J Intell Inf Syst manuscript No.
(will be inserted by the editor)

Improving process models by mining mappings of low-level events to high-level activities

Diogo R. Ferreira · Fernando Szimanski · Célia G. Ralha

Received: date / Accepted: date

Abstract While it is possible to analyze the run-time behavior of a business process through process mining techniques, in practice there is often a gap between the low-level nature of the events recorded in an event log and the high-level of abstraction at which the process is modeled. This makes it difficult to understand the recorded behavior in terms of the high-level activities in the process model. Also, it makes it difficult to improve the model based on run-time data about the process. In this work we present an approach to mine mappings between the events in the log and the activities in the model. These mappings can be used to generate suggestions of how the process model can be extended in order to capture the behavior recorded in the event log. Using a real-world and publicly available event log, we show how the approach can improve the model in a stepwise manner, until it covers all the behavior recorded in the event log.

Keywords Business Process Modeling · Process Mining · Model Enhancement · Recommendation Systems

1 Introduction

Process mining [23] is a new and emergent area, where the main focus is on extracting information about the run-time behavior of business processes from event logs recorded by the information systems available in an organization. Process mining can be used for three different purposes [26]:

D. R. Ferreira (✉)
Instituto Superior Técnico (IST), Universidade de Lisboa, Portugal
E-mail: diogo.ferreira@tecnico.ulisboa.pt

F. Szimanski · C. G. Ralha
Departamento de Ciência da Computação, Universidade de Brasília (UnB), Brazil
E-mail: fszimanski@gmail.com

C. G. Ralha
E-mail: ghedini@cic.unb.br

- *Discovery*, which consists in deriving a process model from the behavior recorded in an event log, without using any previous knowledge about the process which generated such event log. There are many techniques for this purpose, for example the α-algorithm [25], the heuristics miner [30], or the genetic miner [7].
- *Conformance*, which consists in comparing the behavior recorded in an event log with an existing process model, in order to determine to what extent the process is being performed as described in the existing model. There are special metrics for conformance checking [18], and there are also ways to replay the event log in the process model in order to check conformance [28].
- *Enhancement*, which consists in improving or extending an existing process model based on the run-time behavior recorded in an event log. For example, if a process has a decision between two branches at some point, it will be possible to determine the exact percentage of cases that follow each branch, or even to discover other branches which were not in the original model. There has been comparatively less work on enhancement than on discovery or conformance.

This work focuses on model enhancement, and particularly on the problem of how to enhance a given process model when there is a mismatch between the high-level activities in the model and the low-level events that are recorded in the event log.

1.1 Problem description

To illustrate the problem, consider the simple example in Figure 1. Here the process model is expressed in terms of the high-level activities A, B and C. However, when these activities are performed, they generate low-level events, namely U, V, W, X, Y and Z. In particular, activity A generates the sequence UV; B generates WXX...; and C generates YZYZ... A sample trace from such process could be: UVWXXYZYZ.

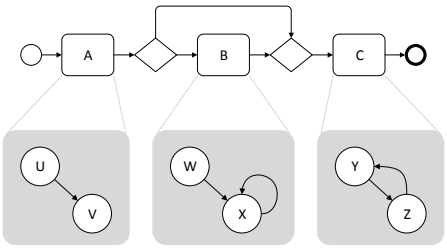


Fig. 1: An simple example of how high-level activities may generate low-level events

In this example, we have purposefully ensured that each kind of event (such as U, V, W, etc.) can belong to one and only one activity. The more general case of allowing an event to be produced by multiple activities (such as U being produced both by A and B) has been addressed in previous work, and we will discuss that shortly. Here, the fact that each event belongs to a single activity will allow us to establish the concept of *mapping* in this work.

