the log trace (i.e., trace projection of $\varphi$). For instance, given the example in Section 2, the responsibility set is computed for the alignments *A1* and *A2*, by respectively considering the model activities in *E1* and *E2*. The resulting sets of responsibilities are then, residuated with respect to the log trace *T*.

For convenience, we extend the notion of residuation of the precedence logic to responsibility relations and to a responsibility set. Given a responsibility set $\mathscr{R}^{\varphi} = \{\mathsf{r}_1, \ldots, \mathsf{r}_k\}$ with $\mathsf{r}_i = \langle R_i, \mathcal{V}_i \rangle$ and $R_i = R(\mathsf{x}_i, \mathsf{u}_i, \mathsf{v}_i)$, let $\sigma' = \langle \sigma'_1, \ldots, \sigma'_p \rangle$ be the trace projection of $\varphi$. Then, the notation $R_i/\sigma'$ is a shorthand for $R(\mathsf{x}_i, \mathsf{u}_i/\sigma', \mathsf{v}_i/\sigma')$. Notation wise, $\mathsf{r}_i/\sigma'$ for $\mathsf{r}_i = \langle R_i, \mathcal{V}_i \rangle$ is a shorthand for $\langle R_i/\sigma', \mathcal{V}_i \rangle$ and $\mathscr{R}^{\varphi}/\sigma'$ is a shorthand for $\{\mathsf{r}_1/\sigma', \ldots, \mathsf{r}_k/\sigma'\}$. Additionally, the residuation of any expression $\mathsf{u}$ with $\gg$ as no effect on the expression, namely $\mathsf{u}/\gg = \mathsf{u}$.

**Proposition 1** (*Consistency*). *Let $\mathscr{R}^{\varphi,j}$ be the responsibility set computed at step $j$ $(1 \leq j \leq p)$ of an alignment $\varphi = \langle (\sigma'_1, u'_1), \ldots, (\sigma'_p, u'_p) \rangle$, let $\sigma'$ be the trace projection of $\varphi$, then for each $r_i \in \mathscr{R}^{\varphi,j}$ the following conditions hold:*

1. *if $r_i/\langle \sigma'_1, \ldots, \sigma'_j \rangle$ is satisfied, then also $r_i/\sigma'$ is satisfied;*
2. *if $r_i/\langle \sigma'_1, \ldots, \sigma'_j \rangle$ is neglected, then also $r_i/\sigma'$ is neglected;*
3. *if $r_i/\langle \sigma'_1, \ldots, \sigma'_j \rangle$ is expired, then also $r_i/\sigma'$ is expired;*
4. *if $r_i/\langle \sigma'_1, \ldots, \sigma'_j \rangle$ is detached, then $r_i/\sigma'$ is either satisfied or neglected.*

This proposition, that follows directly from the rewriting rules of the precedence logic, guarantees a consistent progression of the responsibilities against a log trace. In fact, whenever a responsibility progresses from the detached state to either satisfied, neglected, or expired, such a second state is final: the state of the responsibility can no longer evolve along the same trace. That is, further events along the trace cannot satisfy a neglected responsibility nor vice versa. At the same time, a responsibility detached along a log trace must necessarily evolve to either satisfied or neglected by the end of the same trace. Proposition 1 formally defines what is graphically depicted in Fig. 1.

### 4.1. Example: Computing optimal alignments with responsibilities

In this section we provide a simplified example to show how responsibilities are used in the computation of optimal alignments. Fig. 2 shows a variant of the Fine Management Process sketched previously. In this process only two runs are possible due to

the choice between activities SAP and P, in mutual exclusion. Let us assume that the model is enriched with the following two responsibilities:

$$r1 : \langle R(x, \mathsf{IFN}, \mathsf{SF} \cdot \mathsf{IFN}), 1\rangle \qquad r2 : \langle R(x, \mathsf{SAP}, \bar{\mathsf{P}}), 1\rangle$$

Responsibility $r1$ is attached to activity IFN, whereas $r2$ is attached to activity SAP. The two responsibilities play a different role in the search for the optimal alignment. The former, in fact, describes a desired occurrence of events: when IFN occurs, activity SF must be previously occurred, otherwise the occurrence of IFN is not desired. The latter responsibility, instead, is used to prefer a model run over the other. To see how let us consider the following log trace $\sigma = \langle \mathsf{CF}, \mathsf{SAP}, \mathsf{P}\rangle$, and compute the two possible alignments.

$$A1 = \begin{array}{|c|c|c|c|c|}\hline \mathsf{CF} & \gg & \gg & \mathsf{SAP} & \mathsf{P} \\ \hline \mathsf{CF} & \mathsf{SF} & \mathsf{IFN} & \mathsf{SAP} & \gg \\ \hline \end{array} \qquad A2 = \begin{array}{|c|c|c|c|c|}\hline \mathsf{CF} & \gg & \gg & \mathsf{SAP} & \mathsf{P} \\ \hline \mathsf{CF} & \mathsf{SF} & \mathsf{IFN} & \gg & \mathsf{P} \\ \hline \end{array}$$

The two alignments have the same number of skips, so they could both be considered optimal in the classic approach. When we also consider responsibilities, though, the two alignments are evaluated differently. Responsibility $r1$ is activated in both alignments, as it is associated with IFN (which is part of the model projection of both alignments). By way of $r1$, we can justify the model move on IFN (i.e., the second $\gg$ in both alignments). In fact, had IFN occurred, the responsibility $r1$ would have been neglected. On the other hand, responsibility $r2$ is activated only by alignment $A1$, as it is associated with the model activity SAP. Such a responsibility is first detached by the synchronous move on SAP, and then it is neglected by the occurrence of activity P. Therefore, alignment $A1$ consists of two mismatches plus one neglected responsibility (amounting to cost 3 according to the cost function formalized in Section 4.2); Alignment $A2$ consists of two mismatches and none neglected responsibilities (amounting to cost 2).

The example, thus, shows the important role played by responsibilities in weighting two alignments, especially when spurious log traces mention activities that are expected to be mutually exclusive. Responsibility $r2$ is used exactly in this way: since $r2$ is attached with SAP, it is only considered in alignments where the model runs contain the very same activity; thus, when a log trace contains both P and SAP, responsibility $r2$ progresses to neglected, so adding a cost. On the other hand, alignments where the model runs mention P are not associated with a responsibility. Therefore, alignments passing from P are possibly preferred to alignments passing from SAP. The rationale is that a domain expert may want to model that in a trace containing both events, the problem is more likely to be on a wrong recording of the activity SAP and on the correct execution being the one passing from the payment. As can be noted from this example, reasoning on the set of neglected responsibilities when analyzing an alignment represents and additional source of information. Neglected responsibilities give an intuition of what is the reason for a misalignment.

### 4.2. Cost functions for optimal alignments

The previous example gives an intuition on the use of responsibilities when computing the cost of an alignment. We now formalize our cost function so to consider both the cost of the mismatches between the model run and the log trace, which we call *Flow Alignment Cost* $\mathscr{C}_{\mathsf{N},\varphi}$, and the cost for the neglected responsibilities, which we call *Responsibility Alignment Cost* $\mathscr{C}_{R,\varphi}$.

Let us start from the latter one. The responsibility cost $\mathscr{C}_{R,\varphi}$ is computed for a flow alignment $\varphi = \langle(\sigma'_1, u'_1), \dots, (\sigma'_p, u'_p)\rangle$ between a process model $\mathsf{M} = \langle \mathsf{N}, \mathsf{R}\rangle$ and a log trace $\sigma \in \Sigma^*$. The cost $\mathscr{C}_{R,\varphi}$ corresponds to the sum of the costs of responsibilities that are neglected in $\varphi$. To compute them, first the responsibility set $\mathscr{R}^\varphi$ for $\varphi$ is determined (Definition 6). Then, $\mathscr{R}^\varphi$ is residuated with respect to the trace projection of $\varphi$. Neglected responsibilities are then those that are detached, but not satisfied at the end of the trace. A formal definition is given in Definition 7.

