Aligning Event Logs and Declarative Process Models for Conformance Checking

Massimiliano de Leoni*, Fabrizio Maria Maggi, and Wil M. P. van der Aalst

Eindhoven University of Technology, Eindhoven, The Netherlands
{m.d.leoni,f.m.maggi,w.m.p.v.d.aalst}@tue.nl

Abstract. Process mining can be seen as the "missing link" between data mining and business process management. Although nowadays, in the context of process mining, process discovery attracts the lion's share of attention, conformance checking is at least as important. Conformance checking techniques verify whether the observed behavior recorded in an event log matches a modeled behavior. This type of analysis is crucial, because often real process executions deviate from the predefined process models. Although there exist solid conformance checking techniques for procedural models, little work has been done to adequately support conformance checking for *declarative models*. Typically, traces are classified as fitting or non-fitting without providing any detailed diagnostics. This paper aligns event logs and declarative models, i.e., events in the log are related to activities in the model if possible. The alignment provides then sophisticated diagnostics that pinpoint where deviations occur and how severe they are. The approach has been implemented in ProM and has been evaluated using both synthetic logs and real-life logs from Dutch municipalities.

1 Introduction

Traditional Workflow Management Systems (WFMSs) are based on the idea that processes are described by procedural languages where the completion of a task may enable the execution of other tasks. While such a high degree of support and guidance is certainly an advantage when processes are repeatedly executed in the same way, in dynamic settings (e.g., healthcare) a WFMS is considered to be too restrictive. Users often need to react to exceptional situations and execute the process in the most appropriate manner. Therefore, in these environments systems tend to provide more freedom and do not restrict users in their actions. Comparing such dynamic process executions with procedural models may reveal many deviations that are, however, not harmful. In fact, people may exploit the flexibility offered to better handle cases. In such situations we advocate the use of *declarative models*. Instead of providing a procedural model that enumerates all process behaviors that are allowed, a declarative model simply lists the constraints that specify the forbidden behavior, i.e., "everything is allowed unless explicitly forbidden".

In this paper, we focus on *conformance checking for declarative models*. Conformance checking techniques take an event log and a process model and compare the

* The research of Dr. de Leoni has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement no 257593 (ACSI).

A. Barros, A. Gal, and E. Kindler (Eds.): BPM 2012, LNCS 7481, pp. 82–97, 2012.

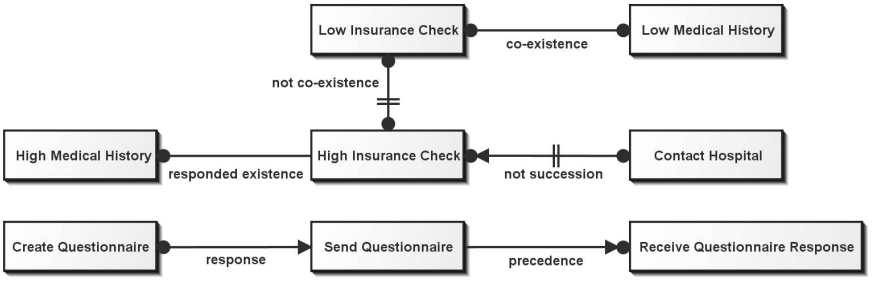


Fig. 1. Declare model consisting of six constraints and eight activities

observed behavior with the modeled behavior [3,4,5,16]. Along with *process discovery* (learning process models from event logs) and *process enhancement* (e.g., extending process models using information extracted from the actual executions recorded in event logs), conformance checking belongs to the area of *process mining* [3].

Since often the actual execution deviates from the prescriptions of theoretical models, this type of analysis is critical in many domains, e.g., process auditing [17] or risk analysis [11]. However, while there is a large stream of research about conformance checking of procedural models, little work has been conducted for declarative models. Existing approaches exclusively focus on determining whether a given process instance conforms with a given process model or not [7,6,9,1]. In this paper, we want to provide diagnostics at the event log level rather than a simple yes/no answer at the trace level. Moreover, our approach is able to pinpoint where deviations more frequently occur and how severely a process instance does not comply the process model.

There are different quality dimensions for comparing process models and event logs, e.g., *fitness*, *simplicity*, *precision*, and *generalization* [3,15,10]. In this paper, we focus on *fitness*: a model with good fitness allows for most of the behavior seen in the event log. A model has a *perfect* fitness if all traces in the log can be replayed by the model from the beginning to the end. Our approach is applied to Declare, a declarative language supported by a toolset that includes a designer, a workflow engine, a worklist handler, and various analysis tools [20,2].[1] Due to space reasons we cannot provide a detailed description of Declare and only highlight some of the most relevant features of the language through an example.

Example 1. *A travel agency has enacted a process to handle health insurance claims. A claim can be classified as high or low, depending on the amount that the customer claims to receive back. Fig. 1 shows a simple* Declare *model with some example constraints to describe this process. The model includes eight activities (depicted as rectangles, e.g.,* Contact Hospital*) and six constraints (shown as connectors between the activities). For low claims, two tasks* Low Insurance Check *and* Low Medical History *need to be executed. The* co-existence *constraint indicates that these activities always occur together (in any order). If a claim is classified as low, no activities referring to high claims can be executed and vice versa. The* not co-existence *constraint indicates that* Low Insurance Check *and* High Insurance Check *can never coexist in the same process instance. Moreover, in case of high claims, the medical history check (*High Medical History*)*

[1] Declare web site - http://www.win.tue.nl/declare/

*can only be executed together with the insurance check (*High Insurance Check*), even though they can be executed in any order. Nevertheless, it is possible to execute* High Insurance Check *without executing* High Medical History. *All this is enforced by the* responded existence *constraint. For every claim, it is also possible to contact the doctor/hospital for verification. However, in case of high claims, this cannot be done before the insurance check. This is defined by the* not succession *constraint, that means that* Contact Hospital *cannot be followed in the same process instance by* High Insurance Check. *In this process, a questionnaire is also created and eventually sent to the applicant (modeled by the* response *constraint); after having sent the questionnaire, the applicant can possibly decide whether to fill it in or not (*precedence *constraint).*

The approach we propose is based on the principle of finding an *alignment* of an event log and a process model. The concept of *alignment* has successfully been used in the context of procedural models (e.g., [4,5,12]); here, we adapt it for declarative models. Similarly to what has been proposed for procedural models, in our approach, events in the log are mapped onto executions of activities in the process model. A cost/weight is assigned to every potential deviation. We use the A* algorithm [4,8] to find, for each trace in the event log, an optimal alignment, i.e., an alignment that minimizes the cost of the deviations. The application of the A* algorithm is more challenging for declarative models than for procedural models. This is due to the fact that, since in a declarative model everything is allowed unless constrained otherwise, the set of admissible behaviors is generally far larger than the set of behaviors allowed by procedural models. This implies that the search space where to find an optimal alignment of a log and a declarative model is much larger. Therefore, for this type of models, it is essential to avoid exploring search-space nodes that certainly lead to non-optimal solutions.

