In Log and Model We Trust?
A Generalized Conformance Checking Framework

Andreas Rogge-Solti[1], Arik Senderovich[2], Matthias Weidlich[3],
Jan Mendling[1], and Avigdor Gal[2]

[1] Vienna University of Economics and Business, Austria
firstname.lastname@wu.ac.at
[2] Technion–Israel Institute of Technology, Haifa, Israel
{sariks@tx,avigal@ie}.technion.ac.il
[3] Humboldt University zu Berlin, Germany
matthias.weidlich@hu-berlin.de

Abstract. While models and event logs are readily available in modern organizations, their quality can seldom be trusted. Raw event recordings are often noisy, incomplete, and contain erroneous recordings. The quality of process models, both conceptual and data-driven, heavily depends on the inputs and parameters that shape these models, such as domain expertise of the modelers and the quality of execution data. The mentioned quality issues are specifically a challenge for conformance checking. Conformance checking is the process mining task that aims at coping with low model or log quality by comparing the model against the corresponding log, or vice versa. The prevalent assumption in the literature is that at least one of the two can be fully trusted. In this work, we propose a *generalized conformance checking* framework that caters for the common case, when one does neither fully trust the log nor the model. In our experiments we show that our proposed framework balances the trust in model and log as a generalization of state-of-the-art conformance checking techniques.

Keywords: process mining, conformance checking, model repair, log repair

1 Introduction

Business process management plays an important role in modern organizations that aim at improving the effectiveness and efficiency of their processes. To assist in reaching this goal, the research area of process mining offers multitude of techniques to analyze event logs that carry data from business processes. Such techniques can be classified into *process discovery* that sheds light into the behavior captured in event logs by searching for a model that best reflects the encountered behavior [3], *conformance checking* that highlights differences between a given process model and an event log [19,2], *model repair* that attempts to update a process model by adding behavior that is between model and log [9,6], and *anomaly detection* that identifies anomalies in event logs with respect to expected behavior to locate sources of errors in business processes [17].

Process mining investigates the interplay among reality (*system*), its reported observations (*event log*), and a corresponding *process model* [5]. While reality is typically

unknown, we are left with the need to reconcile the event log and the process model, where evidence of a certain behavior may only be present in one but not in the other.

Current conformance checking techniques are not capable of defining levels of trust for model and log to cater for uncertainty. Therefore, in this paper we consider the problem of optimally reconciling an event log with a process model, given an input event log and a model (if such exist) and our degree of trust in each. We outline that various process mining tasks can actually be regarded as special cases of this generic problem formulation. Specifically, we define the problem of *generalized conformance checking* (GenCon). It goes beyond locating misalignments between a process model and an event log by providing explanations of misalignments and categorizing them as one of a) anomalies in an event log, b) modeling errors, and c) unresolvable inconsistencies. This generalized conformance checking problem can be seen as the unification of conformance checking, model repair, and anomaly detection.

The contribution of this paper is threefold. First, we introduce a formalization of generalized conformance checking, i.e., the GenCon problem. It is cast as an optimization problem that incorporates distance measures for logs, for models, and for pairs of a log and a model. Second, to demonstrate our approach, we consider a specific instantiation of this problem, using process trees as a formalism to capture models along with distance measures based on (log or tree) edit operations and alignments between a log and a model. For this problem instance, we propose a divide-and-conquer approach that exploits heuristic search in the model space to transform a given model-log pair into their improved counterparts. Third, we provide a thorough evaluation of the approach based on three real-world datasets. Our experiments show that the GenCon problem setting has an empirical grounding, and outline its potential to complement existing process mining techniques.

The remainder of this paper is structured as follows. Section 2 motivates and describes the general problem setting, formalizes the GenCon problem, and relates it to common process mining tasks. In Section 3, we introduce the required notation for a particular instantiation of this problem, i.e., event logs, process trees, and related distance measures. Section 4 then presents a divide-and-conquer approach to address this particular problem instance. Section 5 empirically evaluates our approach in comparison with alternative techniques. Section 6 concludes the paper.

2 The Setting of Generalized Conformance Checking

Buijs et. al [5] define the truly executed business process as the *system* and analyze logs and models with respect to it. An event log may be noisy, with events that occurred in the system yet not recorded (missing events), and also events that did not occur in real-life (log errors) [18]. Discovering models from noisy event logs may result in under-fitting models with respect to the system, or alternatively models that allow impossible behavior. In symmetry, an external process model may allow excess behavior that cannot happen in the system, or it can under-represent behavior that actually occur [5].

Such quality issues gave rise to two process mining tasks, namely *model repair* and *log repair*. Model repair techniques consider the event log as the best evidence a system may produce, and adjust the model accordingly [9]. In contrast, log repair methods trust

a given model from which the log deviates and repair the log [18]. Orthogonal to tasks that aim at changing either the log or the model, conformance checking compares models and logs to detect deviations between the two [1]. For example, the alignment technique proposed in [2] receives an event log and a process model, as well as costs of deviations in log and in model. Based on the costs, the alignment finds an optimal mapping between traces and possible paths through the model that minimizes the total deviation costs. An alignment shows where the model and the log perform unsynchronized moves. For each such move either the model or the log could be blamed and considered for repair.

Generalized conformance checking unites the three tasks of model repair, log repair, and conformance checking under a common roof. Specifically, it aims at altering both model and log such that control-flow discrepancies, which are measured via standard conformance checking, are reduced. The underlying assumption is that the log and the model are not 'wrong' for the same fragments of the system, and their joint repair is beneficial. Under this assumption, generalized conformance checking results in a pair of event log and model that are both better representatives of the originating system, and are tightly aligned together. The assumption on how beneficial it is to correct the model based on the log, or vice versa, stems from trust levels, which are associated with the initial model and log.