**Definition 7** (*Responsibility Alignment Cost*)**.** Let $\varphi = \langle(\sigma'_1, u'_1), \dots, (\sigma'_p, u'_p)\rangle$ be the flow alignment between a process model $\mathsf{M} = \langle \mathsf{N}, \mathsf{R}\rangle$ and a log trace $\sigma \in \Sigma^*$, be $\sigma'$ the trace projection of $\varphi$, and let $\mathscr{R}^\varphi$ be the responsibility set for $\varphi$; the responsibility alignment cost is

$$\mathscr{C}_{R,\varphi} = \sum_{\langle R_i, \mathcal{V}_i\rangle \in \mathscr{R}^\varphi} \mathscr{C}_R(\langle R_i, \mathcal{V}_i\rangle / \sigma')$$

Where $\mathscr{C}_R(\langle R_i, \mathcal{V}_i\rangle / \sigma')$ is defined as follows

$$\mathscr{C}_R(\langle R(x_i, \mathsf{u}_i, \mathsf{v}_i), \mathcal{V}_i\rangle / \sigma') = \begin{cases} \mathcal{V}_i & \text{if } \mathsf{u}_i/\sigma' = \top \text{ and } \mathsf{v}_i/\sigma' \neq \top \\ 0 & \text{otherwise} \end{cases}$$

Given a responsibility and a log trace, the function $\mathscr{C}_R$ defined above residuates the responsibility with respect to the trace and returns the weight of the responsibility if it is neglected. This happens for a *detached* responsibility either when the task condition is violated (i.e., residuates to 0), or when the condition has not been achieved at the end of the log trace (i.e., is $\neq \top$).

The second component of our cost function is the *Flow Alignment Cost* function, that calculates the cost of every mismatch (i.e., either model or log moves) occurring in a given alignment. Notably, this calculation takes into account responsibilities as a sort of context. By using them, in fact, we are able to identify some model moves as correct, and not as mismatches. To this end, $\mathscr{C}_{\mathsf{N},\varphi}$ is computed with respect to a flow alignment $\varphi = \langle(\sigma'_1, u'_1), \dots, (\sigma'_p, u'_p)\rangle$ between a process model $\mathsf{M} = \langle \mathsf{N}, \mathsf{R}\rangle$ and a log trace $\sigma \in \Sigma^*$. To compute $\mathscr{C}_{\mathsf{N},\varphi}$, each alignment step $(\sigma'_j, u'_j)$ is considered. A step $(\sigma'_j, u'_j)$ is a mismatch, and hence has a cost $c$, when it is either a log move (i.e., the label $\lambda(u'_j)$ assigned to transition $u'_j$ is $\gg$), or it is a model move (i.e., $\sigma'_j$ equals $\gg$) which is not justified by

any responsibility. A model move is justified by a responsibility if there is at least one $\langle R(x, \mathsf{u}, \mathsf{v}), \mathcal{V}\rangle \in \mathscr{R}^{\varphi,j}$ which is not neglected at step $j$, but it would be if the log event $\sigma'_j$ were substituted by the corresponding model activity $\lambda(u'_j)$ (i.e., the activity labeling transition $u'_j$ in the synchronous product). In other words, executing the activity corresponding to a synchronous move would have lead a responsibility $R(x, \mathsf{u}, \mathsf{v})$ to progress to neglected i.e., $R(x, \top, 0)$.

**Definition 8** (*Justified Model Move*)**.** Let $\varphi$ be a (partial) flow alignment $\langle(\sigma'_1, u'_1), \ldots, (\sigma'_{j-1}, u'_{j-1}), (\gg\ , u'_j)\rangle$. Let $\sigma' = \langle\sigma'_1, \ldots, \sigma'_{j-1}\rangle$ the trace projection of $\varphi$, and let $\langle R(x, \mathsf{u}, \mathsf{v}), \mathcal{V}\rangle$ be a responsibility.

We say that $R(x, \mathsf{u}, \mathsf{v})$ *justifies* $\sigma'_j = \gg$ iff $\neg(\mathsf{u}/\sigma' = \top$ and $\mathsf{v}/\sigma' = 0)$ and $((\mathsf{u}/\sigma')/\lambda(u'_j) = \top$ and $(\mathsf{v}/\sigma')/\lambda(u'_j) = 0)$

This means that skipping activity $\lambda(u'_j)$ is consistent with a responsibility, and hence $\gg$ does not represent a misbehavior. Definition 9 relies on the definition of *Justified Model Move* to determine the cost of an alignment step.

**Definition 9** (*Flow Alignment Cost*)**.** Let $\varphi = \langle(\sigma'_1, u'_1), \ldots, (\sigma'_p, u'_p)\rangle$ be the flow alignment between a process model $\mathsf{M} = \langle \mathsf{N}, \mathsf{R}\rangle$ and a log trace $\sigma \in \Sigma^*$, be $\sigma'$ the trace projection of $\varphi$, and let $\mathscr{R}^{\varphi}$ be the responsibility set for $\varphi$; the flow alignment cost is

$$\mathscr{C}_{\mathsf{N},\varphi} = \sum_{j=1}^{j\leq p} \mathscr{C}_{\mathsf{N}}(\mathscr{R}^{\varphi,j}, \varphi_j, c)$$

where $\mathscr{C}_{\mathsf{N}}$ is a function that assigns either the cost $c$ or 0 at the alignment step $(\sigma'_j, u'_j)$ of $\varphi_j = \langle(\sigma'_1, u'_1), \ldots, (\sigma'_j, u'_j)\rangle$ as follows:

$$\mathscr{C}_{\mathsf{N}}(\mathscr{R}, \varphi_j, c) = \begin{cases} c & \text{if } \lambda(u'_j) = \gg \text{ or} \\ & \nexists\langle R(x, \mathsf{u}, \mathsf{v}), \mathcal{V}\rangle \in \mathscr{R}^{\varphi,j} \text{ s.t. } R(x, \mathsf{u}, \mathsf{v}) \text{ justifies } \sigma'_j = \gg; \\ 0 & \text{otherwise.} \end{cases}$$

The example in Section 4.1 has already shown how responsibilities can justify model moves, and hence affect the flow cost. Specifically, responsibility $r1$: $\langle R(x, \mathsf{IFN}, \mathsf{SF} \cdot \mathsf{IFN}), 1\rangle$, active both in alignment $A1$ and $A2$, is justified in both alignments as event SF has not been observed in the trace log $\sigma$, the occurrence of IFN in $\sigma$ would progress $r1$ to neglected. Thus, the model move on IFN is justified and not included in the flow cost.

Note that, for generality, we associate each mismatch with a cost $c$, however this constant can be replaced by any suitable cost function. For instance, approaches such as [5,6] determine the cost based on the mismatch and when it occurs.

To conclude, the total cost of an alignment is computed as the weighted sum of the flow and the responsibility costs.

**Definition 10** (*Alignment Cost, Optimal Alignment*)**.** Let $\varphi = \langle(\sigma'_1, u'_1), \ldots, (\sigma'_p, u'_p)\rangle$ be the flow alignment between a process model $\mathsf{M} = \langle \mathsf{N}, \mathsf{R}\rangle$ and a log trace $\sigma \in \Sigma^*$, the cost function of the alignment $\varphi$ is

$$\mathscr{C}_{\varphi} = \gamma \cdot \mathscr{C}_{\mathsf{N},\varphi} + \delta \cdot \mathscr{C}_{R,\varphi}$$

An alignment between a model M and a log trace $\sigma$ is optimal if $\mathscr{C}_{\varphi}$ is minimal.

Coefficients $\gamma$ and $\delta$ are domain-dependent weights that can be tuned for penalizing more either neglected responsibilities or asynchronous moves, and this has an impact on the shape of the sought alignments. A greater weight for $\delta$ (responsibility coefficient) prefers asynchronous moves to neglected responsibilities, and this may result in preferring alignments involving fewer responsibilities (since there are fewer opportunities for neglecting them). On the other hand, a greater weight for $\gamma$ (flow coefficient), prefers neglected responsibilities to mismatches, thus the found alignments would be characterized by as many synchronous moves as possible even though this could lead to violate many responsibilities.