To illustrate the kind of model enhancement we seek here, suppose that, initially, the process is thought to allow the sequence ABC only. However, suppose that in the event log

we find the sequences UVWXXYZYZ and UVYZYZ. If the following mapping is true $\{\mathsf{U} \mapsto \mathsf{A}, \mathsf{V} \mapsto \mathsf{A}, \mathsf{W} \mapsto \mathsf{B}, \mathsf{X} \mapsto \mathsf{B}, \mathsf{Y} \mapsto \mathsf{C}, \mathsf{Z} \mapsto \mathsf{C}\}$ then one must conclude that the sequence of events UVYZYZ must have been produced by the sequence of activities AC. Since AC is not in the original model, we can add it by allowing B to be skipped, as in the model of Figure 1. The model has been enhanced with a new possible route. Furthermore, if we observe an equal number of UVWXXYZYZ and UVYZYZ traces in the event log, we can conclude that 50% of cases follow the route through B, and 50% of cases follow the route around B. Such information is not usually available in the process model, so we consider it to be another kind of enhancement.

The problem with this approach is that, to begin with, the mapping of low-level events to high-level activities is unknown. This mapping must be mined from the event log and from the given process model. Also, the enhancements that are going to be suggested for the process model depend on the particular mapping being used. Therefore, in case there are multiple possible mappings (as is usually the case), one must do a careful choice of mapping.

This work provides a solution to this problem, i.e. it presents an approach that is able to mine possible mappings, select a candidate mapping, and provide suggestions as to how the process model can be enhanced in order to capture the behavior observed in the event log.

1.2 Previous work

In previous work we have introduced a hierarchical Markov model [8, 9] to capture the relationship between high-level activities and low-level events, and in [21] we introduced an approach for improving process models which has the hierarchical Markov model at its core. Through a combination of process mining and agent-based simulation, we described how a process model can be mined and improved iteratively, where at each step a new event log is generated by agent-based simulation.

However, the application of such approach in real-world experiments involving noisy event logs, such as in the Volvo IT case study that we present here, produced models that were difficult to interpret by the business analyst, because each high-level activity ended up containing a complex aggregate of low-level behavior, when the analyst was expecting, in some cases, a one-to-one mapping between a low-level event and a high-level activity.

In other words, the approach was capturing patterns of behavior which were too complex to interpret because each event could be produced by multiple activities and each activity contained a different pattern involving many kinds of low-level events. It became clear to us that, in some situations, the business analyst tends to see certain low-level events as being associated with a single, particular activity. When attempting to embed this restriction in our previous approach, we found that not only it changed the nature of the underlying model, but it also opened the way for using an entirely different approach to mine that model.

Therefore, while in [21] we used a hierarchical Markov model, here we replaced that model with the new concept of mapping, which serves a similar purpose, but has some distinct advantages (one of which is that there is no need for preprocessing of the event log, since the approach can handle any amount of noise). Also, while in [21] we used an iterative approach with agent-based simulation, here we opted for a greedy optimization approach which offers more guarantees of finding meaningful improvements to the process model. As a by-product of this new approach, we are able to automatically generate suggestions for the user regarding possible improvements to the process model.

1.3 Structure of presentation

The structure of this paper is as follows. Section 2 provides a formal definition of what a mapping is, and describes how to find all mappings from a given set of traces and from a set of possible paths through the process model. Then Section 3 explains how to select the best mapping based on concepts such as range and coverage; this task is complicated by the fact that it may be possible to combine existing mappings in order to create an even better mapping. Section 4 presents a case study where we apply the approach to a real-world event log, showing how the simplest possible model can be improved in a stepwise manner until it covers all the behavior in the event log (including all noise). Finally, Section 5 discusses several branches of related work, and Section 6 summarizes and concludes the paper.

2 The concept of mapping

In general, business processes are defined at a high level of abstraction, using modeling languages such as BPMN [17], EPCs [19], and Petri nets [27]. However, when looking at an event log, the recorded behavior often refers to low-level events which have no obvious relationship to the high-level activities defined in a process model.

A mapping is a way to establish a relationship between each kind of event observed in the event log and a specific high-level activity defined in the business process. In particular, an event can be mapped to one and only one activity. Even though this assumption may seem limiting at first sight, in practice there is more than one way to make the assumption hold, as the following example illustrates.