In addition to simply returning an optimal alignment for each trace, we also provide the process analyst with a summary that gives a helicopter view of the conformance of the model with respect to the entire log. In particular, we aggregate the information associated to the optimal alignments and visualize the deviations upon the process model. In fact, we generate a "map" thus highlighting on a Declare model which constraints are more often violated during the performance and which activities are mostly involved in the deviations. The approach has been implemented in ProM and has been evaluated on a variety of synthetic and real-life logs.

The paper is structured as follows. Section 2 introduces event logs and Declare models. Section 3 describes the notion of alignment and how it can be used for conformance checking. Section 4 describes the application of the A* algorithm to find an optimal alignment. Here, we also introduce an optimization of the algorithm to prune large irrelevant parts of the search space (that do not lead to an optimal solution). Section 5 shows which diagnostics and feedback we can provide to the process analyst. Section 6 presents the plug-ins we have implemented and reports some performance results. Here we also validate our approach by applying it to real-life logs of Dutch municipalities. Section 7 concludes the paper highlighting future work.

2 Basic Concepts

Declare is a language that provides both an intuitive graphical representation and a formal semantics for declarative process models. In particular, Declare is grounded in Linear Temporal Logic (LTL) with a finite-trace semantics and every Declare constraint

is formally defined through an LTL formula.[2] For instance, the *response* constraint in Fig. 1 can be formally represented using LTL as $\Box(Create\ Questionnaire \rightarrow \Diamond Send\ Questionnaire)$. This means that "whenever activity *Create Questionnaire* is executed, eventually activity *Send Questionnaire* is executed". In the following, we refer to a **Declare model** $\mathcal{D} = (A, \Pi)$, where A is a set of activities and Π is a set of Declare constraints defined over activities in A.

To identify potential deviations of a log from a reference Declare model, we need to map each activity in the log onto an activity in the model. Each process instance in a log follows a trace of events and different instances may follow the same trace. Therefore, an event log can be seen as a multi-set of traces, i.e., $\mathcal{L} \in \mathbb{B}({A_L}^*)$[3], where A_L is the set of activities in the log. Since Declare is an "open" language, it allows for the execution of any activity which is not in the model. Therefore, the set of activities in the log may include all activities in the model and more, i.e., $A \subseteq A_L$.

For conformance checking, we do not need to distinguish the activities in the set $A_L \setminus A$ as they do not appear in the model. This allows us to reduce the space of the allowed behaviors. Note that we cannot completely abstract from the activities in $A_L \setminus A$ because some constraints use LTL's next operator (e.g., the chain response and chain precedence constraints). Therefore, we map all events referring to some activity in $A_L \setminus A$ to ✓. Afterwards, the log $\mathcal{L} \in \mathbb{B}({A_L}^*)$ is converted into $\mathcal{L}' \in \mathbb{B}(\Sigma^*)$ with $\Sigma = A \cup \{\checkmark\}$. We use function $\chi \in A_L \rightarrow \Sigma$ that maps each activity in A_L onto Σ. $\forall a \in A.\ \chi(a) = a$ and $\forall a \in (A_L \setminus A).\ \chi(a) = \checkmark$, i.e., every occurrence of an activity in a log trace that is also defined in the model is mapped onto the activity itself, whereas other events are mapped onto ✓.

Example 1 (cont.). *The set of activities of the Declare model in Fig. 1 is*

$$A = \langle \textit{Low Insurance Check}, \textit{Low Medical History}, \textit{High Insurance Check}, \textit{High Medical History}, \textit{Contact Hospital}, \textit{Create Questionnaire}, \textit{Send Questionnaire}, \textit{Receive Questionnaire Response} \rangle$$

Let us assume to have a log that contains the following trace:

$$\sigma_L = \langle \textit{Register}, \textit{Low Insurance Check}, \textit{Create Questionnaire}, \textit{Prepare Notification Content}, \textit{Create Questionnaire}, \textit{Send Notification by e-mail}, \textit{Send Notification by Post}, \textit{Archive} \rangle$$

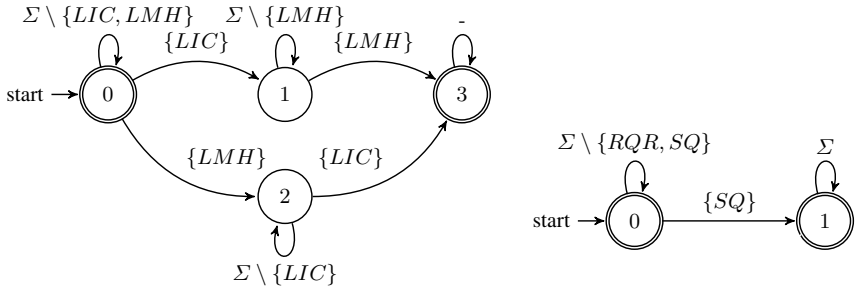
This log trace is defined over an activity set A_L that includes A. Using the mapping function χ, σ_L can be transformed into σ'_L.

$$\sigma'_L = \langle \checkmark, \textit{Low Insurance Check}, \textit{Create Questionnaire}, \checkmark, \textit{Create Questionnaire}, \checkmark, \checkmark, \checkmark \rangle$$

Note that Register, Send Notification by e-mail, Send Notification by Post, Archive are mapped onto ✓.

In the remainder, we only consider event logs after mapping unconstrained activities onto ✓, i.e., $\mathcal{L} \in \mathbb{B}(\Sigma^*)$. To check whether a log trace $\sigma_L \in \mathcal{L}$ is compliant with

[2] For compactness, in the following we will use the LTL acronym to denote LTL on finite traces.
[3] $\mathbb{B}(X)$ the set of all multi-sets over X and X^* is the set of all finite sequences over X.



(a) Constraint automaton for the co-existence constraint

(b) Constraint automaton for the precedence constraint

Fig. 2. Constraint automata for two Declare constraints in Fig. 1

a Declare constraint $\pi \in \Pi$, using the technique described in [19], we translate the corresponding LTL formula into a final-state automaton that accepts all traces that do not violate constraint π.

Definition 1 (Constraint Automaton). *Let $\mathcal{D} = (A, \Pi)$ be a Declare model, $\pi \in \Pi$ and $\Sigma = A \cup \{\checkmark\}$. The* constraint automaton $\mathcal{A}_\pi = (\Sigma, \Psi_\pi, \psi_{0_\pi}, \delta_\pi, F_\pi)$ *is the final-state automaton which accepts precisely those traces $\sigma \in \Sigma^*$ satisfying π, where:*

- *$\Sigma = A \cup \{\checkmark\}$ is the input alphabet;*
- *Ψ_π is a finite, non-empty set of states;*
- *$\psi_{0_\pi} \in \Psi_\pi$ is an initial state;*
- *$\delta_\pi \in \Psi_\pi \times \Sigma \to \Psi_\pi$ is the state-transition function;*
- *$F_\pi \subseteq \Psi_\pi$ is the set of final states.*

We create a constraint automaton $\mathcal{A}_\pi$ for every constraint $\pi \in \Pi$. These automata can be used to check the conformance of a log trace with respect to each constraint in $\mathcal{D}$.