## 5. Computation of the optimal alignment

To compute the optimal alignment, many approaches in the literature adopt A* or branch-and-bound algorithms. The approach we outline in this section exploits a branch-and-bound strategy; in particular, our solution finds all the optimal alignments. In Sections 6 and 7 we comment on the results obtained by applying the algorithm on two examples.

The branch-and-bound approach that we implemented relies on the Synchronous Product Net (SN) [18] between the process net and a sequential Petri Net representing the log trace. Intuitively, the SN combines the two nets representing the synchronous and asynchronous moves. The SN is built as follows: every model (resp. log) transition is augmented with the $\gg$ symbol to represent asynchronous moves. Synchronous moves are represented with additional transitions, each labeled with the activity synchronously performed. We rely on the formal definition of SN as in [1]. In this setting, an alignment corresponds to a *full run* in the SN, that is a firing sequence bringing from the initial marking to a final marking, where the final markings are those where both the process net and the log trace reach one of their final markings. Among the possible SN full runs, our approach finds *all* the optimal full runs, i.e., having all the same minimal cost, computed according to Definition 10.

Algorithm 1 reports a *branch and bound* strategy for the computation of the optimal alignments. The algorithm explores the SN looking, at each step, for the enabled transitions bringing from the current marking to a reachable one. It stores the frontier of the reachable markings and order them in non-decreasing order with respect to the cost. To this aim, the data structure that we use is a tuple $\langle\varphi, m, \mathscr{C}_{\varphi}, \mathscr{R}^{\varphi}\rangle$ where $\varphi$ is the sequence of transitions fired up to the reached marking $m$. That is, $\varphi$ is a partial

alignment found up to step $m$. The cost of the alignment corresponding to the sequence in $\varphi$ is $\mathscr{C}_{\varphi}$, and $\mathscr{R}^{\varphi}$ is the set of the active responsibilities collected up to the reached marking according to the alignment represented by $\varphi$. The algorithm stores in $\text{best}_{\varphi}$ and $\mathscr{C}_{\text{best}}$ respectively the set of the best solutions found up to the current iteration and their cost. At each iteration, the current node is first evaluated. If its cost is greater than $\mathscr{C}_{\text{best}}$ (line 6), then the search can terminate and the set of the best solutions $\text{best}_{\varphi}$ is returned (since markings are explored in a non-decreasing cost order, a solution cheaper than the current one does not exist). Thus, at this point, the algorithm has found all the optimal alignments. Otherwise, the algorithm checks if the marking is final (line 9). If this is the case, the cost of the found alignment is updated by adding the cost of the responsibilities which are *detached*: these responsibilities will not be satisfied anymore, and must thus be considered as neglected. Function `cost_detached_resp` (line 10) serves this purpose. Once the cost is updated, three alternative options are possible: either the found solution is better than the current one(s) (so both the set $\text{best}_{\varphi}$ and the cost $\mathscr{C}_{\text{best}}$ must be updated), or it has the same cost and thus is added to the $\text{best}_{\varphi}$ set, or it is more expensive, and hence discarded. These checks and updates are performed by `det_current_best` (line 11).

**Input :** SP $= (P, T, F, m_0, m_f, (\Sigma \cup \{\gg\}), \lambda, Z, \zeta)$: synchronous product,
R: mapping from $\Sigma$ to $\{\langle R, \mathcal{V}\rangle\}$, $\gamma$: flow alignment coefficient,
$\delta$: responsibility alignment coefficient

1 $Q \leftarrow \{\langle\langle\rangle, m_0, 0, \emptyset\rangle\}$
2 $\text{best}_{\varphi} = \{\}$
3 $\mathscr{C}_{\text{best}} = \infty$
4 **while** $Q \neq \emptyset$ **do**
5 $\quad \langle \varphi, m, \mathscr{C}_{\varphi}, \mathscr{R}^{\varphi}\rangle \leftarrow Q.pop()$
6 $\quad$ **if** $\mathscr{C}_{best} < \mathscr{C}_{\varphi}$ **then**
7 $\quad\quad$ **Return** $best_{\varphi}$
8 $\quad$ **end**
9 $\quad$ **if** $m = m_f$ **then**
10 $\quad\quad \mathscr{C}'_{\varphi} \leftarrow \mathscr{C}_{\varphi} + \delta \times$ `cost_detached_resp`$(\mathscr{R}^{\varphi}, \varphi)$
11 $\quad\quad \text{best}_{\varphi}, \mathscr{C}_{\text{best}} \leftarrow$ `det_current_best` $(\langle \varphi, m, \mathscr{C}'_{\varphi}, \mathscr{R}^{\varphi}\rangle, \text{best}_{\varphi}, \mathscr{C}_{\text{best}})$
12 $\quad$ **end**
13 $\quad$ **else if** $|\varphi| < |T|$ **then**
14 $\quad\quad$ **for** $t \in T$ *with* $m[t\rangle m'$ **do**
15 $\quad\quad\quad \varphi' \leftarrow \varphi \bullet t$
16 $\quad\quad\quad \langle a\rangle =$ `net_steps`$(\langle t\rangle)$
17 $\quad\quad\quad \mathscr{R}^{\varphi'} \leftarrow \mathscr{R}^{\varphi} \cup \mathsf{R}(a)$
18 $\quad\quad\quad \mathscr{C}'_{\mathsf{N},\varphi} \leftarrow$ `compute_mismatch_cost`$(\mathscr{R}^{\varphi'}, \varphi, t)$
19 $\quad\quad\quad \mathscr{C}'_{R,\varphi} \leftarrow$ `compute_resp_cost`$(\mathscr{R}^{\varphi}, \mathscr{R}^{\varphi'}, \varphi, t)$
20 $\quad\quad\quad \mathscr{C}'_{\varphi} \leftarrow \mathscr{C}_{\varphi} + \gamma \times \mathscr{C}'_{\mathsf{N},\varphi} + \delta \times \mathscr{C}'_{R,\varphi}$
21 $\quad\quad\quad Q \leftarrow Q.insert(\langle \varphi', m', \mathscr{C}'_{\varphi}, \mathscr{R}^{\varphi'}\rangle)$
22 $\quad\quad$ **end**
23 $\quad$ **end**
24 **end**
25 **Return** $best_{\varphi}$

**Algorithm 1:** Computation of all the optimal alignments.

If none of these conditions is satisfied, the marking needs to be explored by considering each enabled transition from it (line 14). Each transition $t$ corresponds to a new step appended to the current alignment $\varphi$. The resulting alignment $\varphi'$ is then evaluated by computing its cost with respect to both flow and responsibilities. This is done by functions `compute_mismatch_cost` and `compute_resp_cost` outlined below. These two components are weighted according to the flow and the responsibility coefficients (as in Definition 10) taken as input. Finally, the new node (i.e., the tuple $\langle \varphi', m', \mathscr{C}'_{\varphi}, \mathscr{R}^{\varphi'}\rangle$), is added to the priority queue $Q$ storing the frontier.

We assume the process net, from which the SN is derived, has a single source and a single sink place; moreover it is *sound* [19], that is, the following conditions hold: $(i)$ the final marking of the net is reachable from the initial marking, $(ii)$ there exists a unique final marking containing one token in the sink place and no tokens in all other places, $(iii)$ no transition is dead with respect to the initial marking, that is, each transition belongs to a path from source to sink. The sequential Petri Net corresponding to the trace is sound by construction, being a linear sequence of transitions. Therefore, the Synchronous Product obtained by two sound nets is, in turn, sound.