2.1 Formalization of the GenCon Problem

We now define generalized conformance checking by formalizing its primitives, and formulating an optimization problem that result in a joint of model and log. Let $\mathcal{L}$ be the universe of event logs, and let $\mathcal{M}$ be the universe of process models. We are given an event log $L \in \mathcal{L}$ and a process model $M \in \mathcal{M}$, as well as their corresponding trust levels $\tau_L, \tau_M \in [0, 1]$ (with 0 corresponding to zero trust, and 1 being full trust). A trust level is predefined for both model and log. Model trust can be related to the experience of the modeler, or the validity of a discovery algorithm that was used to extract the model. Log trust corresponds to the veracity of the data. For example, the data that we use for our experiments in Section 5 comes from real-time locating system (RTLS) sensors. These sensors typically come with a known error range, and measurement quality issues. The latter can be used to construct a prior of trust.

We distinguish the following distance functions:

- the function $\delta_{\mathcal{L}^2} : \mathcal{L} \times \mathcal{L} \to [0, 1]$ measures the distance between two event logs.
- the function $\delta_{\mathcal{M}^2} : \mathcal{M} \times \mathcal{M} \to [0, 1]$ measures the distance between two models.
- the function $\delta_{\mathcal{LM}} : \mathcal{L} \times \mathcal{M} \to [0, 1]$ measures the distance (alignment) between an event log and a process model;

We are now ready to formulate the GenCon optimization problem.

Problem 1 (Generalized Conformance Checking (GenCon) Problem) *Given the input tuple $\langle L, M, \delta_{\mathcal{L}^2}, \delta_{\mathcal{M}^2}, \delta_{\mathcal{LM}}, \tau_L, \tau_M \rangle$ of initial log, initial model, the distance functions and the two trust levels, find a pair $(L^*, M^*) \in \mathcal{L} \times \mathcal{M}$ such that*

$$\begin{gathered}(L^*, M^*) = \underset{(L', M') \in \mathcal{L} \times \mathcal{M}}{\arg\min} \langle \delta_{\mathcal{L}^2}(L, L'), \delta_{\mathcal{M}^2}(M, M'), \delta_{\mathcal{LM}}(L', M') \rangle. \\ (\textbf{subject to} : \delta_{\mathcal{L}^2}(L, L') \leq 1 - \tau_L, \textbf{ and } \delta_{\mathcal{M}^2}(M, M') \leq 1 - \tau_M)\end{gathered} \tag{1}$$

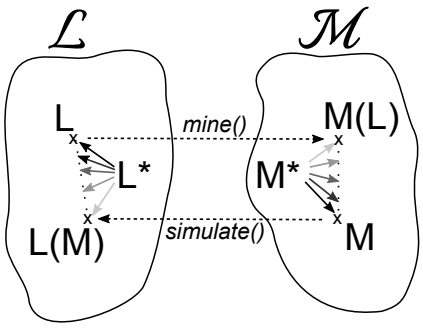


Fig. 1: Conceptual sketch of the problem setting.

We aim at a pair (L^*, M^*) that is in a trust-based proximity to the original pair of log and model, as well as aligned with each other. Note that the abstract framing of the problem, rather than binding the concepts to specific distance notions and specific process models, creates a framework that is general enough to cover several process mining problems. To operationalize generalized conformance checking, we require the definition of the functions $\delta_{\mathcal{L}^2}$, $\delta_{\mathcal{M}^2}$, and $\delta_{\mathcal{L}\mathcal{M}}$, based on suitable data structures. We later provide one such example, introducing distance measures over logs and models.

Figure 1 illustrates the setting of generalized conformance checking. On the left, the universe of logs $\mathcal{L}$; on the right, the universe of models $\mathcal{M}$ is scetched. Given are the input log L and input model M. We can mine a model from L, to obtain a fitting $M(L)$; and simulate M to get a fitting log $L(M)$. Conceptually, the optimal log L^* can move closer to the image of the model, and also the optimal model M^* can be closer to the mined model. This depends on the respective trust levels, indicated by the grayscale of the arrows (darker for more trust, lighter for less). It is worth to highlight that the arrows labeled *mine()* and *simulate()* do not specify functions, as there can be countable infinite sets of simulated logs or mined models.

2.2 Related Work

Table 1 maps process mining tasks and respective techniques to the GenCon problem. In a nutshell, previous work on process discovery and conformance checking can be viewed as special cases of our approach. Regarding conformance checking, setting $\tau_L = 1$, and $0 < \tau_M < 1$, the GenCon problem corresponds to *model repair*. For $0 < \tau_L < 1$, and $\tau_M = 1$, we are facing *log repair*. When both trust levels are 1, GenCon corresponds to standard *conformance checking* (e.g., model-log alignment). Hence, GenCon extends existing approaches for process discovery and conformance checking in that it not only finds the deviations between a log and a model, but it also identifies if deviations between the two stem from the log, or from the model.

Process discovery is yet another special case of our approach, where the model trust is set to be zero. Recently proposed process discovery algorithms optimally balance model quality measures such as fitness, precision, and generalization [1,5]. Specifically, in [4] minimal values of those quality measures are used as inputs for the evolutionary

Process mining task	Log Trust	Model Trust
Classical Process Discovery finds a model that best fits to the entire event log, e.g., the alpha algorithm [3].	$\pi_L = 1$	$\pi_M = 0$
Heuristic Process Discovery algorithms apply preprocessing to the event log by discarding infrequent patterns [10,23].	$0 < \pi_L < 1$	$\pi_M = 0$
Model Repair fixes deficient models due to e.g., a change in the system that is reflected in the log. For example [9].	$\pi_L = 1$	$0 < \pi_M < 1$
Conformance Checking. This task tries to find misalignments between event log and model. Example works include [19,2,20].	$\pi_L = 1$	$\pi_M = 1$
Log Repair. Given a trusted model and a noisy log, we modify the log until it conforms to the model [18,17,21].	$0 < \pi_L < 1$	$\pi_M = 1$
"Happy Path" Simulation is complementary to heuristic process discovery. It is a theoretical use case where we do not trust infrequent parts of the model [15].	$\pi_L = 0$	$0 < \pi_M < 1$
Process Simulation is complementary to process discovery, where we are given an untrustworthy empty log and a fully trustworthy model.	$\pi_L = 0$	$\pi_M = 1$
Garbage In, Garbage Out. When both the model and the log are untrustworthy, the best log and model tuple that fits them is any pair of model and log that fits each other, including an empty log and an empty model.	$\pi_L = 0$	$\pi_M = 0$
Generalized Conformance Checking is the focus of this paper. Instead of only detecting the misalignments, as in conformance checking, we also provide, where the model would best be adopted, and where the log would best be adopted for a better overall fit.	$0 < \pi_L$	$0 < \pi_M$