Example 1 *Suppose that one observes the trace* $UVWXUVYZ$ *in the event log. This trace does not seem to fit the sequence* ABC *if the mapping is* $\{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$. *If one is to believe this mapping, then it suggests that* $UVWXUVYZ$ *was generated by* $ABAC$, *and therefore the process model should include this possibility. Alternatively, if one knows that the sequence* $ABAC$ *cannot take place, then it is possible to come up with a different mapping to explain how the trace* $UVWXUVYZ$ *was generated from the current model. Such a mapping is* $\{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto A, Y \mapsto C, Z \mapsto C\}$, *implying that* $UVWXUVYZ$ *was generated by the sequence* AC.

As illustrated in the example above, the problem of mapping an event to a single activity can be turned into a problem of whether the current model or the current mapping is actually a good description for the process. Since a process model can be freely designed or redesigned to accommodate the behavior observed in an event log (if it cannot be freely redesigned, them probably it cannot be enhanced), we will keep the assumption that an event can be mapped to a single activity, leaving it up to the process analyst to come up with a model that is coherent with this assumption.

The following definitions explain in more formal terms what a mapping actually is:

Definition 1 (Process model) *A process model is any kind of state machine defined in a high-level modeling language. Each state in the process model is referred to as an activity.*

Definition 2 (Activity) *An activity represents an abstract unit of work, such as a human task in the process. Even though an activity may be regarded as a single step in the process, its execution may generate multiple events.*

Definition 3 (Event) *An event is defined as a tuple $e = (u, t)$ where u is a symbol denoting the type of event, and t is the timestamp of the event. Let $\pi_1(\cdot)$ be a function (i.e. projection) that returns the symbol of the event, and let $\pi_2(\cdot)$ be a function that returns the timestamp of event. For $e = (u, t)$ we have $\pi_1(e) = u$ and $\pi_2(e) = t$.*

Definition 4 (Trace) *A trace is a sequence of events $\tau = \langle e_1, e_2, \ldots, e_n \rangle$ such that $\forall_{i<j}$ $\pi_2(e_i) \leq \pi_2(e_j)$, i.e. the events are ordered by timestamp.*

Definition 5 (Micro-sequence) *A micro-sequence is a sequence of symbols which is obtained by projection of a trace: $\mu = \langle \pi_1(e_1), \pi_1(e_2), \ldots, \pi_1(e_n) \rangle$ such that $\forall_{i<j}\ \pi_2(e_i) \leq \pi_2(e_j)$. In other words, a micro-sequence corresponds to the sequence of symbols in a trace.*[1]

Definition 6 (Macro-sequence) *A process model may allow one or more paths of execution. Each path corresponds to a sequence of activities $\sigma = \langle a_1, a_2, \ldots, a_m \rangle$. As these activities are performed, multiple events are generated. The sequence of events that is generated by executing a sequence of activities σ is a trace τ (see Def. 4 above).*

Definition 7 (Mapping) *Let $\mu = \langle u_1, u_2, \ldots, u_n \rangle$ be a micro-sequence, which is the projection of some trace τ. Let $\sigma = \langle a_1, a_2, \ldots, a_m \rangle$ be the macro-sequence which generated the trace τ. Now, let U be the set of symbols in the micro-sequence μ, i.e. $U = \{u \mid u \in \mu\}$, and let A be the set of activities in the macro-sequence σ, i.e. $A = \{a \mid a \in \sigma\}$.*

Then a mapping is defined as a function $f : U \to A$ which maps each symbol $u \in U$ to an activity $a \in A$.

A mapping f can also be represented as a set in the form: $\lambda = \{u \mapsto f(u) \quad | \quad u \in U \ \wedge \ f(u) \in A\}$.

For a given micro-sequence and macro-sequence, there can be multiple possible mappings of events (in the micro-sequence) to activities (in the macro-sequence). However, the set of possible mappings is definitely finite and restricted by the given macro-sequence. The following examples illustrate this point.