It is worth noting that our algorithm can handle cases where the process model contains loops. The termination of the algorithm is in fact guaranteed by the control at line 13, where the current alignment $\varphi$ is extended only when its length is less that the number of transitions ($|T|$) in the synchronous product SN. The rationale is that any optimal alignment cannot be but a path in SN from the source to the sink where each transition is fired at most once. In fact, when we look for an optimal alignment in the SN, we can safely ignore loops since they would necessarily correspond to model moves, and hence to costs.

```
1  Function compute_mismatch_cost(ℛ′, φ, t)
2      ⟨l⟩ ← log_steps(⟨t⟩)
3      ⟨n⟩ ← net_steps(⟨t⟩)
4      σ′ = log_steps(φ)
5      if l ==≫ then
6          ℛ_U ← {⟨R(x, u, v), 𝒱⟩ ∈ ℛ′ | u/σ′ = ⊤, v/σ′ = 0}
7          ℛ′_U ← {⟨R(x, u, v), 𝒱⟩ ∈ ℛ′ | u/(σ′ • n) = ⊤, v/(σ′ • n) = 0}
8          if ℛ′_U \ ℛ_U ≠ ∅ then
9              return mismatch_cost
10         end
11     end
12     if n ==≫ then
13         return mismatch_cost
14     end
15     return 0
16 end

17 Function compute_resp_cost(ℛ, ℛ′, φ, t)
18     ⟨l⟩ ← log_steps(⟨t⟩)
19     σ′ = log_steps(φ)
20     𝒞 ← 0
21     ℛ_U ← {⟨R(x, u, v), 𝒱⟩ ∈ ℛ | u/σ′ = ⊤, v/σ′ = 0}
22     ℛ′_U ← {⟨R(x, u, v), 𝒱⟩ ∈ ℛ′ | u/(σ′ • l) = ⊤, v/(σ′ • l) = 0}
23     for ⟨R, 𝒱⟩ ∈ ℛ′_U \ ℛ_U do
24         𝒞 ← 𝒞 + 𝒱
25     end
26     return 𝒞
27 end
```

**Algorithm 2:** Flow (`compute_mismatch_cost`) and responsibility (`compute_resp_cost`) cost function algorithms applied at step $t$.

*Flow and responsibility alignment costs.* To compute the costs deriving from performing a transition $t$, attached to a current alignment $\varphi$, the first step is to collect the set of responsibilities associated with it. Recall that $t$ is a transition of the synchronous product SN between a log trace and the process model; thus, it consists of a pair: one element is a step from the log trace and the other is a step from the model run. For this reason, both functions `compute_mismatch_cost` and `compute_resp_cost` (Algorithm 2) take as an argument the set $\mathscr{R}^{\varphi'}$ of active responsibilities collected along the model projection of $\varphi'$. The two functions exploit two subroutines `log_steps` and `net_steps` for extracting from a sequence of transitions in SN, respectively, the trace and model projection.

Function `compute_mismatch_cost` computes the cost of transition $t$ with respect to the flow dimension. Thus, in case $t$ corresponds to either a synchronous move or a log move, the function returns, respectively, zero or the cost assigned to mismatches. In case of a model move, instead, the function checks whether performing the activity would have made some responsibilities to be neglected. This is checked by comparing the set of such responsibilities before and after performing the activity in the model move (lines 6–7). If no new responsibility results from the comparison, then the cost of the mismatch is returned (the $\gg$ does not have a justification related to neglected responsibilities). Otherwise, the returned cost (for the considered step) is zero.

Function `compute_resp_cost` computes the costs of the responsibilities which are progressed to "neglected" as a consequence of performing transition $t$. The function computes the set of such responsibilities before (line 21) and after performing $t$ (line 22). Then, since the responsibility neglected in $\mathscr{R}_U$ are also contained in $\mathscr{R}'_U$, the set difference $\mathscr{R}'_U \setminus \mathscr{R}_U$ singles out only those responsibilities neglected right by transition $t$, and the weights of these responsibilities are summed and added to the current alignment cost.

*Computational analysis.* Concerning the computational complexity, to analyze the worst case scenario we consider the case where no pruning is performed. As such, the algorithm behaves as a breadth-first search, whose search space enumerates at most $b^{(d+1)} - 1$ nodes, where $b$ is the branching factor and $d$ is the depth of the search tree.

At each step, our algorithm selects one among the enabled transitions. Being $T$ the set of transitions in the SN, the upper bound for the branching factor is $|T|$. The upper bound for the depth of the search tree is given by condition at line 13 of Algorithm 1, limiting the maximal length of a path to $|T|$ as well. So, the worst case complexity of the search space is $\mathcal{O}(|T|^{|T|+1} - 1)$. However, considering the way the SN is constructed (from the sequential Petri Net corresponding to the log trace), we can say that the branching factor is in general much less than $|T|$. Let us denote it with $n$, being $n \ll |T|$. This gives the complexity $\mathcal{O}(n^{|T|+1} - 1)$.

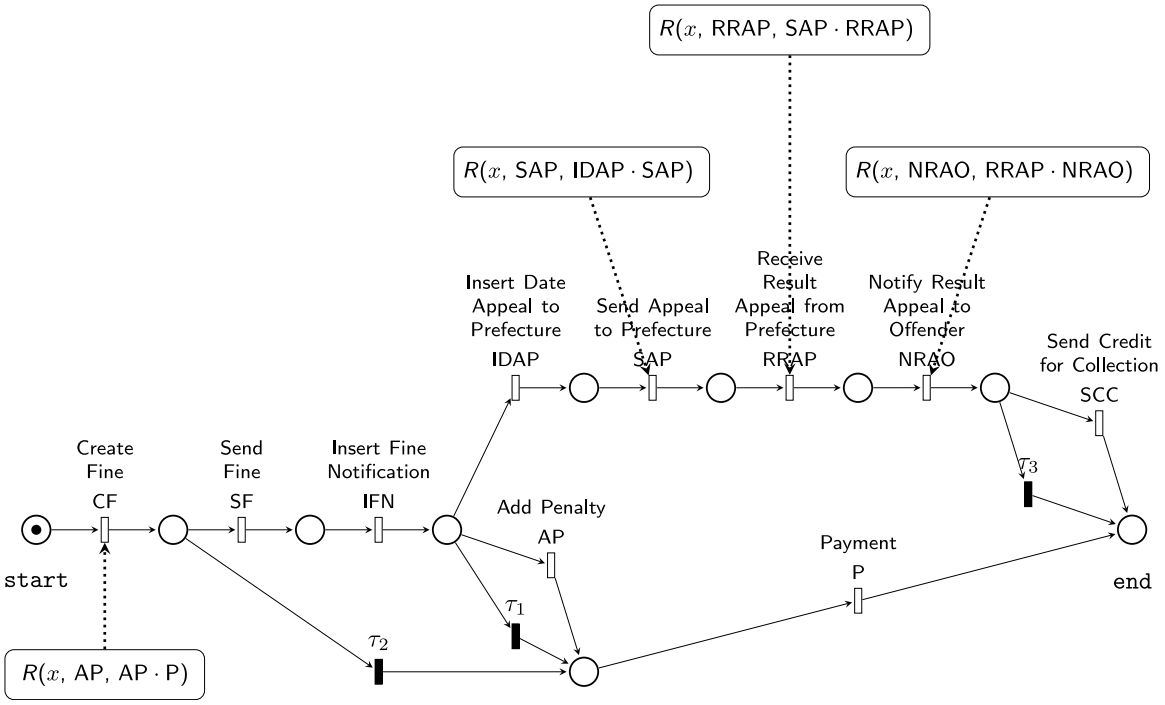


**Fig. 3.** The Petri Net of the Fine Management Scenario used as a reference model.

Additionally, at each step, the algorithm computes the residual of the responsibilities. This consists in residuating context and task conditions with respect to the log trace, for each responsibility. Let us denote with $m$ the number of responsibilities defined in the model (the worst case, indeed, is when all responsibilities of a model become active since the first event). The residual is performed with respect to the log trace, that, in the worst case, has length $|T|$. So, the cost of responsibilities evaluation is, at each step, $\mathcal{O}(m \cdot |T|)$. Combining this with the search complexity, we obtain that the overall complexity is in $\mathcal{O}(m \cdot |T| \cdot (n^{|T|+1} - 1))$.