Example 1 (cont.). *For the* co-existence *constraint and the* precedence *constraint in Fig. 1, we obtain the automata depicted in Fig. 2a and 2b. In both cases, state* 0 *is the initial state and accepting states are indicated using a double outline. A transition is labeled with the set of the activities triggering it (we use the initial letters to denote an activity, e.g., we use LIC to indicate* Low Insurance Check*). This indicates that we can follow the transition for any event included in the set (e.g., we can execute event* High Insurance Check *from state* 0 *of the* precedence *automaton and remain in the same state).*

The **process behavior set** $\mathcal{P}_\mathcal{D} \subseteq \Sigma^*$ of a Declare model $\mathcal{D} = (A, \Pi)$ is the set of traces that are accepted by all automata $\mathcal{A}_\pi$ with $\pi \in \Pi$, i.e., all process executions that comply the model $\mathcal{D}$.

3 The Conformance Checking Framework

To check the conformance of an event log $\mathcal{L}$ with respect to a Declare model $\mathcal{D}$, we adopt an approach where we search for an *alignment* of the log and the model. Such an alignment shows how the event log can be replayed on the Declare model.

An alignment relates *moves in log* and to *moves in model* as explained in the following definition. Here, we explicitly indicate *no move* with $\gg$. $\Sigma_{\gg} = \Sigma \cup \{\gg\}$, where Σ denotes the input alphabet of each constraint automaton in $\mathcal{D}$.

Definition 2 (Alignment). *A pair* $(s', s'') \in (\Sigma_{\gg} \times \Sigma_{\gg}) \setminus \{(\gg, \gg)\}$ *is*

- *a* move in log *if* $s' \in \Sigma$ *and* $s'' = \gg$,
- *a* move in model *if* $s' = \gg$ *and* $s'' \in \Sigma$,
- *a* move in both *if* $s' \in \Sigma$, $s'' \in \Sigma$ *and* $s' = s''$.

$\Sigma_A = (\Sigma_{\gg} \times \Sigma_{\gg}) \setminus \{(\gg, \gg)\}$ *is the set of the* legal moves.

The alignment *of two execution traces* $\sigma', \sigma'' \in \Sigma^*$ *is a sequence* $\gamma \in {\Sigma_A}^*$ *such that the projection on the first element (ignoring* $\gg$*) yields* σ' *and the projection on the second element yields* σ''.

In particular, if $\sigma' = \sigma_L \in \mathcal{L}$ and $\sigma'' \in \mathcal{P}_{\mathcal{D}}$, we refer to the alignment γ as a *complete alignment* of σ_L and $\mathcal{D}$. An alignment of the event log $\mathcal{L}$ and the Declare model $\mathcal{D}$ is a multi-set $\mathcal{A} \in \mathbb{B}({\Sigma_A}^*)$ of alignments such that, for each log trace σ_L, there exists an alignment $\gamma \in \mathcal{A}$ of σ_L and $\mathcal{D}$. The definition of $\mathcal{A}$ as a multi-set is motivated by the fact that an event log can contain the same log trace σ_L multiple times and, hence, the same alignment can be given for all its occurrences.

Example 1 (cont.). *Given the log trace* $\sigma_L = \langle \checkmark, LIC, CQ, \checkmark, CQ, \checkmark, \checkmark, \checkmark \rangle$*, there are many possible complete alignments of* σ_L *and the* Declare *model in Fig. 1. For instance, the following are valid complete alignments:*

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \textbf{L:} & \checkmark & LIC & CQ & \checkmark & CQ & \gg & \checkmark & \checkmark & \checkmark \\ \hline \textbf{P:} & \checkmark & \gg & CQ & \checkmark & CQ & SQ & \checkmark & \checkmark & \checkmark \\ \hline \end{array} \quad \gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \textbf{L:} & \checkmark & LIC & \gg & CQ & \checkmark & CQ & \checkmark & \checkmark & \checkmark \\ \hline \textbf{P:} & \checkmark & LIC & LMH & \gg & \checkmark & \gg & \checkmark & \checkmark & \checkmark \\ \hline \end{array}$$

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline \textbf{L:} & \checkmark & LIC & \gg & CQ & \checkmark & CQ & \gg & \checkmark & \checkmark & \checkmark \\ \hline \textbf{P:} & \checkmark & LIC & LMH & CQ & \checkmark & CQ & SQ & \checkmark & \checkmark & \checkmark \\ \hline \end{array} \quad \gamma_4 = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \textbf{L:} & \checkmark & LIC & CQ & \checkmark & CQ & \checkmark & \checkmark & \checkmark \\ \hline \textbf{P:} & \checkmark & \gg & \gg & \checkmark & \gg & \checkmark & \checkmark & \checkmark \\ \hline \end{array}$$

Conversely, $\gamma_0 = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline \textbf{L:} & \checkmark & LIC & CQ & \checkmark & CQ & \checkmark & \checkmark & \checkmark \\ \hline \textbf{P:} & \checkmark & LIC & CQ & \checkmark & CQ & \checkmark & \checkmark & \checkmark \\ \hline \end{array}$ *is not a complete alignments since* σ_L *is not in the process behavior set of the* Declare *model. Indeed, the* co-existence *constraint is violated, because* Low Insurance Check *occurs in the log trace and* Low Medical History *does not. Moreover, two occurrences of* Create Questionnaire *are not followed by* Send Questionnaire*, as prescribed by the* response *constraint.*

In order to quantify the severity of a deviation, we introduce a cost function on the legal moves $\kappa \in \Sigma_A \to \mathbb{R}_0^+$. One can use a standard cost function with unit costs for moves in log or in model. However, the costs may also depend on the specific characteristics of the process, e.g., it may be more costly to skip an insurance check for high claims than for low claims. Therefore, a different cost function κ needs to be defined for individual processes. The cost of an alignment γ is defined as the sum of the costs of the individual moves in the alignment, $\mathcal{K}(\gamma) = \sum_{(s', s'') \in \gamma} \kappa(s', s'')$.

Given a log trace $\sigma_L \in \mathcal{L}$, our goal is to find a complete alignment of σ_L and a valid trace $\sigma_M \in \mathcal{P}_{\mathcal{D}}$ that minimizes the cost with respect to all $\sigma'_M \in \mathcal{P}_{\mathcal{D}}$. This complete alignment is referred to as an *optimal alignment*.