Example 2 *Given the micro-sequence UVWXXYZYZ and the macro-sequence ABC, the possible mappings are:*

$\lambda_1 = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto A, Y \mapsto A, Z \mapsto A\}$
$\lambda_2 = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto A, Y \mapsto B, Z \mapsto B\}$
$\lambda_3 = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto B, Y \mapsto B, Z \mapsto B\}$
$\lambda_4 = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\lambda_5 = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto B, Y \mapsto B, Z \mapsto B\}$
$\lambda_6 = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\lambda_7 = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto C, Y \mapsto C, Z \mapsto C\}$
$\lambda_8 = \{U \mapsto A, V \mapsto B, W \mapsto B, X \mapsto B, Y \mapsto B, Z \mapsto B\}$
$\lambda_9 = \{U \mapsto A, V \mapsto B, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\lambda_{10} = \{U \mapsto A, V \mapsto B, W \mapsto B, X \mapsto C, Y \mapsto C, Z \mapsto C\}$
$\lambda_{11} = \{U \mapsto A, V \mapsto B, W \mapsto C, X \mapsto C, Y \mapsto C, Z \mapsto C\}$

Example 3 *For the micro-sequence UVWXXYZYZ and the macro-sequence ABC, here are some examples of impossible mappings:*

$\lambda_{12} = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto A, Y \mapsto A, Z \mapsto B\}$

[1] For simplicity, in the text we often refer to a micro-sequence as a "sequence of events", even though what we actually mean is the "sequence of symbols" (μ), to be fully precise.

(impossible because the end of the micro-sequence YZYZ would be mapped to ABAB, which is not allowed by the macro-sequence, since A cannot occur after B)

$\lambda_{13} = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto B, Y \mapsto B, Z \mapsto C\}$

(impossible because the end of the micro-sequence YZYZ would be mapped to BCBC, which is not allowed by the macro-sequence, since B cannot occur after C)

$\lambda_{14} = \{U \mapsto B, V \mapsto B, W \mapsto B, X \mapsto C, Y \mapsto C, Z \mapsto C\}$

(impossible because U at the beginning of the micro-sequence would be mapped to B, which is not allowed since the macro-sequence cannot begin with B)

$\lambda_{15} = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto B, Y \mapsto A, Z \mapsto A\}$

(impossible because XY in the micro-sequence would be mapped to BA , which is not allowed by the macro-sequence, since A cannot occur after B)

Not all mappings in Example 2 are interesting. Mapping λ_1, while certainly possible, is not interesting because every event has been mapped to activity A. Similarly, mappings λ_2, λ_3, λ_5, and λ_8 are not interesting because not every activity in the macro-sequence appears in the mapping. In this work we are interested in complete mappings, meaning that the mapping contains every activity in the macro-sequence.

Definition 8 (Complete mapping) *Recall that U is the set of symbols in the micro-sequence and A is the set of activities in the macro-sequence. A mapping is said to be complete if and only if the function $f : U \to A$ is surjective, i.e. for every activity $a \in A$ there exists an event $u \in U$ such that $f(u) = a$.*

The following example illustrates that not all of the mappings in Example 2 are admissible according to Definition 8.

Example 4 *Given the micro-sequence UVWXXYZYZ and the macro-sequence ABC, the complete mappings are:*

$\lambda_4 = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\lambda_6 = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\lambda_7 = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto C, Y \mapsto C, Z \mapsto C\}$
$\lambda_9 = \{U \mapsto A, V \mapsto B, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\lambda_{10} = \{U \mapsto A, V \mapsto B, W \mapsto B, X \mapsto C, Y \mapsto C, Z \mapsto C\}$
$\lambda_{11} = \{U \mapsto A, V \mapsto B, W \mapsto C, X \mapsto C, Y \mapsto C, Z \mapsto C\}$

2.1 Finding all complete mappings

The problem of finding all complete mappings for a given micro-sequence and macro-sequence is relatively easy to solve. Basically, one starts by assigning the first event in the micro-sequence to the first activity in the macro-sequence. For each subsequent event, if it has not been mapped already, then it can be mapped either to the current activity in the macro-sequence or to the next activity in the macro-sequence (by advancing the current position in the macro-sequence). By the time the end of the micro-sequence is reached, the end of the macro-sequence should have been reached as well. If this is true, then the mapping is complete.