## 6. The fine management example

We implemented the algorithm described in Section 5 in Python leveraging on the PM4Py library [20]. We implemented the responsibility progression in the rule based language CLIPS [21] via the Python library clipspy [22]. In this section, we exemplify the computation of the optimal alignments with responsibilities in the scenario of the Traffic Fine Management Process [9]. We compare, in qualitative terms, our results with a classical approach (based on the Dijkstra algorithm and available in the PM4Py library).[1]

*Process model.* The process model that we consider for the Fine Management example is shown in Fig. 3. As depicted, the fine is first created (Create Fine – CF). Then, either a payment occurs (Payment – P) or the fine is sent (Send Fine – SF) and the notification to the offender is recorded into the system (Insert Fine Notification – IFN). At this point, we consider three possible alternative evolutions: in one case, the offender starts an *appeal* (upper branch in figure); in another case, a penalty has to be added, for instance because of a late payment (Add Penalty – AP) and then a payment occurs; in the last alternative, a payment is done. In the first case, i.e., dealing with an appeal, first the date of the appeal is inserted in the system (Insert Date Appeal to Prefecture – IDAP) and then the appeal is sent to the prefecture (Send Appeal to Prefecture – SAP). After receiving a result of the appeal (Receive Result Appeal from Prefecture – RRAP), the result is notified to the offender (Notify Result Appeal to Offender – NRAO). Finally, depending on the result, the process either ends or an appointed body is sent for the credit collection (Send for Credit Collection – SCC).

The model shown in Fig. 3 has been synthesized by considering both the available textual description of the procedure and the event log, from which possible correct executions have been identified. We have then annotated the resulting process model with responsibilities that highlight salient dependencies among activities, e.g., the information/documents produced by an activity may be required for another one.

Each responsibility is attached to an activity. The rationale is that an executor of the process knows (since the beginning) that some activities, when performed, come along with a number of responsibilities that will affect his/her behavior (provided that people

[1] The implementation of the algorithm and of the example can be found at https://di.unito.it/dke.

**Table 1**
Summary of the responsibilities.

| Responsibility | Attached to | Intuitive Reading |
|---|---|---|
| $R(x, \mathsf{AP}, \mathsf{AP} \cdot \mathsf{P})$ | Create Fine | The responsibility becomes active when the fine is created. Captures that in the context in which a penalty is added (AP), the penalty has to be added only if a payment has not yet occurred. |
| $R(x, \mathsf{SAP}, \mathsf{IDAP} \cdot \mathsf{SAP})$ | Send Appeal to Prefecture | This responsibility releases the executor of the process from sending the appeal to the prefecture if no date of the appeal has been inserted (i.e., if no appeal procedure has started yet). |
| $R(x, \mathsf{RRAP}, \mathsf{SAP} \cdot \mathsf{RRAP})$ | Send Appeal to Prefecture | To receive the result of an appeal, then the appeal must have been sent to the prefecture. The responsibility releases the executor from receiving the result of the appeal (i.e., registering the information about the result) if the appeal has not been previously sent. |
| $R(x, \mathsf{NRAO}, \mathsf{RRAP} \cdot \mathsf{NRAO})$ | Notify Result Appeal to Offender | An offender can be notified of the result of an appeal only if this has occurred previously. Therefore, this responsibility releases the executor from notifying the offender if the result has not been previously performed. |

will behave in accordance with the taken responsibilities). For convenience, we list them and provide an intuitive interpretation in Table 1. We assume responsibilities to have all the same weight, set to one. The responsibility $R(x, \mathsf{AP}, \mathsf{AP} \cdot \mathsf{P})$ is attached to the activity Create Fine (CF). This means that, according to the model, when CF is performed, the responsibility is activated, and hence collected in the responsibility set. Hence, the algorithm will check, at each step, the progression of such a responsibility. In particular, the algorithm will check, in case activity add penalty (AP) is performed, whether payment (P) has not occurred yet, otherwise the responsibility progresses to neglected.

The other three responsibilities that we specify appear on the branch dealing with a request of appeal by the offender. As a difference to the previous responsibility, these three responsibilities are meant to capture a behavior of the kind: "an activity can be performed only if another one has been performed before". This can be used to capture dependencies where an information is produced by an activity and required by another one to be executed. This behavior can be captured by specifying context and task conditions differently from the responsibility described before. Specifically, the three responsibilities that we consider in the branch dealing with an appeal are: $R(x, \mathsf{SAP}, \mathsf{IDAP} \cdot \mathsf{SAP})$, $R(x, \mathsf{RRAP}, \mathsf{SAP} \cdot \mathsf{RRAP})$, and $R(x, \mathsf{NRAO}, \mathsf{RRAP} \cdot \mathsf{NRAO})$. As an example, let us consider the responsibility $R(x, \mathsf{RRAP}, \mathsf{SAP} \cdot \mathsf{RRAP})$, stating that the result of an appeal (RRAP) can be received only if the appeal has been previously sent (SAP). If, for some reason, SAP does not occur in reality the responsibility of performing RRAP only after SAP, releases the executor of performing RRAP nevertheless. This would instead be the expectation of classical approaches, which, trying to minimize the number of mismatches, prefer an execution where RRAP is performed even without SAP. The other two responsibilities are of a similar kind.

*Alignments.* On the traffic management event log [9] we analyzed 131 variants. For each variant we compute the set of optimal solutions obtained applying our approach and compare it with the solution given by the classical approach. To avoid introducing biases, we attribute to each responsibility a weight equals to one (i.e., a neglected responsibility costs as a flow misalignment), and set both coefficients $\gamma$ and $\delta$ in Definition 10 to one, so that the alignment cost is just the sum of the number of neglected responsibilities and flow misalignments.

We observed that for 115 variants the set of solutions found by our approach would be optimal also applying the classical approach. This we determine by counting the number of misalignments in our optimal solutions and obtaining that this is the same as the number of mismatches in the classical optimal solution.

For 10 variants, instead, the set of solutions we find would be optimal also in the classical approach, but there is at least one optimal solution in the classical approach which is not optimal in our approach (i.e., is not included in our set of solutions). This is possible because of some neglected responsibility which make the solution not to be optimal in our approach. Let us comment one example. Given the trace log variant $\langle \mathsf{CF}, \mathsf{SF}, \mathsf{IFN}, \mathsf{IDAP}, \mathsf{AP}, \mathsf{SAP} \rangle$, the classical approach finds as an alignment the following one:

| CF | SF | IFN | IDAP | AP | SAP | ≫ |
|---|---|---|---|---|---|---|
| CF | SF | IFN | ≫ | AP | ≫ | P |

The cost of this alignment in the classical approach is therefore three.

In our approach, given the responsibilities as specified in Fig. 3, the same alignment costs four; in fact, responsibility $R(x, \mathsf{AP}, \mathsf{AP} \cdot \mathsf{P})$ gets neglected. Therefore, this solution is discarded by our approach. The solution found by our algorithm is instead the following:

| CF | SF | IFN | IDAP | AP | SAP | ≫ | ≫ |
|---|---|---|---|---|---|---|---|
| CF | SF | IFN | IDAP | ≫ | SAP | RRAP | NRAO |

The alignment above contains three skips and therefore is an optimal solution also in the classical approach. Also in our approach the cost of the solution is three, but this is due to a neglected responsibility and to a flow alignment cost of two. In our approach,

indeed, the last skip is justified by the responsibility $R(x, \mathsf{NRAO}, \mathsf{RRAP} \cdot \mathsf{NRAO})$ (stating that a notification has to be sent to the offender only if the result of the appeal has been received). As in the previous case, however, the responsibility $R(x, \mathsf{AP}, \mathsf{AP} \cdot \mathsf{P})$ is neglected (since P is never performed).