Table 1: Some process mining tasks cast as problem instances.

tree miner (ETM), which finds a model that balances all quality measures. Similarly to our approach, a normative model can be assumed as an input to the ETM, and the resulting model is guaranteed to be in proximity to the normative model. However, since the ETM is an evolutionary approach, it has no termination guarantees [24], and real-life log discovery takes a long time to terminate, if at all [13]. Other discovery algorithms are based on log filtering to balance the quality measures. These algorithms neither return an aligned log, nor do they provide guarantees on the quality of the resulting model [13,10]. Therefore, generalized conformance checking extends process mining in that it does not impose $\tau_M = 0$, and returns both a process model and a repaired log.

3 Model

To give the necessary background for a specific instantiation of the GenCon problem, this section recalls notions of event logs, process trees, and related distance measures.

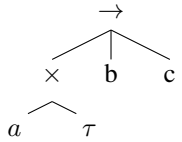
Event Logs. We adopt a common model of event logs that is grounded in sequences of activity labels that denote the activity executions as part of a single process instance (aka case). Let $\mathcal{A}$ be a universe of *activity labels* (*activities* for short). A *trace* $\sigma = \langle a_1, \ldots, a_n \rangle \in \mathcal{A}^*$ is a finite sequence of activities with a cardinality $|\sigma| = n$. The universe of traces is denoted by $\mathcal{T}$. An *event log* $L : \mathcal{T} \to \mathbb{N}$ is a multi-set of traces. A flattened representation $\bar{L}$ of a log L associates each trace with an identity such that $\bar{L} \in \mathcal{T}^*$ is a sequence of traces, and each trace $\sigma \in L$ is contained $L(\sigma)$ times in $\bar{L}$. The order of traces in $\bar{L}$ is arbitrary and $|\bar{L}|$ represents $\bar{L}$'s cardinality.

For example, an event log $L = \{\langle a, b, c \rangle^3, \langle b, c \rangle^1\}$ of three traces $\langle a, b, c \rangle$ and one trace $\langle b, c \rangle$ is flattened to $\bar{L} = \langle \langle a, b, c \rangle, \langle a, b, c \rangle, \langle a, b, c \rangle, \langle b, c \rangle \rangle$ with four distinct traces.

Process Trees. To represent process models, we adopt the notion of a process tree [12]. A process tree is a rooted tree, in which the leaf nodes are activities and all non-leaf nodes are control-flow operators. Common control-flow operators include sequences of activities ($\rightarrow$), exclusives choice ($\times$), concurrent execution ($\wedge$), and structured loops ($\circlearrowleft$). Process trees are defined recursively, as follows. Let $\Psi = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ be a set of *operators* and $\tau \notin \mathcal{A}$ be the *silent activity*. Then, $a \in \mathcal{A} \cup \{\tau\}$ is a *process tree*; and $\psi(T_1, \ldots, T_n)$, $n > 0$, with $T_1, \ldots, T_n$ being process trees and $\psi \in \Psi$ being an operator is a *process tree* ($n > 1$ if $\psi = \circlearrowleft$). The universe of process trees is denoted by $\mathcal{M}_T$.

The semantics of a process tree T is defined by a set of traces, which is also constructed recursively: A function $\iota : \mathcal{M}_T \to \wp(\mathcal{T})$ assigns a set of traces to a process tree. Trivially, $\iota(a) = \{\langle a \rangle\}$ for $a \in \mathcal{A}$ and $\iota(\tau) = \{\langle\rangle\}$. The interpretation of an operator $\psi \in \Psi$ is grounded in a specific language join function $\psi_l : \wp(\mathcal{T}) \times \ldots \times \wp(\mathcal{T}) \to \wp(\mathcal{T})$. Then, the semantics of a process tree $\psi(T_1, \ldots, T_n)$ is defined as $\iota(\psi(T_1, \ldots, T_n)) = \psi_l(\iota(T_1), \ldots, \iota(T_n))$. For instance, the trace set of the exclusive choice operator $\times_l(L_1, \ldots, L_n)$ is given by the union of the trace sets of its children $\bigcup_{1 \leq i \leq n} L_i$, i.e., See [12] for the formal execution semantics of all operators in Ψ.

A fitting tree model to the example log L above would be T:



Process tree T describes a sequence of a choice between a and a *silent activity* τ, followed by activity b, and then c.

Distance of Logs. We quantify the distance of event logs based on the weighted, normalized string edit distance of their traces. For two traces $\sigma, \sigma' \in \mathcal{A}^*$, we first define their normalized distance as $\delta_t(\sigma, \sigma') = \frac{d(\sigma, \sigma')}{\max(|\sigma|, |\sigma'|)}$ with $d(\sigma, \sigma')$ as the string edit distance, i.e., the minimal number of atomic activity operations (insert, delete, update) needed to transform one trace into another. Given two event logs in their flattened representation $\bar{L}, \bar{L}'$, let $\bar{L}^\epsilon = \bar{L} \cup \{\epsilon\}$ and $\bar{L}'^\epsilon = \bar{L}' \cup \{\epsilon\}$ with ϵ being the empty trace. Then, we define a mapping $\mu \subseteq \bar{L}^\epsilon \times \bar{L}'^\epsilon$, requiring that it is left-total, right-total, injective, and surjective when ignoring empty traces, i.e., for all $x \in \bar{L}$ there exists $y \in \bar{L}'^\epsilon$ such that $(x, y) \in \mu$; for all $y \in \bar{L}'$ there exists $x \in \bar{L}^\epsilon$ such that $(x, y) \in \mu$; for all $(x, x'), (y, y') \in (\mu \cap (\bar{L} \times \bar{L}))$ it holds that $x = y \Leftrightarrow x' = y'$.