Definition 3 (Optimal Alignment). *Let $\sigma_L \in \mathcal{L}$ be a log trace and $\mathcal{D}$ a Declare model. Let $\Gamma_{(\sigma_L,\mathcal{D})}$ be the set of the complete alignments of σ_L and $\mathcal{D}$. A complete alignment $\gamma \in \Gamma_{(\sigma_L,\mathcal{D})}$ is an* optimal alignment *of $\sigma_L \in \mathcal{L}$ and $\mathcal{D}$ iff $\forall \gamma' \in \Gamma_{(\sigma_L,\mathcal{D})}$. $\mathcal{K}(\gamma') \geq \mathcal{K}(\gamma)$. The projection of γ on the second element (ignoring $\gg$) yields $\sigma_M \in \mathcal{P}_\mathcal{D}$, i.e., a valid trace of $\mathcal{D}$ that is the closest to σ_L.*

Example 1 (cont.). *In our example, we can suppose that deviations for activity* Send Questionnaire *are less severe than those referring to the other activities, since this activity is automatically performed by a system. Moreover, moves associated to ✓ can be weighted less than any other, since they refer to activities that are not in the model. Therefore, a reasonable cost function on legal moves can be defined as follows:*

$$\kappa(a', \gg) = \kappa(\gg, a') = \begin{cases} 1 & \text{if } a' = \checkmark \\ 2 & \text{if } a' = \text{Send Questionnaire} \\ 4 & \text{otherwise} \end{cases}$$

$$\kappa(a', a'') = \begin{cases} 0 & \text{if } a' = a'' \\ \infty & \text{if } a' \neq a'' \ \wedge \ a' \neq \gg \ \wedge \ a'' \neq \gg \end{cases}$$

Using this cost function, alignments γ_1, γ_2, γ_3 and γ_4 have the following costs: $\mathcal{K}(\gamma_1) = 6$, $\mathcal{K}(\gamma_2) = 12$, $\mathcal{K}(\gamma_3) = 6$ and $\mathcal{K}(\gamma_4) = 12$. Therefore, γ_1 and γ_3 are better complete alignments. According to the given cost function, γ_1 and γ_3 are, in fact, optimal alignments.

When focusing on the fitness dimension of conformance, we are not only interested in finding the optimal alignment and, hence, diagnosing where a log trace does not conform with a model. We also need to quantify the fitness level of traces and logs. Therefore, we introduce a *fitness function* $\mathcal{F} \in (\Sigma^* \times 2^{\Sigma^*}) \rightarrow [0, 1]$. $\mathcal{F}(\sigma_L, \mathcal{D}) = 1$ if σ_L can be replayed by the model from the beginning to the end with no non-conformance costs. Conversely, $\mathcal{F}(\sigma_L, \mathcal{D}) = 0$ denotes a very poor fitness. $\mathcal{K}(\cdot)$ cannot be used as fitness function directly, as we are interested in expressing the fitness level as a number between 0 and 1. The normalization between 0 and 1 can be done in several ways. In our approach, we divide the cost of the optimal alignment by the maximal possible alignment cost. Typically, the greatest possible cost of an alignment of a log trace $\sigma_L = \langle a_1^L, \ldots, a_n^L \rangle$ and a model trace $\sigma_M = \langle a_1^M, \ldots, a_m^M \rangle \in \mathcal{P}_\mathcal{D}$ is obtained for the *reference alignment* in which there are only moves in model and in log:

$$\gamma^{\text{ref}}_{(\sigma_L,\sigma_M)} = \begin{array}{|c|c|c|c|c|c|c|} \hline \textbf{L:} & a_1^L & \ldots & a_n^L & \gg & \gg & \gg \\ \hline \textbf{P:} & \gg & \gg & \gg & a_1^M & \ldots & a_m^M \\ \hline \end{array}.$$

Therefore, the *fitness level* of a log trace can be defined as follows:

Definition 4 (Fitness Level). *Let $\sigma_L \in \Sigma^*$ be a log trace and let $\mathcal{D}$ be a Declare model. Let $\gamma_O \in {\Sigma_A}^*$ be an optimal alignment of σ_L and $\mathcal{D}$ and $\sigma_M \in \mathcal{P}_\mathcal{D}$ the model trace in the optimal alignment. The fitness level of σ_L and $\mathcal{D}$ is defined as follows:*

$$\mathcal{F}(\sigma_L, \mathcal{D}) = 1 - \frac{\mathcal{K}(\gamma_O)}{\mathcal{K}(\gamma^{ref}_{(\sigma_L,\sigma_M)})}$$

Therefore, the fitness of σ_L and $\mathcal{D}$ is valued 1 if in the alignment there are only moves in both, i.e., there are no deviations. $\mathcal{F}(\sigma_L, \mathcal{D}) = 0$ if the optimal alignment only contains moves in log and in model. Note that this fitness function always returns a value between 0 and 1: if $\gamma_O \in {\Sigma_A}^*$ is an optimal alignment, any other alignment, including $\gamma^{\text{ref}}_{(\sigma_L,\sigma_M)}$, must have the same or a higher cost.

In the next section, we introduce an approach to create an optimal alignment with respect to a custom cost function κ. The approach is based on the A* algorithm that is intended to find the path with the lowest overall cost between two nodes in a direct graph with costs associated to nodes.

4 The A* Algorithm for Conformance Checking

Let us suppose to have a graph V with costs associated to arcs. The A* algorithm, initially proposed in [8], is a pathfinding search in V. It starts at a given *source* node $v_0 \in V$ and explores adjacent nodes until one node of a given *target set* $V_{Trg} \subset V$ of destination nodes is reached, with the intent of finding the path with the overall lowest cost. Every node $v \in V$ is associated to a cost, which is determined by an *evaluation* function $f(v) = g(v) + h(v)$, where

- $g : V \to \mathbb{R}_0^+$ is a function that returns the smallest path cost from v_0 to v;
- $h : V \to \mathbb{R}_0^+$ is an heuristic function that estimates the smallest path cost from v to any target node $v' \in V_{Trg}$.

Function h is said to be *admissible* if it never underestimates the smallest path cost to reach any target node: for each node $v \in V$ and for each target node $v' \in V_{Trg}$ reachable from v, $h(v) \leq g(v')$. Technical results in [8] shows that if h is admissible, A* finds a path that is guaranteed to have the overall lowest cost.

The A* algorithm keeps a priority queue of nodes to be visited: a higher priority is given to nodes with lower costs so as to traverse those with the lowest costs first. The algorithm works iteratively: at each step, the node v with lowest cost is taken from the priority queue. If v belongs to the target set, the algorithm ends returning v. Otherwise, v is expanded: every successor v' of v is added to the priority queue with cost $f(v')$.