In other 16 cases, the solution found by the classical approach is among the solutions found by our algorithm, but our optimal solutions also include alignments which are not optimal in the classical approach. This basically happens when responsibilities justify some mismatches. As an example, let us consider the log trace variant $\langle \mathsf{CF}, \mathsf{IDAP}, \mathsf{SF}, \mathsf{SAP} \rangle$. The trace clearly contains activities from the branch dealing with a request of appeal, however the solution found by the classical approach considers events IDAP and SAP as mistakes, and matches the trace with the model run where the creation of the fine is followed by its payment. The alignment is the following:

| CF | IDAP | SF | SAP | ≫ |
|---|---|---|---|---|
| CF | ≫ | ≫ | ≫ | P |

Another possible alignment, of equivalent cost in the classical approach, is the one where after the fine is created it is sent to the offender and then it is paid.

| CF | IDAP | SF | SAP | ≫ | ≫ |
|---|---|---|---|---|---|
| CF | ≫ | SF | ≫ | IFN | P |

Indeed, these two model runs are possible explanation of the log trace and both are among the optimal solutions both in the classical and in our approach.

However, among the classical solutions there is no alignment considering the model branch for raising an appeal. One can notice that this model run is the longest one; moreover, the activities along this branch depend on one another. Therefore, the missing of one event in the trace may cause few subsequent events to miss as well, thus increasing the cost of the alignment. The classical approach, in fact, penalizes longer model runs over shorter ones even though they are plausible. Responsibilities, by justifying the absence of events, allows the search for alignments in areas of the search space neglected by the classical approach. Specifically, out approach finds the following solution:

| CF | ≫ | ≫ | IDAP | SF | SAP | ≫ | ≫ |
|---|---|---|---|---|---|---|---|
| CF | SF | IFN | IDAP | ≫ | SAP | RRAP | NRAO |

In this case, the activity Send Fine is misplaced and the activity Insert Fine Notification is missing. However, the Insert Date Appeal to Prefecture from the model finds a correspondence in the trace, as well as the activity Send Appeal to Prefecture. After this a Receive Result Appeal from Prefecture is expected to occur, but is not in the trace and there is no responsibility justifying its absence. Next, Notify Result Appeal to Offender is expected to occur, but since no result has been received, there is a responsibility justifying its absence.

## 7. The reimbursement process

We, now, apply the proposal to the Domestic Trips Refunding Process [23] of the Eindhoven University of Technology (TU/e). The process and the dataset have been proposed as the BPI challenge 2020.[2] The peculiarity of this scenario is that, in some cases, the log traces have several possible (optimal) alignments. That is, their interpretation is highly ambiguous. In such a case, our approach is able to provide the user with all the possible optimal alignments, together with the sets of satisfied/neglected responsibilities, so she/he can gain a better understanding of what has actually happened and what should have happened.

*Process model.* The process concerns the submission and approval steps of a reimbursement procedure. After the declaration has been *submitted by an employee* (Sub_Empl), the process requires an *approval by the administration* (Appr_Admin), followed by an *approval by the budget owner* (Appr_BO) and then by the *supervisor* (Appr_Superv). If these two figures coincide, then only the latter is required. For each approval, a corresponding reject activity is foreseen as alternative to the approval (Rej_Admin, Rej_BO, Rej_Superv). In case of rejection, the employee withdraws the request by performing a *reject* (Rej_Emp) of the declaration (she/he could then resubmit the request again). In case all approvals are granted, first a *payment is requested* (PR), then the *payment is handled* (PH), and then the process ends. We enrich this specification by defining the following responsibilities.

**R1:** $R(x, \mathsf{Appr\_Superv}, \mathsf{PR} \cdot \mathsf{PH})$ captures that, in the context in which the approval by the supervisor is obtained (Appr_Superv), then the payment is handled (PH) if and *only if* the payment has been requested beforehand (PR). The responsibility is associated with the activity Appr_Admin, thus becomes active after (and only in case) the approval by the Administration is executed.

[2] https://icpmconference.org/2020/bpi-challenge/

**R2:** $R(x, \mathsf{Appr\_Superv}, \mathsf{Appr\_Admin} \cdot \mathsf{Appr\_Superv})$ becomes active when the activity Appr_Superv is performed. This responsibility captures that, in the context in which the supervisor approval is performed (Appr_Superv), then this is done after the approval by the administration (Appr_Admin). This requirement is in agreement with the process control flow. Additionally, the responsibility captures the requirement that *only if* the approval by the administration is obtained then the approval by the supervisor is expected. If the former activity is not observed in an event log, then the latter is justified not to be performed (and thus to be missing in the log).

**R3:** $R(x, \mathsf{Rej\_Superv}, \mathsf{Appr\_Admin} \cdot \mathsf{Rej\_Superv})$ which is associated with the activity Rej_Superv. This responsibility is similar to the previous one, but refers to the rejection from the supervisor. It captures that if (and only if) the rejection from the supervisor is performed, then this is done after the approval by the administration.

**R4:** $R(x, \mathsf{PH}, \mathsf{Appr\_Superv} \cdot \mathsf{PH})$ which is associated with the activity *Payment Handled* (PH). This responsibility requires the approval by the supervisor and the payment handling to be performed in this order. As such, if the approval by the supervisor is not observed in an execution, the handling of the payment is justified not to be performed. On the contrary, in those cases where the payment handling is performed without having obtained the authorization by the supervisor, the responsibility is neglected.

*Alignments.* The available event log includes 97 variants. Among these, for 88 variants our approach finds solutions that would be optimal also by applying the classical approach (i.e., the optimal alignment that we obtain has the same number of skips than the classical optimal solution).[3] Actually, in 9 of these variants, the classical optimal alignment is also among the solutions returned by our algorithm. Interestingly, for all these 9 variants, our algorithm returns more solutions than the classical one (recall that our algorithm returns all optimal solutions). Overall, we obtain 218 solutions for the 97 variants, thus obtaining 121 solutions more than those found by the classical approach (that stops after the first optimal one).

However, comparing the two approaches from a quantitative point of view does not clarify how responsibilities affect the search of optimal alignments. Thus, as in the previous case, we are interested in a qualitative analysis of the results produced by our algorithm, and in the influence of responsibilities on the optimal alignments. To this aim, we consider some variants and comment the obtained results.

As a first example, let us consider the log variant: $\langle \mathsf{Sub\_Empl}, \mathsf{Rej\_Pre\_Appr}, \mathsf{Rej\_by\_Missing} \rangle$. In this execution, after the declaration is submitted by the Employee, the request is first rejected by a "Pre Approver" (Rej_Pre_Appr) and then "by Missing" (Rej_by_Missing). These two last activities are not part of the process: they are not mentioned in the description of the BPI challenge, nor in the process described in [23]. Thus, these two events trace some unexpected situation during the process execution. This does not represent a problem for the alignment algorithms, but it is indicator of high uncertainty about what the expected execution should have been. The classical optimal solution does not capture such an uncertainty; it adopts a too simplistic approach by selecting the shortest execution to the end: it concludes that the declaration has been rejected by the administration and then by the employee as follows

| Sub_Empl | Rej_Pre_Appr | Rej_by_Missing | ≫ | ≫ |
|---|---|---|---|---|
| Sub_Empl | ≫ | ≫ | Rej_Admin | Rej_Empl |

Responsibilities, instead, better capture such an ambiguity via the possible justification of model moves, that allows our algorithm to keep open alternative paths. In the case specified above, we obtain alignments going from 4 misalignments (as in the classical case) to 6. For instance, a possible solution is the one where the declaration should have been approved by the administration, and then rejected by the supervisor, as follows:

| Sub_Empl | Rej_Pre_Appr | Rej_by_Missing | ≫ | ≫ | ≫ |
|---|---|---|---|---|---|
| Sub_Empl | ≫ | ≫ | Appr_Admin | Rej_Superv | Rej_Empl |

In the classical approach, the cost of this solution would be 5 and thus would be excluded from the set of possible optimal solutions. However, given responsibility **R3**, since the activity Appr_Admin is missing in the log, then activity Rej_Superv is justified to be missing, as well. Its appearance in the log would indeed violate the responsibility. The fact of obtaining such a variety of solutions is a direct consequence of the uncertainty given by the variant itself: it is not clear what the expected model execution should be when only one activity (the first one in the example) matches.