The cost of such a mapping is defined as $\delta_m(\mu) = \sum_{(\sigma,\sigma')\in\mu} \delta_t(\sigma,\sigma')$. Then, the distance between two event logs L, L is defined based on the optimal mapping between their flattened representations $\bar{L}, \bar{L}'$ as follows:

$$\delta_{\mathcal{L}^2}(L, L') = \min_{\mu\subseteq \bar{L}^\epsilon \times \bar{L}'^\epsilon} \delta_m(\mu).$$

Distance of Models. To quantify the distance of process models, we exploit the fact that models are given as process trees and employ the tree edit distance [16]. Latter is, given two process trees T, T', the minimum cost sequence of node edit operations that transforms T into T'. Node edit operations are node deletion (connecting its children to its a parent maintaining the order); node insertion (between an existing node and a consecutive subsequence of its children); and node relabeling. Each of these node edit operations is assigned a cost.

When applying node edit operations to process trees, insertion and relabeling needs to respect the syntax of the model: inserted/relabeled nodes need to be activities if the node is a leaf; and control-flow operators otherwise. Further, we observe that the impact of node relabeling on the semantics of a process tree depends on the position of the relabeled node in the tree—intuitively, the higher the node is in the tree, the larger the effect on the language of the process tree would be. We define a relabeling cost for a node of a process tree as the number of leaf nodes of the subtree that is rooted in this node. Therefore, for a trivial process tree a with $a \in \mathcal{A} \cup \{\tau\}$, the node edit cost is $c(a) = 1$; for a process tree $\psi(T_1, \ldots, T_n)$, $n > 0$, the cost is $c(\psi(T_1, \ldots, T_n)) = \sum_{1\leq i\leq n} c(T_i)$. For node insertion and deletion, we employ unit costs.

Using this cost model, we define a normalized distance for process trees. As a normalization factor, we take the sum of the tree sizes: the size $|a|$ of a trivial process tree is defined as $|a| = 1$; for a process tree $\psi(T_1, \ldots, T_n)$, the size is defined as $|\psi(T_1, \ldots, T_n)| = 1 + \sum_{1\leq i\leq n} |T_i|$. Given two process trees T, T', we define the normalized distance as:

$$\delta_{\mathcal{M}^2}(T, T') = \frac{\delta_d(T, T')}{|T| + |T'|}, \text{ where } \delta_d(T, T') \text{ is the tree edit distance of } T \text{ and } T'.$$

Distance of Log and Model. As a distance measure between a log and a process model we consider different dimensions of the relation of logs and process models [5]: fitness (can a model show the behavior of a log?), precision (does a model allow for precisely the behavior of a log?), and generalization (does a model generalise over the behavior of a log?). We employ measures that are grounded in the notion of an *alignment* [2]. It is defined based on steps $(x, y) \in \mathcal{A}^\perp \times \mathcal{A}^\perp$, where $\mathcal{A}^\perp = \mathcal{A} \cup \{\perp\}$ is constructed from the universe of activities and a symbol $\perp \notin \mathcal{A}$. A step (x, y) is legal if $x \in \mathcal{A}$ or $y \in \mathcal{A}$ and is interpreted such that an alignment is said to 'move in both' traces ($(x, y) \in \mathcal{A} \times \mathcal{A}$), 'move in first' ($y = \perp$), or 'move in second' ($x = \perp$). Given two traces σ, σ', an alignment is a sequence of legal steps, such that the projection on the first step component (modulo $\perp$) yields σ and the projection on the second step component (modulo $\perp$) yields σ'.

Each step is assigned a cost and a common cost model assigns unit cost if either $x = \perp$ or $y = \perp$; zero cost if $x = y$; and infinite cost if $x \neq y$. An alignment is a

sequence of steps and the alignment cost is the sum of the costs of its steps. An alignment between two traces σ, σ' is optimal if it has the smallest possible cost.

Fitness is quantified by finding for each log trace $\sigma \in \mathrm{dom}(L)$, a trace $\sigma' \in \iota(T)$ of model T for which the optimal alignment has minimal cost regarding all traces in $\iota(T)$. We denote this alignment cost as $\delta_a(\sigma, T)$. Costs per log trace are then aggregated, weighted by the trace frequency in the log, and normalized by the maximal possible cost:

$$\delta_{fit}(L,T) = \sum_{\sigma \in \mathrm{dom}(L)} L(\sigma) \, \frac{\delta_a(\sigma,T)}{|\sigma| + \min\limits_{\sigma' \in \iota(T)} |\sigma'|}.$$

To quantify precision, we apply the measure proposed by Buijs et. al [5]. It is grounded in the transition system underlying the model T. Given an alignment of a log L and model T, let $state(L,T)$ be all states of this transition system that are visited when replaying all alignment steps of all traces $\sigma \in \mathrm{dom}(L)$ (ignoring steps with $\perp$ for the component of the model trace). Further, let $out(s)$ be the number of outgoing transitions of state $s \in state(L,T)$ and $taken(s,L)$ the number of these transitions that are taken during the replay of the log traces. Then, precision quantifies the additional allowed behavior of model T as

$$\delta_{pre}(L,T) = \frac{\sum\limits_{s \in state(L,T)} (out(s) - taken(s,L))}{\sum\limits_{s \in state(L,T)} out(s)}.$$

The replay of steps of an alignment is also used to quantify generalisation, directly on the model structure. Such a measure can be based on how often a node of a process tree is visited during replay [5]. Let $N(T)$ be the set of all nodes of a process tree T. Then, given L, let $visit(n,L)$ be the number of visits of node $n \in N(T)$ during replay of all alignment steps of all traces $\sigma \in \mathrm{dom}(L)$. Generalisation is defined as:

$$\delta_{gen}(L,T) = \frac{1}{|T|} \sum_{n \in N(T)} \left(\sqrt{visit(n,L)} \right)^{-1}.$$

Taking into account the above dimensions, we define the distance between a log L and a model T as follows:

$$\delta_{\mathcal{LM}}(L,T) = \frac{1}{3} \left(\delta_{fit}(L,T) + \delta_{pre}(L,T) + \delta_{gen}(L,T) \right).$$

4 A Divide-and-Conquer Approach for the GenCon Problem

In this section, we propose a two-step divide-and-conquer approach for addressing Problem 1, when the problem is instantiated for the model introduced in the previous section. The main idea of this approach is to avoid the inherent complexity of the problem induced by the freedom to change the model or the log by sequentialization: first identifying changes in the model, before turning to changes applied to the log.