4.1 Usage of A* to Find an Optimal Alignment

We use A* to find any of the optimal alignments of a log trace $\sigma_L \in \Sigma^*$ and a Declare model $\mathcal{D}$. In order to be able to apply A*, an opportune search space needs to be defined. Every node γ of the search space V is associated to a different alignment that is a prefix of some complete alignment of σ_L and $\mathcal{D}$. Since a different alignment is also associated to every node and vice versa, later on we use the alignment to refer to the associated node. The source node is the empty alignment $\gamma_0 = \langle\rangle$ and the set of target nodes includes every complete alignment of σ_L and $\mathcal{D}$. Since the successors of an alignment are obtained by adding a move to it, the search space is, in fact, a tree.

Let us denote the length of a trace σ with $\|\sigma\|$. Given a node/alignment $\gamma \in V$, the search-space successors of γ include all alignments $\gamma' \in V$ obtained from γ by concatenating exactly one move. Let us consider a custom cost function κ and denote

with $\kappa^{\min}$ the smallest value returned by κ greater than 0. Given an alignment $\gamma \in V$ of σ'_L and σ'_M, the cost of a path from the initial node to $\gamma \in V$ is defined as:

$$g(\gamma) = \kappa^{\min} \cdot \|\sigma'_L\| + \mathcal{K}(\gamma).$$

It is easy to check that, given two complete alignments γ'_C and γ''_C, $\mathcal{K}(\gamma'_C) < \mathcal{K}(\gamma'_C) \Leftrightarrow g(\gamma'_C) < g(\gamma''_C)$ and $\mathcal{K}(\gamma'_C) = \mathcal{K}(\gamma'_C) \Leftrightarrow g(\gamma'_C) = g(\gamma''_C)$. Therefore, an optimal solution returned by the A* algorithm coincides with an optimal alignment. We have added the term $\kappa^{\min} \cdot \|\sigma'_L\|$ (which does not affect the optimality) to define a more efficient and admissible heuristics. Given an alignment $\gamma \in V$ of σ'_L and σ'_M, we utilize the following heuristic:

$$h(\gamma) = \kappa^{\min} \cdot (\|\sigma_L\| - \|\sigma'_L\|)$$

For an alignment γ, the number of moves to add in order to reach a complete alignment cannot exceed the number of moves of σ_L that have not been included yet in the alignment, i.e., $\|\sigma_L\| - \|\sigma'_L\|$. Since the additional cost to traverse a single node is at least $\kappa^{\min}$, the cost to reach a target node is at least $h(\gamma)$, corresponding to the case in which the part of the log trace that still needs to be included in the alignment (i.e., $\sigma_L \setminus \sigma'_L$) fits in full.

4.2 Search Space Reduction

Declarative models allow for more flexibility and, therefore, for more behavior than procedural models. Hence, the search space in the A* algorithm may be extremely large. Nevertheless, many search-space nodes (i.e., partial alignments) are, in fact, equivalent, i.e., some partial alignments can be extended with the same moves:

Definition 5 (Alignment Equivalence). *Let $\mathcal{D} = (A, \Pi)$ be a Declare model and let $\mathcal{A}_\pi = (\Sigma, \Psi_\pi, \psi_{0_\pi}, \delta_\pi, F_\pi)$ be the constraint automaton for $\pi \in \Pi$. Let $\sigma_L \in \Sigma^*$ be a log trace. Let γ' and γ'' be alignments of σ'_L and σ'_M, and of σ''_L and σ''_M, where σ'_L and σ''_L are prefixes of σ_L and σ'_M and σ''_M are prefixes of model traces in $\mathcal{P}_\mathcal{D}$. Let $\psi'_\pi = \delta^*_\pi(\psi_{0_\pi}, \sigma'_M)$ and $\psi''_\pi = \delta^*_\pi(\psi_{0_\pi}, \sigma''_M)$ be the states reached by $\mathcal{A}_\pi$ when replaying σ'_M and σ''_M on it.*[4] *Alignments γ' and γ'' are equivalent with respect to $\mathcal{D}$, if $\sigma'_L = \sigma''_L$ and, for all $\overline{\pi} \in \Pi$, $\psi'_{\overline{\pi}} = \psi''_{\overline{\pi}}$. We denote this with $\gamma' \sim_\mathcal{D} \gamma''$.*

If two partial alignments γ' and γ'' are equivalent, the cost of the least expensive path to reach a target node (i.e., a complete alignment) from γ' is the same as from γ''. Indeed, since they are equivalent, they both can be extended with the same sequences of alignment moves. In order to get an optimal alignment, it is only necessary to visit one of them, specifically the one with lowest g cost. Therefore, it is possible to prune the sub-trees with roots in the nodes/alignments that do not have to be visited.

[4] Given a state-transition function δ and a symbol sequence $\sigma = \langle s_1, \ldots, s_n \rangle$, $\delta^*(\psi_0, \sigma)$ denotes the recursive application of the state-transition function over a symbol sequence σ starting from state ψ_0, i.e., $\delta^*(\psi_0, \sigma) = \psi_n$ where, for all $0 < i \leq n$, ψ_i is recursively defined as $\psi_i = \delta(\psi_{i-1}, s_i)$.

Theorem 1. *Let $\mathcal{D}$ be a Declare model. Let σ_L be a log trace to be aligned. Let $\Gamma_{(\sigma_L,\mathcal{D})}$ be the set of complete alignments of σ_L and $\mathcal{D}$. Let γ' and γ'' be two alignments such that $\gamma' \sim_{\mathcal{D}} \gamma''$ and $g(\gamma') > g(\gamma'')$. For all complete alignments $\gamma' \oplus \widehat{\gamma} \in \Gamma_{(\sigma_L,\mathcal{D})}$, there exists a complete alignment $\gamma'' \oplus \widetilde{\gamma} \in \Gamma_{(\sigma_L,\mathcal{D})}$ such that $g(\gamma' \oplus \widehat{\gamma}) \geq g(\gamma'' \oplus \widetilde{\gamma})$.*

Proof. Let σ'_L be the portion of σ_L aligned by γ' and γ''. Let $\overline{\gamma'} = \gamma' \oplus \widehat{\gamma} \in \Gamma_{(\sigma_L,\mathcal{D})}$ be one of the complete alignments with the lowest cost among the ones that can be obtained by extending γ'. Since $\gamma' \sim_{\mathcal{D}} \gamma''$, alignment γ'' can also be extended with $\widehat{\gamma}$, i.e., $\overline{\gamma''} = \gamma'' \oplus \widehat{\gamma} \in \Gamma_{(\sigma_L,\mathcal{D})}$. $g(\overline{\gamma'}) = k^{\min} \cdot \|\sigma_L\| + \mathcal{K}(\gamma' \oplus \widehat{\gamma})$ and $g(\overline{\gamma''}) = k^{\min} \cdot \|\sigma_L\| + \mathcal{K}(\gamma'' \oplus \widehat{\gamma})$. Suppose that $g(\overline{\gamma''}) > g(\overline{\gamma'})$ with $g(\gamma') > g(\gamma'')$. If this holds, $\mathcal{K}(\gamma'' \oplus \widehat{\gamma}) = \mathcal{K}(\gamma'') + \mathcal{K}(\widehat{\gamma}) > \mathcal{K}(\gamma' \oplus \widehat{\gamma}) = \mathcal{K}(\gamma'') + \mathcal{K}(\widehat{\gamma})$. Therefore, $\mathcal{K}(\gamma'') > \mathcal{K}(\gamma')$ and, hence, $g(\gamma'') > g(\gamma')$, which is a contradiction. Therefore, there is a complete alignment $\overline{\gamma''}$ obtained by extending with cost lower or equal to $g(\overline{\gamma'})$, i.e., the lower bound of the costs of all complete alignments obtained by extending γ'.