Comparing the solutions obtained by the two approaches, we also notice that in 29 cases the solutions are actually the same, but our solution attributes a higher cost. This happens when some responsibilities are neglected, thus increasing the cost of a solution. For instance, both algorithms match the log variant $\langle \mathsf{Sub\_Empl}, \mathsf{Appr\_Superv}, \mathsf{RP}, \mathsf{PH} \rangle$ with the same run in the process model where, immediately after Sub_Empl, the approval by the administration is obtained. Although the matching process run is the same, our approach penalizes this alignment by attributing it a higher cost given by the violation of responsibility **R2**. Thus, in this case, the cost of our optimal solution is 2, while for the classical approach the cost is 1. This aspect represents a valuable piece of information because a neglected responsibility signals a problem in the execution which is not signaled in the classical approach. In

[3] We set both flow and responsibility cost coefficients to one (see Definition 10), to keep the two cost sources balanced.

this case, the fact of **R2** being neglected signals that the activity Appr_Superv was performed when it should have not. A subsequent re-engineering phase would exploit such a piece of knowledge to look for the causes of the problem, and possibly fix it to avoid further occurrences.

Similar considerations can be done on other traces involving responsibilities **R1** and **R4**. For instance, there are situations where **R4** gets neglected since the payment is handled even when it has not been previously approved by supervisor. In these cases, responsibilities capture a domain semantics that is missing in the classical approach, and which improve the user's understanding.

## 8. Related works

Process alignment has been addressed in the literature from several points of view. One can distinguish approaches focusing on the perspective (e.g., control flow, multi-perspective), on the paradigm (e.g., declarative, procedural or mixed-paradigm approaches) or on the technique (such as automata or Petri Nets based).

Many proposals that focus on the efficiency of alignment computation rely on Petri Nets. They consider the Synchronous Product Net obtained from model and execution nets, and search for the optimal alignment as a firing sequence from the initial to a final marking. In general, the shortest path search techniques rely on Dijkstra- or A*-based algorithms [5,24]; possibly, pruning techniques are applied to speed up the overall search [1,5]. Approaches based on planning [6] and SAT algorithms [25] have also been proposed.

Recently, a research direction advocate that deviations from a process model are not all the same, and just considering the number of mismatches in the cost functions may lead to ambiguous conclusions. The works in [5,6,26] suggest considering the context (i.e., when and how the misalignment occurs) in the cost functions so to obtain more informative alignments. Boltenhagen et al. [5] propose to consider when a deviation in the alignment occurs with the idea that mismatches occurring at the early stages of an execution have more impact than those occurring at later stages. To this aim, they define a cost function which assigns a cost to every asynchronous move in an alignment weighted w.r.t. to the position in which it occurred. They propose an A*-based search algorithm over the synchronous product net. They also propose an alternative heuristic-based search algorithm, aiming at improving the performance at the detriment of the solution optimality.

Acitelli et al. [6] agree on the fact that deviations of an execution from a process model are not all the same, since they may have different consequences, and thus should have different costs in the process alignment computation. Beside considering the position in which a mismatch occurs, they also consider which activities occurred before and after that mismatch. They propose a planning algorithm to find the cheapest path on a deterministic finite state automaton which consists of both synchronous and asynchronous moves. For each move, they assign a cost. The idea proposed in [26] is to favor alignments where the number of synchronous moves is maximized. This is achieved by defining a cost function which penalizes log moves only. Additionally, they also consider milestone activities, where the process model and the execution must agree. To this purpose, the cost of a model move for these activities is set to infinite.

In the multi-perspective class, the attention is directed on how additional perspectives, other than control flow, can be considered in the search for an optimal alignment. Typical perspectives are, e.g., costs, resources, data [27–29], and time constraints [15]. In [30] the authors propose a cost function which, similarly to our proposal, combines the costs from different perspectives. Customizing how they are combined, one gives more importance to a perspective over another. Basically, all these approaches complements a Petri-Net, capturing the control flow, with additional perspectives that are then used by dedicated search algorithms. Multi-Perspective Declare (MP-Declare) [31], instead, starts from a declarative representation of the process, enriched with constraints capturing, besides the control flow, also time and data dimensions. MP-Declare semantics is given in terms of Metric First-Order Linear Temporal Logic, and new algorithms are provided for assessing the value of a constraint given a trace.

Approaches in the mixed-paradigm class model their processes as a combination of procedural and declarative formalisms. In [32], for instance, Petri-Nets (i.e., the procedural part), are used to model the "sunny day" scenario. Declare constraints are instead used to capture activities that may be performed at any time, or conditions that may change over time.

To the best of our knowledge, no existing approaches consider responsibilities as a further dimension of a process model. In our proposal, we consider it in two ways: as an additional perspective compared to the control flow and to evaluate model moves. Similarly to [26], we propose a strategy for not counting the cost of model moves. However, while in [26] model moves are ignored systematically, our approach considers responsibilities as a context: a model move is not a cost when the corresponding synchronous move would lead to neglect some responsibilities. Moreover, the approaches in [5,26] define a general cost function which is domain independent. In our approach we support the definition of a cost function tailored on each responsibility. In this aspect, our proposal is more in line with the approach in [6], where misalignment costs can be explicitly associated with mismatches. As a difference, misalignment costs are determined by considering both the control flow and the responsibilities, that are declarative in nature.

## 9. Conclusions and future work

In this paper, we have pointed out how responsibilities can be exploited in the task of process alignment. In particular, we have discussed and formalized an approach for process alignment where the role of responsibilities is twofold. First, they provide useful insights for interpreting correctly the detected mismatches between a model run and a log trace. By means of responsibilities, in fact, one can distinguish mismatches that correspond to errors (i.e., missing activities), from mismatches that are coherent with the job description of a given role (i.e., activities that have been skipped on purpose).

Second, responsibilities can be satisfied or neglected by the activities performed (or not) by the executors of a process. Neglected responsibilities are a symptom of something unexpected occurred during a log trace, and hence they represent a cost as well as mismatches.

We have therefore proposed an alignment cost function that captures both the cost due to the flow alignment (i.e., the number of mismatches), and the cost due to the neglected responsibilities. Such a cost function is used to drive a branch-and-bound algorithm that looks for all the optimal alignments given a process model and a trace log. The algorithm and its implementation show the feasibility of the approach and allows us to qualitatively evaluate the produced solutions (also in comparison with the classical approach). As a future work, we will consider how to improve the algorithm performance without degrading the quality of the found solutions. The approach based on Integer Linear Programming proposed in [33] could provide some useful hint since it starts from the synchronous product, as in our case, and aims at finding (possibly suboptimal) solutions without exploring explicitly the model behavior.

The integration of responsibilities in the process model opens several future directions. First, responsibilities may play a role in increasing the understanding of human users. In fact, each alignment found by our algorithm is associated with a set of met and unmet responsibilities which model the context (i.e., a justification) of the alignment itself. In addition, by considering the set of unmet responsibilities for a number of log traces, one could reason about possible inefficiencies and flaws in the process model. This would enable a responsibility-driven procedure for re-engineering a process, where the responsibilities themselves could be redefined.

Finally, it is interesting to note that the notion of responsibility is often related to that of accountability in agent organizations [34]. Specifically, accountability can be seen as a relationship between two actors, in which one is responsible for a given job (accountability giver), and the other has some interest in that job (e.g., for her decision process), and hence has the faculty of asking accounts about the very same job to the first actor. Thus, through accountability, it may be possible to establish feedback channels that are orthogonal to a process model, and that can be exploited to gain a better understanding of what is actually going on within an organization. An interesting future direction, thus, is to complement our responsibility framework with accountability relationships as a way for improving both the computation of alignments and their understanding in the context of a business organization.

## CRediT authorship contribution statement

**Matteo Baldoni:** Writing – review & editing, Writing – original draft, Supervision, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Cristina Baroglio:** Writing – review & editing, Writing – original draft, Supervision, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Elisa Marengo:** Writing – review & editing, Writing – original draft, Supervision, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Roberto Micalizio:** Writing – review & editing, Writing – original draft, Supervision, Software, Methodology, Investigation, Formal analysis, Conceptualization.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data will be made available on request.