Algorithm 1 specifies this procedure. Besides the micro-sequence μ and the macro-sequence σ, the procedure has the following parameters: Λ is the set of complete mappings, which is initially empty ($\Lambda = \{\}$); i is the current position in the micro-sequence, which begins at the first position ($i = 1$); j is the current position in the macro-sequence, which

begins at the first position ($j = 1$); and λ is the current mapping, which is initially empty ($\lambda = \{\}$) and is built during recursion. At the end of the micro-sequence, if λ is a complete mapping, then it will be added to Λ (line 18).

Algorithm 1 Find all complete mappings for a given micro-sequence $\mu = \{u_1, u_2, \ldots, u_n\}$ and a given macro-sequence $\sigma = \{a_1, a_2, \ldots, a_m\}$ where $m \leq n$

```
1:  procedure MAPPINGS(μ, σ, Λ={}, i=1, j=1, λ={})
2:      if i = 1 then
3:          λ ← {u_i ↦ a_j}
4:          MAPPINGS(μ, σ, Λ, i+1, j, λ)
5:      if 1 < i ≤ n then
6:          if (u_i ↦ a_j) ∈ λ then
7:              MAPPINGS(μ, σ, Λ, i+1, j, λ)
8:          else if (j < m) ∧ (u_i ↦ a_{j+1}) ∈ λ then
9:              MAPPINGS(μ, σ, Λ, i+1, j+1, λ)
10:         else
11:             λ' ← λ ∪ {u_i ↦ a_j}
12:             MAPPINGS(μ, σ, Λ, i+1, j, λ')
13:             if (j < m) then
14:                 λ' ← λ ∪ {u_i ↦ a_{j+1}}
15:                 MAPPINGS(μ, σ, Λ, i+1, j+1, λ')
16:     if i > n then
17:         if j = m then
18:             Λ ← Λ ∪ {λ}
19:         return
```

The algorithm begins by checking if i is at the beginning of the micro-sequence (line 2). In this case, the first event is mapped to the first activity (line 3), and the same procedure is called for the next position in the micro-sequence (line 4).

For $i > 1$ (line 5), it may be the case that the event has been mapped already or not:

- If it has already been mapped, then the only viable options are that it has been mapped to the current activity (line 6) or to the next activity (line 8).
- If the event has not been mapped already, then the same possibilities should be considered, i.e. either mapping it to the current activity (line 11), or mapping it to the next activity (line 14).

In any of these cases, the procedure advances to the next position ($i+1$) in the micro-sequence (lines 7, 9, 12, 15).

If i gets past the end of the micro-sequence (line 16), then all events have been mapped, and the last thing to check is if j reached the end of the macro-sequence (line 17). This ensures that the mapping is complete.

Taking into account that each symbol (u_i) in the micro-sequence can be mapped either to the current activity (a_j) or to the next (a_{j+1}), at first sight it would seem that the complexity of Algorithm 1 would be of the order of $O(2^n)$, where n is the length of the micro-sequence. However, once a symbol has been mapped to an activity, every occurrence of that symbol in the micro-sequence has already been mapped and there is no other choice for that symbol, so the complexity is actually $O(2^{|U|})$ where $|U|$ (i.e. the size of U) is the number of distinct symbols in the micro-sequence. Additionally, the first symbol in the micro-sequence is mapped to the first activity in the macro-sequence, so there is only one choice for that symbol. Therefore, the complexity of Algorithm 1 is $O(2^{|U|-1})$.

Still, many branches in the recursion get pruned because, in general, if is often the case that a symbol u_i can be mapped to only one of the two possibilities a_j and a_{j+1}, or even to none of them. As an example, for the micro-sequence UVWXXYZYZ and the macro-sequence ABC, there are only 11 mappings remaining at the end of the micro-sequence (see Example 2); this is less than what the estimate $2^{|U|-1} = 2^5 = 32$ would suggest.

2.2 Working with multiple macro- and micro-sequences

In practice, an event log may contain several different micro-sequences, and each of these micro-sequences may occur more than once. We define the *support* of a micro-sequence as the number of times that the micro-sequence appears in an event log.