We maintain a set Γ_V of nodes $\overline{\gamma}$ that have already been visited and their costs $g(\overline{\gamma})$. When a new node-alignment γ' is encountered, we check whether it is a candidate node to be visited, i.e., whether its successors need to be added to the priority queue of nodes to be visited. Node γ' is a candidate if for every node $\gamma'' \in \Gamma_V$ equivalent to γ' $g(\gamma'') > g(\gamma')$. It is also a candidate if there is no equivalent node in Γ_V.

5 Provided Diagnostics

This section details some advanced diagnostics that we build on top of the optimal alignments that have been returned for all traces in the event log. First, we indicate why an optimal alignment includes a certain move in log/model: in fact, such a move was introduced to solve a violation of a constraint that occurred in the log trace. Second, we provide a helicopter view that allows one to determine which activities are mostly involved in deviations and which constraints are more often violated. On this concern, we provide metrics to measure the "degree of conformance" of single activities and constraints in a Declare model against the entire event log, in addition to simply evaluating the fitness level (Definition 4) of each log trace against the entire Declare model.

5.1 Why Do I Need This Move?

Let $\gamma = \langle (a_1^L, a_1^P), \ldots, (a_n^L, a_n^P) \rangle$ be an optimal alignment of σ_L and $\mathcal{D}$. Let $\mathcal{A}_\pi$ be the constraint automaton for $\pi \in \Pi$. For each move $(a_i^L, a_i^P) \in \gamma$ in log or in model of an alignment (i.e., s.t. either $a_i^L = \gg$ or $a_i^P = \gg$), we indicate which constraint(s) in the Declare model the move aims to solve. For this purpose, we build an execution trace σ_i obtained from $\langle a_1^P, \ldots, a_{i-1}^P, a_i^L, a_{i+1}^P, \ldots, a_n^P \rangle$ by removing all $\gg$. Then, for each constraint $\pi \in \Pi$, we check whether σ_i is accepted by $\mathcal{A}_\pi$. If it is not accepted, (a_i^L, a_i^P) has been introduced to solve a violation in π. Note that, a move in log or in model always solves at least one violation.

Example 1 (cont.). *Let us again consider the optimal alignment γ_1 (see Page 87). It contains two moves in log or in model: $(LIC, \gg)$ and $(\gg, SQ)$. For $(LIC, \gg)$, we build the execution*

trace $\sigma_1 = \langle \checkmark, LIC, CQ, \checkmark, CQ, \checkmark, \checkmark, \checkmark \rangle$. *This sequence is accepted by all the constraint automata in the* Declare *model, apart from the constraint automaton for the* co-existence *constraint (see Fig. 2a). Similarly, for* $(\gg, SQ)$, $\sigma_5 = \langle \checkmark, CQ, \checkmark, CQ, \checkmark, \checkmark, \checkmark \rangle$ *is accepted by all the constraint automata, apart from the constraint automaton for the* precedence *constraint (shown in Fig. 2b). Therefore,* $(LIC, \gg)$ *has been introduced to solve a violation in the* co-existence *constraint and* $(\gg, SQ)$ *has been introduced to solve a violation in the* precedence *constraint.*

5.2 Degree of Conformance

We denote with $MC_\gamma(\pi)$ the metric representing the number of moves in model and in log of a complete alignment γ that contribute to solve a violation of π. For $a \in \Sigma$, we denote with $MM_\gamma(a)$ the number of a moves in model, with $ML_\gamma(a)$ the number of a moves in log and with $MB_\gamma(a)$ the number of a moves in both model and log. $MC_\gamma(\pi)$, $MM_\gamma(a)$, $ML_\gamma(a)$ and $MB_\gamma(a)$ can be used to quantify the degree of conformance.

For reliability, we average over all optimal alignments. Let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ be the set of the optimal alignments of a log $\mathcal{L} = \{\sigma_1, \ldots, \sigma_n\}$ and $\mathcal{D}$. The *degree of conformance* of $a \in \Sigma$ with respect to Γ is defined as follows:

$$\mathsf{DConf}_\Gamma(a) = 1 - \frac{1}{n} \cdot \sum_{\gamma \in \Gamma} \frac{MM_\gamma(a) + ML_\gamma(a)}{MM_\gamma(a) + ML_\gamma(a) + MB_\gamma(a)}.$$

$\mathsf{DConf}_\Gamma(a) = 1$ if the moves that involve a are only moves in both (i.e., there are no deviations related to a). $\mathsf{DConf}_\Gamma(a)$ decreases with the fraction of moves in model or in log. $\mathsf{DConf}_\Gamma(a) = 0$ if all moves that involve a are only moves in log or moves in model.

Given a constraint $\pi \in \Pi$, the degree of conformance π with respect to Γ is defined as follows:

$$\mathsf{DConf}_\Gamma(\pi) = 1 - \frac{1}{n} \cdot \sum_{\gamma \in \Gamma} \frac{MC_\gamma(\pi)}{\|\gamma\|}.$$

$\mathsf{DConf}_\Gamma(\pi) = 1$ if π is never violated. $\mathsf{DConf}_\Gamma(\pi)$ decreases towards 0 as the fraction of moves in model and in log needed to solve violations of π increases.

6 Implementation and Experiments

To check the conformance of Declare models, we have implemented two plug-ins of ProM, a generic open-source framework for implementing process mining functionality [18]. The first plug-in is the *Declare Replayer* that takes as input a Declare model and an event log and, using the algorithm described in Section 4, finds an optimal alignment for each trace in the event log. Starting from the results of the *Declare Replayer*, a second plug-in, the *Declare Diagnoser* generates a map based on the diagnostics described in Section 5.

Section 6.1 reports some experiments to analyze the performance of our approach. Then, Section 6.2 presents our plug-ins and illustrates how diagnostics are graphically

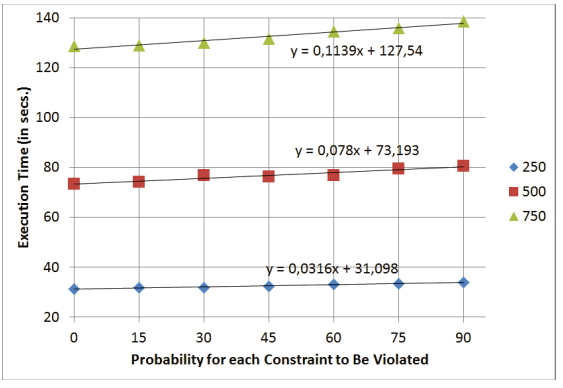


Fig. 3. Results of the experiments conducted on synthetic logs with different combinations of sizes and degrees of non-conformance

visualized in a map. Most of the experiments in Section 6.1 use synthetic logs. Nevertheless, we have also validated our approach on a real case study in the context of the CoSeLoG project[5] involving 10 Dutch municipalities.