Our solution is outlined in Figure 2. The first step consists of two parts, denoted by 1a) and 1b). In 1a), we lift the problem into the model space (i.e., process trees) by representing event logs as their discovered counterparts. Then, in 1b), we approximate

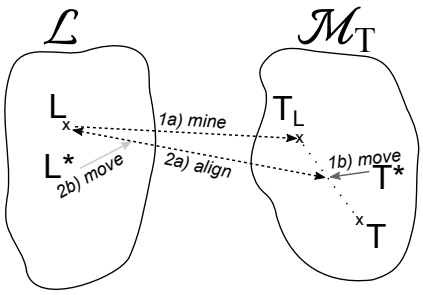


Fig. 2: Conceptual sketch of the heuristic proposed to solve the problem.

T^* by applying a greedy heuristic search. In the second step, we use the approximation for T^* to transform L into the approximated L^* via an alignment-based technique. Specifically, we align the input log with the originating model, and make moves in L until an approximation of L^* is reached.

4.1 A Quest for T^*

Log Lifting. The formulation of Problem 1 involves a search for (L^*, T^*) in two spaces: space of logs and space of models. To ease the search for (L^*, T^*) we lift the log L to its model representation by applying process discovery. This enables us to search for T^* in the model space without considering the log space. This step is agnostic to the specific discovery technique used for obtaining T_L. Aiming at efficient discovery of fitting process trees, in this work, we rely on the Inductive Miner [12].

Further, we denote the resulting process tree as T_L, and approximate T^* by an interpolation between T and T_L (step 1b) in Figure 2). To this end, we apply a greedy heuristic that is based on the behavioral similarity of intermediate models, T', and T'_L.

Heuristic Search. We quantify the tree-edit distance between T and T_L, which results in a cost-minimal set of tree-edit operations $\mathcal{E}$ with a a corresponding total tree-edit cost $C_{\mathcal{E}}$. Operating all edit operations in $\mathcal{E}$ to T transforms it into T_L. However, we do not always reach T_L, since we have a (possibly) non-zero trust in the originating model, and a (possibly) non-zero mistrust in the input log, L. Thus, the number of allowed edit operations depends on τ_L and τ_M.

We define the proportion of the total mistrust in both M and L, as

$$\gamma_{mistrust} = \min(1, 1 - \tau_M + 1 - \tau_L) = \min(1, 2 - \tau_M - \tau_L).$$

Consequently, we allow the algorithm to repair a total cost proportion of the model or the log that is at most $\gamma_{mistrust}$ of the entire cost $C_{\mathcal{E}}$.

However, since in this step, we only move from T towards T_L, we quantify the number of allowed repair operations on T. Denote the proportion of model repair operations out of the allowed $\gamma_{mistrust}$ by γ_{model}. The latter is defined as:

$$\gamma_{model} = \frac{1 - \tau_M}{2 - \tau_M - \tau_L} \gamma_{mistrust}.$$

Hence, the quest for T^* problem boils down to finding a set of edit operations $E \in 2^{\mathcal{E}}$ such that the total tree-edit cost of E is at most $\gamma_{model} C_{\mathcal{E}}$.

Our heuristic search contains the following steps. Initially, we consider $E = \emptyset$. We search for a tree-edit operation on T, $e \in \mathcal{E} \setminus E$ that maximizes the behavioral similarity of the resulting model T_E and T_L. The set must not violate the cost bound of $\gamma_{model} C_{\mathcal{E}}$. In practice, we consider several behavioral similarity functions, including behavioural profile-based similarity [11], and behavioural footprints [3].

If such e exists, we update E to be $E \cup \{e\}$ and consider further edit operations out of the remaining set, namely $\mathcal{E} \setminus E$. Selection of edit operations stops if there are no further candidates to enter E. In other words, one of the following condition holds: (1) the algorithm reached the mined model T_L, or (2) applying any repair operation in the set $\mathcal{E} \setminus E$ is beyond the allowed repair bound γ_{model}. Operating all tree-edit operations in E transforms T into a process tree T_E, which is our approximation of T^*.

4.2 Approximating L^*

Once an approximation for T^* is obtained, we turn to search for L^*. In this step, we assume that T^* is correct, and adjust the log to better fit T^*. Here, we rely on the alignment between log and model [2] that yields the minimal edit operations per trace. Specifically, the edit operations can be: (1) move in model (i.e., an activity in the model is missing an event counterpart in the trace), (2) move in log (i.e., the event has no corresponding part in the model), and (3) synchronous move (i.e., the event has a matching representation in the current state of the model).

Based on the assumption that model T^* is optimal, we assume that all remaining misalignments stem from the event log L. Analogously to the greedy heuristic for finding T^*, we greedily repair L towards a log that perfectly fits T^*. As before, we do not allow repairing the log beyond the trust level τ_L. To this end, we sort the misaligned candidates according to their frequency in the alignment result, and apply repair operations sequentially, until the distance of the current log L' to L is smaller than (1-τ_L). The result of this step is L^*. Finally, we return the approximated optimal pair (L^*, T^*).