Definition 9 (Event log) *An event log L is defined as a bag of micro-sequences, e.g. $L = [\mu_1, \mu_2, \mu_1, \mu_3, \mu_2, \ldots]$, where these micro-sequences are not necessarily different (i.e. there can be duplicates among them).*

Definition 10 (Support) *The support of a micro-sequence is the number of times that μ appears in an event log L, and it is denoted by $\mu@L$. Let $q_i = \mu_i@L$ be the support of μ_i in L. Then an alternative representation for L is $L = [\mu_1^{q_1}, \mu_2^{q_2}, \mu_3^{q_3}, \ldots]$ where all micro-sequences are now different.*

Also, as explained in Definition 6, a process model may have several possible paths of execution and, typically, each of these paths generates a different macro-sequence. We define the *macro-model* as the set of possible macro-sequences that can be generated by a process model.

Definition 11 (Macro-model) *A macro-model is the set of all macro-sequences $M = \{\sigma_1, \sigma_2, \ldots\}$ that can be generated by a given process model.*

In general, the macro-model can be obtained from any kind of process model (BPMN, EPC, Petri net, etc.) by following all possible paths of execution. Even if the process model contains loops, which could originate an infinite set of macro-sequences, it is usually possible to establish a maximum number of possible iterations for any given loop. In fact, it would be unrealistic to think that a loop in a business process can run an infinite number of times.

In Definitions 9 and 10, we have seen that an event log may contain several different micro-sequences. Now in Definition 11 we see that a process model may originate several different macro-sequences. This means that, if we are given an event log and a process model, we can obtain the bag of micro-sequences L and the set of macro-sequences M. However, in general we do not know which macro-sequence in M produced each micro-sequence in L.

Let $L = [\mu_1^{q_1}, \mu_2^{q_2}, \mu_3^{q_3}, \ldots]$ and $M = \{\sigma_1, \sigma_2, \ldots\}$. If we are interested in finding all possible mappings, then we need to consider the possibility that each micro-sequence $\mu_i \in L$ might have been generated by any of the macro-sequences $\sigma_j \in M$. Therefore, we need to extend Algorithm 1 in order to consider all of these possibilities.

Algorithm 2 provides a function that returns all complete mappings as a table T of records in the form $(\mu_i, \sigma_j, \lambda_k)$ where μ_i is a micro-sequence, σ_j is a macro-sequence, and λ_k is a mapping between μ_i and σ_j. There may be several possible mappings between μ_i and σ_j, so table T may contain several records for the same μ_i and σ_j, as illustrated in Example 5.

Algorithm 2 Find all complete mappings for a given event log $L = [\mu_1^{q_1}, \mu_2^{q_2}, \mu_3^{q_3}, \ldots]$ and macro-model $M = \{\sigma_1, \sigma_2, \ldots\}$

```
1: function AllMappings(L, M)
2:     T ← {}
3:     for each μ_i ∈ L do
4:         for each σ_j ∈ M do
5:             Λ ← {}
6:             Mappings(μ_i, σ_j, Λ)
7:             for each λ_k ∈ Λ do
8:                 T ← T ∪ {(μ_i, σ_j, λ_k)}
9:     return T
```

Example 5 *Let $L = [\mu_1^{q_1}, \mu_2^{q_2}]$ be an event log with the micro-sequences $\mu_1 =$ UVWXXYZYZ and $\mu_2 =$ UVYZYZ, and some arbitrary q_1 and q_2. Also, let $M = \{\sigma_1, \sigma_2\}$ be a macro-model with the macro-sequences $\sigma_1 =$ ABC and $\sigma_2 =$ AC.*

Then Algorithm 2 produces the following table of mappings between each micro-sequence μ_i and each macro-sequence σ_j:

μ_i	σ_j	λ_k
$\mu_1 =$ UVWXXYZYZ	$\sigma_1 =$ ABC	$\lambda_1 = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\mu_1 =$ UVWXXYZYZ	$\sigma_1 =$ ABC	$\lambda_2 = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\mu_1 =$ UVWXXYZYZ	$\sigma_1 =$ ABC	$\lambda_3 = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto C, Y \mapsto C, Z \mapsto C\}$
$\mu_1 =$ UVWXXYZYZ	$\sigma_1 =$ ABC	$\lambda_4 = \{U \mapsto A, V \mapsto B, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\mu_1 =$ UVWXXYZYZ	$\sigma_1 =$ ABC	$\lambda_5 = \{U \mapsto A, V \mapsto B, W \mapsto B, X \mapsto C, Y \mapsto C, Z \mapsto C\}$
$\mu_1 =$ UVWXXYZYZ	$\sigma_1 =$ ABC	$\lambda_6 = \{U \mapsto A, V \mapsto B, W \mapsto C, X \mapsto C, Y \mapsto C, Z \mapsto C\}$
$\mu_1 =$ UVWXXYZYZ	$\sigma_2 =$ AC	$\lambda_7 = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto A, Y \mapsto C, Z \mapsto C\}$
$\mu_1 =$ UVWXXYZYZ	$\sigma_2 =$ AC	$\lambda_8 = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto C, Y \mapsto C, Z \mapsto C\}$
$\mu_1 =$ UVWXXYZYZ	$\sigma_2 =$ AC	$\lambda_9 = \{U \mapsto A, V \mapsto A, W \mapsto C, X \mapsto C, Y \mapsto C, Z \mapsto C\}$
$\mu_1 =$ UVWXXYZYZ	$\sigma_2 =$ AC	$\lambda_{10} = \{U \mapsto A, V \mapsto C, W \mapsto C, X \mapsto C, Y \mapsto C, Z \mapsto C\}$
$\mu_2 =$ UVYZYZ	$\sigma_1 =$ ABC	$\lambda_{11} = \{U \mapsto A, V \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\mu_2 =$ UVYZYZ	$\sigma_2 =$ AC	$\lambda_{12} = \{U \mapsto A, V \mapsto A, Y \mapsto C, Z \mapsto C\}$
$\mu_2 =$ UVYZYZ	$\sigma_2 =$ AC	$\lambda_{13} = \{U \mapsto A, V \mapsto C, Y \mapsto C, Z \mapsto C\}$

2.3 Subsumption and compatibility

In table T it is possible that the same mapping λ_k applies to different pairs of micro-sequences and macro-sequences. For example, suppose that T contains the tuples $(\mu_1, \sigma_1, \lambda_3)$ and $(\mu_2, \sigma_2, \lambda_3)$; then λ_3 covers both μ_1 and μ_2. In fact, this is not only *possible* but it is also *desirable* to happen since, ideally, we would like to be able to cover all micro-sequences with the same mapping.

Now, suppose that T contains the tuples $(\mu_1, \sigma_1, \lambda_3)$ and $(\mu_2, \sigma_2, \lambda_4)$ and that λ_4 is "contained" in λ_3 in the sense that every pair $u \mapsto a$ that appears in λ_4 also appears in λ_3 (but λ_3 may contain more). Then we say that λ_3 *subsumes* λ_4. It is clear that if λ_3 subsumes λ_4 then λ_3 covers all the micro-sequences which are covered by λ_4. The following example illustrates this point.

Example 6 *Let UVWXXYZYZ be a micro-sequence which can be mapped to the macro-sequence ABC through the following mapping:*

$$\lambda_1 = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$$

Let UVYZYZ be a micro-sequence which can be mapped to the macro-sequence AC through the following mapping:

$$\lambda_2 = \{U \mapsto A,\ V \mapsto A,\ Y \mapsto C,\ Z \mapsto C\}$$

Since λ_1 subsumes λ_2, λ_1 covers both micro-sequences.

Definition 12 (Subsumption) *Let λ_1 and λ_2 be two given mappings. We say that λ_1 subsumes λ_2, denoted as $\lambda_1 \sqsupseteq \lambda_2$, if and only if for every $(u \mapsto a) \in \lambda_2$ we have $(u \mapsto a) \in \lambda_1$.*

In addition to the concept of subsumption, there is the related concept of *compatibility*. This concept is illustrated by means of the following example.