6.1 Performance Experiments

To carry out our experiments, we have generated a set of synthetic logs by modeling the process described in Example 1 in CPN Tools (`http://cpntools.org`) and by simulating the model. We use logs with different degrees of non-conformance. A degree of non-conformance of 90% means that each constraint in the log is violated with a probability of 90%. Note that multiple constraints can be violated at the same time in a log trace. In our experiments, the logs have varying degrees of non-conformance: $0\%, 15\%, 30\%, 45\%, 60\%, 75\%$ and 90%. For all degrees of non-conformance, we use randomly generated logs including $250, 500$ and 750 instances to verify the scalability of the approach when varying the log size. The experiments have been conducted on a dual-core CPU at 2.40 GHZ.

Fig. 3 shows the execution times for the *Declare Replayer* plug-in (implementing the A* algorithm) for the different logs. For each combination of log size ($250, 500$ or 750) and degree of non-conformance (x-axis), the figure plots the average value of the execution time over 5 runs. The three lines show the trends for the three different log sizes. Fig. 3 illustrates the scalability of the algorithm.

Table 1 shows the effect of pruning the state space and of the heuristics. If we do not prune the search-space employing the technique described in Section 4.2, the *Declare Replayer* has to visit 41% of extra nodes and, consequently, the execution time increases by 33%. Table 1 also shows a dramatic reduction in time and nodes achieved by the heuristics described in Section 4.1. Without using the heuristics, the needed amount of memory increases from 300 MBs to 3.5 GBs.

[5] `http://www.win.tue.nl/coselog`

Table 1. Comparison of the execution time of the *Declare Replayer* when all optimizations are enabled with respect to the cases when they are selectively turned off. The results refer to a log with 250 instances and a degree of non-conformance of 15%.

Employed Technique	Visited Nodes	Tree Size	Execution Time
Optimized A*	21	181	30 seconds
Without pruning	30	252	40 seconds
Without heuristics	123419	1204699	ca. 7 hours

We have also performed various experiments using real-life event logs from the CoSeLoG project. For the validation reported here, we have used two logs of processes enacted by two different Dutch municipalities. Process instances in these event logs refer to permissions for building or renovating private houses. We have first discovered a Declare model using an event log of one municipality using the *Declare Miner* plug-in in ProM [14,13]. Then, using the *Declare Replayer*, we have checked the conformance of the mined model with respect to an event log of the second municipality, where every deviation is assigned the same cost/weight. Analysis showed commonalities and interesting differences. From a performance viewpoint the results were also encouraging: 3271 traces with 14338 events could be replayed in roughly 8 minutes, i.e., 137 milliseconds per trace. Section 6.2 provides more details about this experiment.

6.2 User Interface and Diagnostics

Fig. 4 illustrates the output produced by the *Declare Replayer*. The screenshot shows an analysis used in the context of a case study involving two municipalities. An event log of one municipality is compared with a Declare model learned from an event log of another municipality.

Each sequence of triangles in Fig. 4 refers to an alignment of a trace with respect to the model. Each triangle is a different alignment move; the color of the move depends on its type (see the legend on the right-hand side), i.e., move in log (yellow), move in model (purple) or move in both (green). Each sequence is also associated with a number that identifies the fitness level of the specific trace. A button *Detail* is also associated to each trace; it allows us to show the alignment details at the bottom (e.g., for trace 1649 in the screenshot). Each rectangle represents a different move and is annotated with the activity involved in the move. Also here, the color of the move depends on its type, i.e., move in log (yellow), move in model (purple) or move in both (green). In case of moves in log or in model, when moving with the mouse over the rectangle, the *Declare Replayer* shows which constraint violation it aims to solve, in line with the diagnostics described in Section 5.1. In the figure, for trace 1649, the 8th alignment move concerns a move in log for *Verzenden beschikking*. This move has been introduced to solve a violation in a precedence constraint modeling that if activity *Verzenden beschikking* occurs in the log, *Beslissing* must precede, being *Beslissing* not present beforehand. The *Declare Replayer* also provides the average fitness with respect to all log traces (0.8553808). This value indicates that the Declare model mined from the first event log is not fully conforming with the second log, i.e., the two municipalities execute the two processes in a slightly different manner.

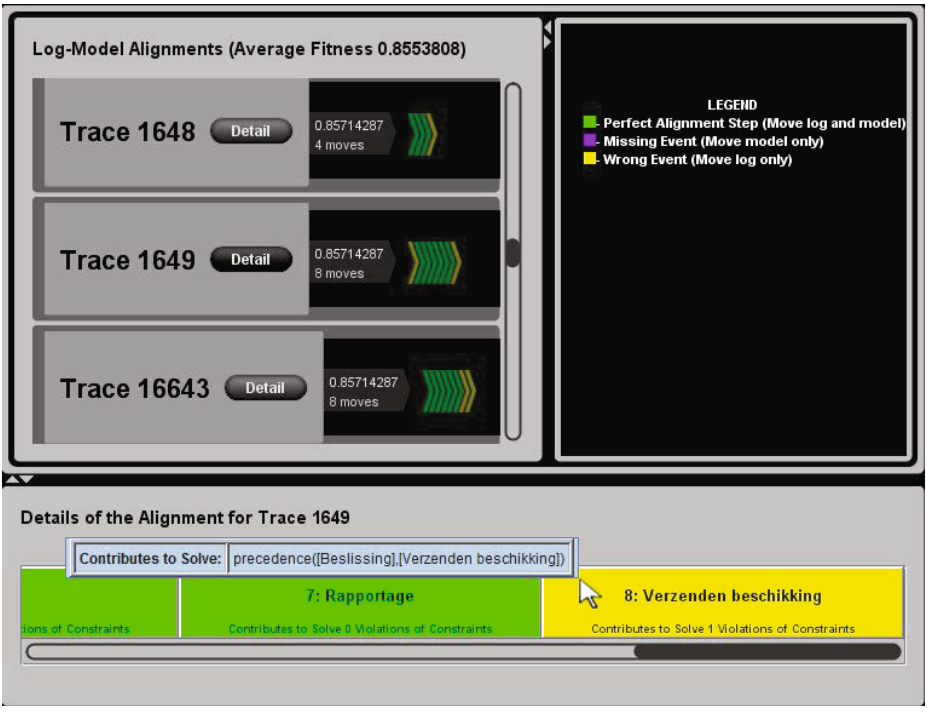


Fig. 4. A screenshot showing the output of the *Declare Replayer* plug-in. For clarifying, we provide the English translation of the Dutch activity names. *Administratie, Toetsing, Beslissing, Verzenden beschikking* and *Rapportage* can be translated with *Administration, Verification, Judgement, Sending Outcomes* and *Reporting*, respectively.