The run-time complexity of the entire method is dominated by optimal alignments, which are in worst-case exponential in the length of the alignment [14]. The inductive miner that is used for discovery is polynomial in the number of activities [13], and tree-edit distance using a greedy heuristic has worst-case complexity of $O(|T|^2)$ with $|T|$ being the size of the larger tree [16]. Last, the heuristic search for T^* is also quadratic in the size of the tree, because the behavioral footprint needs to be read from each candidate tree (linear) when applying one operation, and there are maximum $|T|$ operations. The heuristic search for L^* is dominated by the alignment, as the remaining steps are sorting ($O(n \log n)$ where n is the number of activities in the model) and greedily picking the misalignment with the highest occurrence count (linear in the number of misalignments).[4]

Note that the problem is symmetric by nature, and we could also approach it the other way around. In that case, we would first align the input log L to the input model T,

[4] The approach has been implemented in the `GeneralizedConformance` plugin in ProM. A screencast demonstrating the usage of the plugin is provided here: `http://andreas.solti.de/generalized-conformance-checking/`

and move the log toward a better fit with respect to τ_L. This results in L^* first. Based on this, we would mine a representation of L^* in the model space, i.e., discover T_{L^*}. And last, we would move the input model toward the discovered model with respect to trust level τ_M, to derive T^*.

5 Evaluation

We evaluated the benefits of generalized conformance checking in general, and the proposed approach to address a specific instantiation of the GenCon problem, by answering the following questions:

Effect of Trust (EoT): How do the trust levels τ_L and τ_M affect the quality of the resulting log and model pair? We use this step to explore the range of possible solutions.

Model Repair Quality (MRQ): Can the proposed solution compete with state-of-the-art (specialized) model repair algorithms?

Log Repair Quality (LRQ): How does our technique affect the quality of models when we repair the log?

Below, we first outline our experimental setup and present the event logs and models used as inputs to the experiments. Then, we provide an overview of the main results.

5.1 Experiment Setup

To evaluate our approach we consider event logs and process trees as detailed in Section 3. For process tree discovery, we use the Inductive Miner [12]. For model search (see Section 4.1), we consider the following greedy heuristics:

- Simple heuristic – prioritizes delete operations, then applies add operations, and lastly renames nodes.
- Random heuristic – applies edit operations in random order.
- Footprints heuristic – uses a similarity measure based on behavioral footprints (as in [3]) to select the next edit operation.
- Behavioral profiles heuristic – uses a similarity measure based on behavioral profiles (as in [11]) to select the next edit operation.

Due to space limitation, we present the results for the heuristic that yielded the best results, namely the footprints heuristic. In the remainder of the experimental setup, we specify the controlled variables and the responses that we measure to answer the three aforementioned questions.

Controlled Variables. The experiments vary on the trust levels (τ_L, τ_M). First, for assessing the effect of trust (EoT) we vary for each event log and a corresponding model, both trust levels. For the model repair quality (MRQ) experiment, we fix the input trust in the log to $\tau_L = 1$, and vary the trust in the model τ_M between 0 and 1. Similarly, to evaluate the log repair quality (LRQ) we fix $\tau_M = 1$ and vary the trust in the log.

Response Variables. As our response in the EoT experiments we consider the three-way similarity (TWS) as follows. For each of the experiments we calculated the inverts of the three quality measures that we consider in Problem 1, namely $1 - \delta_{\mathcal{L}^2}(L^*, L)$, $1 - \delta_{\mathcal{LM}}(L^*, T^*)$ and $1 - \delta_{\mathcal{M}^2}(T, T^*)$, turning distances into similarities. The TWS response is the average of these similarity measures. For the MRQ and LRQ experiments we measured replay fitness, precision, and generalization as presented in Section 3.

5.2 Input Logs and Models

Event Logs. We use the following sample of event logs to evaluate the approach:

a) Real-World Data of DayHospital DayHospital is an outpatient clinic located in the United States. Approximately 250 patients arrive daily to receive treatment from 300 healthcare providers. Patients stay in the hospital for an average time of 4.4 hours and typically go through vital signs collection performed by a Clinical Assistant (CA) and an Infusion performed by an infusion nurse (InfRN). The log contains $4,281$ traces with a total of $26,286$ events and 17 event classes (single month of data). Traces are sequences of roles that correspond to the aforementioned activity. The most frequent trace $\langle Arrival, CA, InfRN, End\rangle$ appears 943 times and represents 22% of the event log.

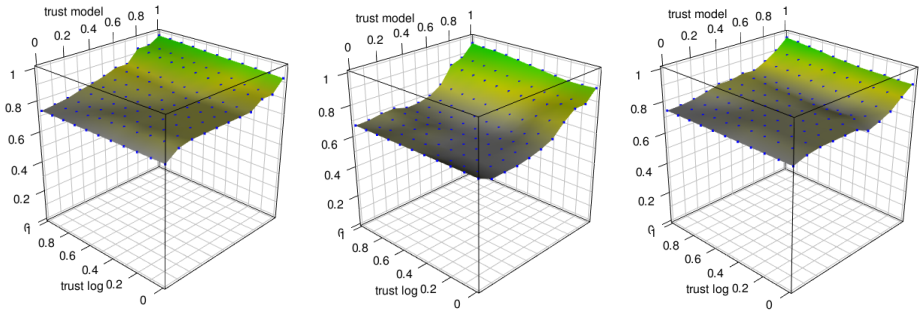
b) Student data from an e-Learning platform The second data set corresponds to an event log from a university. The university records whenever students take course exams. Specifically, students need to select two specializations (SBWLs). Each of the SBWLs consists of five courses. Exams for these courses can be taken individually (Courses I to V) or for multiple of the five at once (Examination) [22]. This event log contains $2,777$ traces with a total of $10,590$ events, and 6 event types (one for each type of the courses). The most frequent trace is Course I, Course V, Examination with 499 occurrences that amount to almost 18% of the traces.

c) The Loan Application Process BPIC 2012 This log is taken from the BPI challenge of 2012 [8]. The log stems from a financial company handling a loan application process. We only consider the top level process, i.e., we apply a log filter to retain only the events whose labels start with "A_". This results in $13,087$ traces with a total of $60,849$ events, 10 distinct event types, and 15 trace variants. Further, most cases (43.7%) are declined and emit the trace $\langle A{-}SUBMITTED,\ A{-}PARTLYSUBMITTED,\ A{-}DECLINED\rangle$.

Corresponding Models. In all our experiments, the input models, T, are constructed as follows. In order to avoid a fully aligned pair of model and log, we set the input model to be the "happy path" model, that is, the process tree capturing the sequence $(\rightarrow)$ of the events in the log that corresponds to the most frequent trace $\sigma^* = \arg\max_{\sigma \in L} L(\sigma)$. Note that these models correspond to business process models as captured by process analysts in their first modeling attempt.