## Acknowledgments

This work is part of the projects NODES, which has received funding from the MUR-M4C2 1.5 of PNRR with grant agreement no. ECS00000036.

## References

[1] J. Carmona, B.F. van Dongen, A. Solti, M. Weidlich, Conformance Checking - Relating Processes and Models, Springer, 2018.
[2] Project Management Institute, A Guide to the Project Management Body of Knowledge (PMBOK guide), Project Management Institute, Inc., 2021.
[3] W.M.P. van der Aalst, A. Adriansyah, B.F. van Dongen, Replaying History on Process Models for Conformance Checking and Performance Analysis, WIREs Data Min. Knowl. Discov. 2 (2) (2012) 182–192.
[4] R.P.J.C. Bose, W.M.P. van der Aalst, Process Diagnostics Using Trace Alignment: Opportunities, Issues, and Challenges, Inf. Syst. 37 (2) (2012) 117–141.
[5] M. Boltenhagen, T. Chatain, J. Carmona, A Discounted Cost Function for Fast Alignments of Business Processes, in: BPM, in: LNCS, vol. 12875, Springer, 2021, pp. 252–269.
[6] G. Acitelli, M. Angelini, S. Bonomi, F.M. Maggi, A. Marrella, A. Palma, Context-Aware Trace Alignment with Automated Planning, in: ICPM, IEEE, 2022, pp. 104–111.
[7] OMG, BMM Model, 2015, https://www.omg.org/spec/BMM/1.3/PDF.
[8] M. Baldoni, C. Baroglio, E. Marengo, R. Micalizio, A Responsibility Framework for Computing Optimal Process Alignments, in: Proc. of 7th International Workshop on Artificial Intelligence for Business Process Management, AI4BPM, in: LNBIP, vol. 492, 2023, pp. 5 – 17.
[9] M. de Leoni, F. Mannhardt, Road Traffic Fine Management Process, 2015, https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249/1.
[10] B. van Dongen, BPI Challenge 2020, 2020, https://data.4tu.nl/collections/_/5065541/1.
[11] N.A. Vincent, A Structured Taxonomy of Responsibility Concepts, in: Library of Ethics and Applied Philosophy, vol. 27, 2011, pp. 15–35.

[12] M. Baldoni, C. Baroglio, O. Boissier, R. Micalizio, S. Tedeschi, Accountability and Responsibility in Multiagent Organizations for Engineering Business Processes, in: Engineering Multi-Agent Systems - 7th International Workshop, EMAS 2019, Revised Selected Papers, in: LNCS, vol. 12058, Springer, 2019, pp. 3–24.
[13] A. Bauer, M. Leucker, C. Schallhart, Runtime Verification for LTL and TLTL, ACM Trans. Softw. Eng. Methodol. 20 (4) (2011).
[14] G. De Giacomo, R. De Masellis, M. Grasso, F.M. Maggi, M. Montali, Monitoring Business Metaconstraints Based on LTL and LDL for Finite Traces, in: Business Process Management, in: LNCS, vol. 8659, Springer, 2014, pp. 1–17.
[15] G. Di Federico, A. Burattin, Do You Behave Always the Same? - A Process Mining Approach, in: ICPM Workshops, in: LNBIP, vol. 468, Springer, 2022, pp. 5–17.
[16] M.P. Singh, M.N. Huhns, Service-Oriented Computing - Semantics, Processes, Agents, Wiley, 2005.
[17] M. Baldoni, C. Baroglio, R. Micalizio, S. Tedeschi, Accountability in Multi-Agent Organizations: From Conceptual Design to Agent Programming, Autonomous Agents Multi Agent Syst. 37 (1) (2023) 7.
[18] A. Giua, F. DiCesare, Supervisory Design Using Petri Nets, in: [1991] Proceedings of the 30th IEEE Conference on Decision and Control, vol. 1, 1991, pp. 92–97.
[19] W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, M.T. Wynn, Soundness of Workflow Nets: Classification, Decidability, and Analysis, Formal Aspects Comput. 23 (3) (2011) 333–363.
[20] A. Berti, S. van Zelst, D. Schuster, PM4Py: A Process Mining Library for Python, Softw. Impacts 17 (2023) 100556.
[21] Clips: A Tool for Building Expert Systems, https://www.clipsrules.net/.
[22] Clipspy: Clips Python Bindings, https://clipspy.readthedocs.io/.
[23] D. Bano, M. Völker, S. Remy, H. Leopold, M. Weske, Multi-Perspective Analysis of Approval Processes based on Multiple Event Logs, in: Tenth Int'L Business Process Intelligence Challenge, BPIC 20, 2020.
[24] A. Adriansyah, Aligning Observed and Modeled Behavior (Ph.D. thesis), Technische Universiteit Eindhoven, 2014.
[25] M. Boltenhagen, T. Chatain, J. Carmona, Model-Based Trace Variant Analysis of Event Logs, Inf. Syst. 102 (2021) 101675.
[26] V. Bloemen, S.J. van Zelst, W.M.P. van der Aalst, B.F. van Dongen, J. van de Pol, Aligning Observed and Modelled Behaviour by Maximizing Synchronous Moves and Using Milestones, Inf. Syst. 103 (2022) 101456.
[27] M. Alizadeh, X. Lu, D. Fahland, N. Zannone, W.M.P. van der Aalst, Linking Data and Process Perspectives for Conformance Analysis, Comput. Secur. 73 (2018) 172–193.
[28] P. Felli, A. Gianola, M. Montali, A. Rivkin, S. Winkler, CoCoMoT: Conformance Checking of Multi-perspective Processes via SMT, in: Business Process Management, in: LNCS, vol. 12875, Springer, 2021, pp. 217–234.
[29] A.S. Mozafari Mehr, R.M. de Carvalho, B.F. van Dongen, Detecting Complex Anomalous Behaviors in Business Processes: A Multi-perspective Conformance Checking Approach, in: ICPM Workshops, in: LNBIP, vol. 468, Springer, 2022, pp. 44–56.
[30] F. Mannhardt, M. de Leoni, H.A. Reijers, W.M.P. van der Aalst, Balanced Multi-Perspective Checking of Process Conformance, Computing 98 (2016).
[31] A. Burattin, F.M. Maggi, A. Sperduti, Conformance Checking Based on Multi-Perspective Declarative Process Models, Expert Syst. Appl. 65 (2016) 194–211.
[32] B.F. van Dongen, J. De Smedt, C. Di Ciccio, J. Mendling, Conformance Checking of Mixed-Paradigm Process Models, Inf. Syst. 102 (2021) 101685.
[33] B. van Dongen, J. Carmona, T. Chatain, F. Taymouri, Aligning Modeled and Observed Behavior: A Compromise Between Computation Complexity and Quality, in: Advanced Information Systems Engineering, Springer, 2017, pp. 94–109.
[34] M. Baldoni, C. Baroglio, R. Micalizio, S. Tedeschi, Robustness Based on Accountability in Multiagent Organizations, in: Proceedings of the Autonomous Agents and MultiAgent Systems, AAMAS 21, 2021, pp. 142–150, IFAAMAS.

**Matteo Baldoni** is a full professor at the University of Torino. He got a Ph.D. in Computer Science and his research interests are artificial intelligence, multiagent systems, and business processes.

**Cristina Baroglio** is an associate professor of computer science at the University of Torino, Italy, since 2005. She has been working in the research field of artificial intelligence since her Ph.D. studies. In the last twenty years she has been investigating several topics in the area of multi-agent systems, with a specific interest on coordination.

**Elisa Marengo** got her Ph.D. in Computer Science from the University of Torino, where she currently work as Researcher. Her main research interests concern modeling and reasoning in the field of Business Process Management, Process Mining and Predictive Process Monitoring.

**Roberto Micalizio** received his Master's degree and his PhD in Computer Science from the University of Turin, Italy in 2004 and 2007, respectively. He is currently an Associate Professor at the same university. His research interests include Multi-Agent Systems, Model-Based Reasoning and Planning.