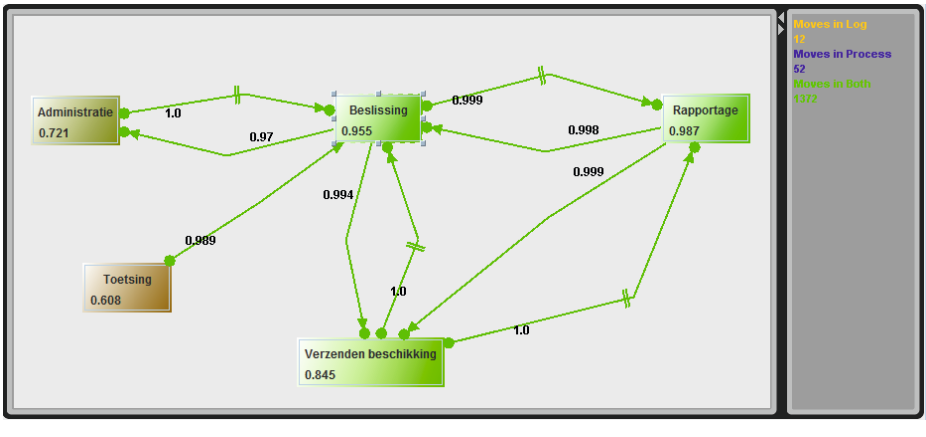


Fig. 5. A screenshot showing the output of the *Declare Diagnoser* plug-in

The results obtained through the *Declare Replayer* can be "projected" on the Declare model by the *Declare Diagnoser* plug-in. The *Declare Diagnoser* plug-in annotates activities and constraints of the Declare model with the degree of conformance. In this way, the process analyst can easily understand where the deviations occur most frequently. Fig. 5 depicts a screenshot of the output of this second plug-in when taking as input the alignments shown in Fig. 4. Activities and constraints are annotated with numbers showing the degree of conformance. To make the visualization more effective also colors are used. Green and red nodes and arcs indicate a degree 1 or 0 of conformance, respectively. Intermediate shades between green and red reflect values in-between these two extremes.

The coloring of activities and constraints in Fig. 5 shows that the level of conformance is reasonable (most parts are close to green). As shown, most of the detected deviations are related to activities *Toetsing* and *Administratie*, which have the lowest degree of conformance ($\mathsf{DConf}_\Gamma(Toetsing) = 0.608$ and $\mathsf{DConf}_\Gamma(Administratie) = 0.721$). The other activities have degree of conformance close to 1. By selecting an activity, a better insight is provided: in the figure, the selected activity *Beslissing* is involved 12 times in a move in log, 52 times in a move in model and 1372 times in a move in both. The degree of conformance of a constraint indicates whether the constraint is somewhere violated. For instance, $\mathsf{DConf}_\Gamma(precedence(Beslissing, Administratie)) = 0.97$ highlights that moves in log and/or in model have been included in some alignments to solve a violation in this constraint. $\mathsf{DConf}_\Gamma(not\ succession(Administratie, Beslissing)) = 1$ indicates that this constraint is never violated.

7 Conclusion

This paper presents a novel conformance checking approach tailored towards declarative models. The many conformance checking techniques defined for procedural models (e.g., Petri nets) are not directly applicable to declarative models. Moreover, these techniques tend to provide poor diagnostics, e.g., just reporting the fraction of fitting cases. We adapted alignment-based approaches to be able to deal with the large search spaces induced by the inherent flexibility of declarative models. Based on such alignments we provide novel diagnostics, at the trace level, showing why events need to be inserted/removed in a trace, and at the model level, coloring constraints and activities in the model based on their degree of conformance. As future work, we plan to extend our approach in order to incorporate in our analysis data and resource perspectives.

References

1. van der Aalst, W.M.P., de Beer, H.T., van Dongen, B.F.: Process Mining and Verification of Properties: An Approach Based on Temporal Logic. In: Meersman, R., Tari, Z. (eds.) CoopIS/DOA/ODBASE 2005. LNCS, vol. 3760, pp. 130–147. Springer, Heidelberg (2005)
2. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative Workflows: Balancing Between Flexibility and Support. Computer Science - R&D, 99–113 (2009)
3. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)

4. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
5. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking Using Cost-Based Fitness Analysis. In: IEEE International Enterprise Distributed Object Computing Conference, pp. 55–64. IEEE Computer Society (2011)
6. Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 326–341. Springer, Heidelberg (2008)
7. Bauer, A., Leucker, M., Schallhart, C.: Runtime Verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology (2011)
8. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. Journal of the ACM (JACM) 32, 505–536 (1985)
9. Governatori, G., Milosevic, Z., Sadiq, S.W.: Compliance Checking Between Business Processes and Business Contracts. In: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference, pp. 221–232. IEEE Computer Society (2006)
10. Greco, G., Guzzo, A., Pontieri, L., Sacca, D.: Discovering expressive process models by clustering log traces. IEEE Trans. on Knowl. and Data Eng. 18(8), 1010–1027 (2006)
11. Hulstijn, J., Gordijn, J.: Risk analysis for inter-organizational controls. In: Proceedings of the 12th International Conference on Enterprise Information Systems. SciTePress (2010)
12. de Leoni, M., van der Aalst, W.M.P., van Dongen, B.F.: Data- and Resource-Aware Conformance Checking of Business Processes. In: Abramowicz, W., Kriksciuniene, D., Sakalauskas, V. (eds.) BIS 2012. LNBIP, vol. 117, pp. 48–59. Springer, Heidelberg (2012)
13. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient Discovery of Understandable Declarative Process Models from Event Logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012)
14. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, pp. 192–199. IEEE (2011)
15. Munoz-Gama, J., Carmona, J.: Enhancing precision in process conformance: Stability, confidence and severity. In: IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011), pp. 184–191 (April 2011)
16. Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems 33(1), 64–95 (2008)
17. Vasarhelyi, M., Alles, M., Kogan, A.: Principles of Analytic Monitoring for Continuous Assurance. Journal of Emerging Technologies in Accounting 1(1), 1–21 (2004)
18. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, xESame, and proM 6. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 60–75. Springer, Heidelberg (2011)
19. Westergaard, M.: Better Algorithms for Analyzing and Enacting Declarative Workflow Languages Using LTL. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 83–98. Springer, Heidelberg (2011)
20. Westergaard, M., Maggi, F.M.: Declare: A tool suite for declarative workflow modeling and enactment. CEUR Workshop Proceedings, vol. 820. CEUR-WS.org (2011)