5.3 Results

On the Effect of Trusts (EoT). Figure 3 shows the three surfaces (one for each log) that represent the TWS as function of the two trust levels. Each point on the surface corresponds to two trust levels, and the resulting similarity average. We observe that the best scores are achieved by assigning full trust levels $\tau_L = \tau_M = 1$. Here, we start with small “happy path” models that represent the most frequent behavior in the log. With full trusts in model and log, we return the input log-model pair and therefore, $\delta_{\mathcal{L}^2}$ and $\delta_{\mathcal{M}^2}$ are zero. The distance between a “happy path” model and the event log is also not high (although fitness is not optimal, precision and generalization are high). Therefore, the result is a high TWS score. Note that there is a general deterioration as the trust in



(a) DayHospital log score. (b) eLearning log score. (c) BPIC'12 log score.

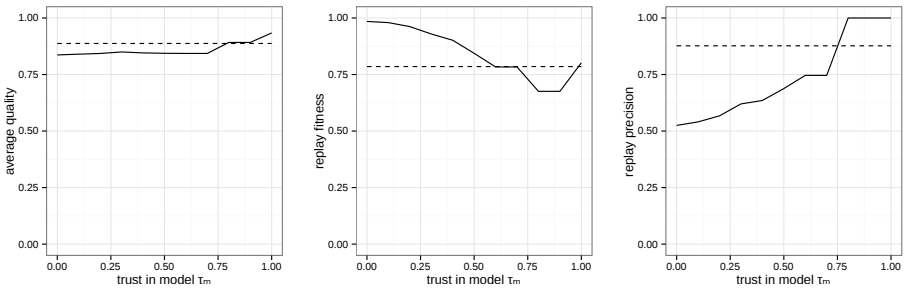
Fig. 3: Depending on the different trust levels the weighted average score of log similarity, model similarity and log-model alignment is shown on the vertical axis. The controlled trust levels are presenting on the remaining two axes.

the log and the model decreases. This yields a new pair of model-log that may have any similarity between 0 and 1, especially because any change to a small model is a relatively costly.

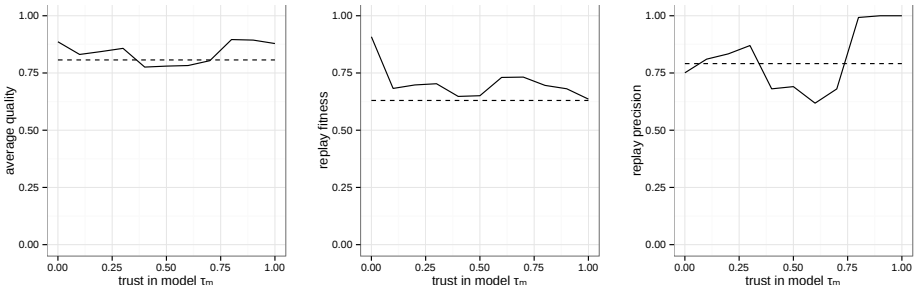
An interesting result is depicted in Figure 3b. One may expect a monotonic decline of the TWS when going from full trust levels and no-trust levels. However, in Figure 3b we observe that when lowering trust levels in the model, we pass a valley in quality to later reach again better results. Therefore, one may have to move away from a current local optimal solution, through a set of sub-optimal solutions to reach an optimum. This provides evidence in favor of the trust-based setting that we propose in the GenCon problem, using trust levels to parameterize our search for optimal log/ model matching.

On Model Repair Quality (MRQ). Figure 4 presents the model repair quality evaluation with respect to replay fitness, precision, generalization and their average. Specifically, it shows the results for setting trust in the log $\tau_L = 1$ and controlling only trust in the model τ_M between 1 to 0. We compare the quality of our approach to a state-of-the-art model repair technique with default settings as presented in [9]. The average quality (leftmost chart for all logs) is the average of the three measures, namely fitness, precision, and generalization. We also present fitness (middle) and precision (right). The generalization remained steady at 1.0 for all models.

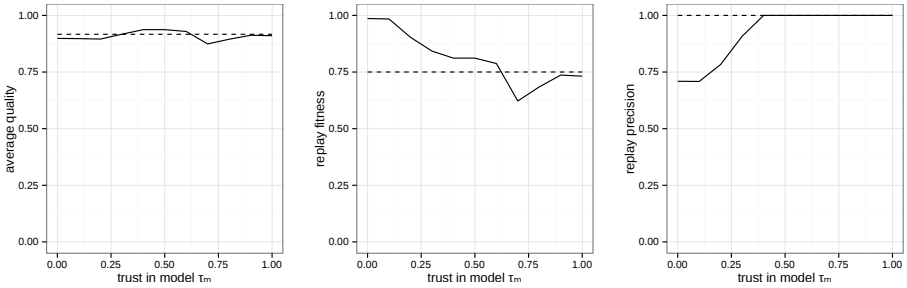
We see that the average quality is comparable to the model repair technique from [9], with a slight deterioration for the hospital log as the trust in model decreases. Unsurprisingly, replay fitness tends to improve when stepping away from the initial model T towards T_L, while precision declines as more behaviour is allowed (we always start with a simple model). Note however that some intermediate steps worsen the results temporarily. For the loan application log we observe that the best model results in the trust area of 0.5, which indicates that one should neither mistrust the initial model (T) completely, nor trust it to the full extent. An uninformative prior of $\tau_M = 0.5$ yields the best result.



(a) DayHospital model repair



(b) eLearning model repair



(c) BPIC'12 model repair

Fig. 4: Model repair with different trust levels. The graphs show the resulting averages for (left to right): (1) average of fitness, precision, generalization, (2) replay fitness, and (3) precision. Dashed lines show competing results based on applying [9].

On Log Repair Quality (LRQ). Repairing the behavior of an event log cannot be compared to any state-of-the-art algorithm. Figure 5 presents the average of fitness, precision, and generalisation for model's full trust $\tau_M = 1$, while we vary levels of log trust. For full log trust, we do not repair the log, and calculate the three qualities based on the input model. Here, we see a smooth transition between different trust levels, which shows that we can smoothly select the ratio of traces which we mistrust and correct them according to the model.

This result provides evidence for the consistency of our approach by showing that the resulting logs are in high proximity to the input model (that is fully trusted), whenever the log trust is less than 1.

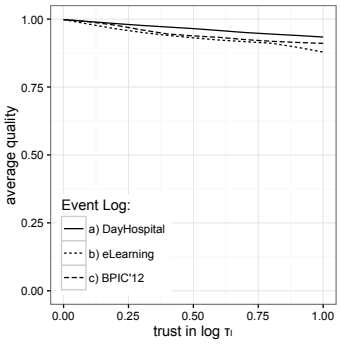


Fig. 5: Log repair results. The x-axis depicts the controlled variable τ_L; the y-axis is the average of fitness, precision, and generalization.

6 Conclusion

In this work, we presented the task of generalized conformance checking and formalized the GenCon problem. It strives for a balance between two independent input parameters: the trust in the log quality, and the trust in the model quality. Specifically, when presented with an event log and a process model, generalized conformance checking attempts at repairing both according to the initial trust levels, and returns an improved log-model pair. We instantiated the GenCon problem with process trees, and with distance measures based on (log or tree) edit operations and alignments between a log and a process tree. Further, we proposed a technique to obtain the improved log-model pair by first lifting the log into the model space, and applying a greedy heuristic to search for the best model in the model space. The improved event log is obtained by aligning it to the resulting best model. An evaluation with real-world datasets demonstrates evidence in favor of the proposed trust-based approach. Further, generalized conformance checking is comparable to state-of-the-art model repair techniques in model quality measures.

One limitation of our approach is that we use a sub-optimal search for the best model. Although state-of-the-art techniques for optimal search are notorious for their steep run-time costs, recent ideas in the direction of automated planning demonstrate positive empirical results with respect to time complexity [7]. In future work, we aim at applying these techniques to the model search problem. Further, a key insight from our experiments is that process trees are overly conservative in their allowed behavior. A more flexible model representation might allow finer grained model repairs, even for high levels of model trust. Therefore, we aim at addressing the GenCon problem for other model spaces, e.g. workflow nets. Last, we plan to lift generalized conformance checking to operational process models, such as Generalized Stochastic Petri Nets.

Acknowledgments. This work was partially supported by the European Union's Seventh Framework Programme (FP7/2007-2013) grant 612052 (SERAMIS) and the German Research Foundation (DFG), grant WE 4891/1-1.

References

1. van der Aalst, W.M.P.: Process mining: discovery, conformance and enhancement of business processes. Springer Science & Business Media (2011)
2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying History on Process Models for Conformance Checking and Performance Analysis. WIREs: Data Mining and Knowledge Discovery 2, 182–192 (2012)

3. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. Knowledge and Data Engineering, IEEE Transactions on 16(9), 1128–1142 (2004)
4. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: Evolutionary Computation (CEC'12). pp. 1–8. IEEE (2012)
5. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. International Journal of Cooperative Information Systems 23(01), 1440001 (2014)
6. Buijs, J.C.A.M., La Rosa, M., Reijers, H.A., van Dongen, B.F., van der Aalst, W.M.P.: Improving business process models using observed behavior. In: Data-Driven Process Discovery and Analysis, pp. 44–59. Springer (2012)
7. Domshlak, C., Mirkis, V.: Deterministic oversubscription planning as heuristic search: Abstractions and reformulations. J. Artif. Intell. Res. (JAIR) 52, 97–169 (2015)
8. van Dongen, B.: BPI Challenge 2012 (2012), http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
9. Fahland, D., van der Aalst, W.M.: Model repair — aligning process models to reality. Information Systems 47, 220 – 243 (2015)
10. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In: Business Process Management, pp. 328–343. Springer (2007)
11. Kunze, M., Weidlich, M., Weske, M.: Behavioral similarity–a proper metric. In: Business Process Management, pp. 166–181. Springer (2011)
12. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - A constructive approach. In: Application and Theory of Petri Nets and Concurrency, PETRI NETS. LNCS, vol. 7927, pp. 311–329 (2013)
13. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Business Process Management Workshops. pp. 66–78. Springer (2013)
14. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing 98(4), 407–437 (2016)
15. Marquard, M., Shahzad, M., Slaats, T.: Web-based modelling and collaborative simulation of declarative processes. In: Business Process Management, pp. 209–225. Springer (2015)
16. Pawlik, M., Augsten, N.: Tree edit distance: Robust and memory-efficient. Information Systems 56, 157 – 173 (2016)
17. Rogge-Solti, A., Kasneci, G.: Temporal anomaly detection in business processes. In: Business Process Management, pp. 234–249. Springer (2014)
18. Rogge-Solti, A., Mans, R.S., van der Aalst, W.M.P., Weske, M.: Improving documentation by repairing event logs. In: The Practice of Enterprise Modeling, pp. 129–144. Springer (2013)
19. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64 – 95 (2008)
20. Senderovich, A., Weidlich, M., Yedidsion, L., Gal, A., Mandelbaum, A., Kadish, S., Bunnell, C.A.: Conformance checking and performance improvement in scheduled processes: A queueing-network perspective. Information Systems Forthcoming (2016)
21. Wang, J., Song, S., Lin, X., Zhu, X., Pei, J.: Cleaning structured event logs: A graph repair approach. In: Data Engineering (ICDE'15). pp. 30–41. IEEE (2015)
22. Weber, I., Farshchi, M., Mendling, J., Schneider, J.: Mining processes with multi-instantiation. In: 30th Annual ACM Symposium on Applied Computing. pp. 1231–1237 (2015)
23. Weijters, A.J.M.M., van der Aalst, W.M.P., De Medeiros, A.K.A.: Process mining with the heuristics miner-algorithm. Tech. Rep. 166 (2006)
24. Whitley, D.: An overview of evolutionary algorithms: practical issues and common pitfalls. Information and software technology 43(14), 817–831 (2001)