Example 7 *Let UVWXXYZYZ be a micro-sequence which can be mapped to the macro-sequence ABC through the following mapping:*

$$\lambda_1 = \{U \mapsto A,\ V \mapsto A,\ W \mapsto B,\ X \mapsto B,\ Y \mapsto C,\ Z \mapsto C\}$$

Let UVST be a micro-sequence which can be mapped to the macro-sequence AD through the following mapping:

$$\lambda_2 = \{U \mapsto A,\ V \mapsto A,\ S \mapsto D,\ T \mapsto D\}$$

Then λ_1 does not subsume λ_2, but λ_1 is compatible with λ_2 in the sense that it is possible to create a third mapping $\lambda_3 = \{U \mapsto A,\ V \mapsto A,\ W \mapsto B,\ X \mapsto B,\ Y \mapsto C,\ Z \mapsto C,\ S \mapsto D,\ T \mapsto D\}$ which combines λ_1 and λ_2.

Definition 13 (Compatibility) *Let λ_1 and λ_2 be two given mappings, and let U_1 and U_2 be their domains, respectively. We say that λ_1 is compatible with λ_2 if and only if every event $u \in (U_1 \cap U_2)$ is mapped to the same activity a in both λ_1 and λ_2, i.e. $(u \mapsto a) \in \lambda_1$ and $(u \mapsto a) \in \lambda_2$. In particular, if $U_1 \cap U_2 = \varnothing$ then the two mappings are compatible.*

If two mappings are compatible, then they can be merged into a third mapping, as λ_3 in Example 7.

Definition 14 (Merge) *Let λ_1 and λ_2 be two compatible mappings. Then λ_1 and λ_2 can be merged into a third mapping $\lambda_3 = \{u \mapsto a \mid (u \mapsto a) \in \lambda_1 \vee (u \mapsto a) \in \lambda_2\}$. This merge operation will be denoted as $\lambda_3 \leftarrow \textsc{Merge}(\lambda_1, \lambda_2)$.*

2.4 Coverage of a mapping

With the table of mappings provided by Algorithm 2, it possible to find which micro-sequences are covered by a given mapping λ. These are the micro-sequences that can be mapped to some macro-sequence σ_j (it does not matter which one in particular) through mapping λ. In essence, this is just a matter of finding the tuples $(\mu_i, \sigma_j, \lambda_k)$ from T where λ_k is subsumed by λ, i.e. $\lambda_k \sqsubseteq \lambda$.

Algorithm 3 describes how the coverage of a given mapping λ is computed. The event log $L = [\mu_1^{q_1}, \mu_2^{q_2}, \mu_3^{q_3}, \ldots]$ and the macro-model $M = \{\sigma_1, \sigma_2, \ldots\}$ are assumed to be global variables, and table T is the result of Algorithm 2, as indicated in line 1. In line 3, the function initializes S_{cov}, which will contain the set of micro-sequences covered by λ. Similarly, in line 4 it initializes S_{all}, which will contain all micro-sequences (both covered and non-covered). The sets S_{cov} and S_{all} are built through lines 5–8.

In line 9, s_{cov} is a count of how many micro-sequences from the event log L are actually covered. Recall from Definition 10 that $\mu_i @ L$ is the support of μ_i in L, i.e. it is the number of times that μ_i appears in L. Line 10 does a similar count, but for all the micro-sequences in T. The coverage of λ is then the ratio of s_{cov} to s_{all}. This is usually expressed as a percentage value, as in line 11.

Algorithm 3 Compute the coverage for a given mapping λ

$T \leftarrow \text{ALLMAPPINGS}(L, M)$

function COVERAGE(λ)
  $S_{cov} \leftarrow \{\}$
  $S_{all} \leftarrow \{\}$
  for each $(\mu_i, \sigma_j, \lambda_k) \in T$ **do**
    if $\lambda_k \sqsubseteq \lambda$ **then**
      $S_{cov} \leftarrow S_{cov} \cup \{\mu_i\}$
    $S_{all} \leftarrow S_{all} \cup \{\mu_i\}$
  $s_{cov} \leftarrow \sum_{\mu_i \in S_{cov}} \mu_i @ L$
  $s_{all} \leftarrow \sum_{\mu_i \in S_{all}} \mu_i @ L$
  $p \leftarrow \frac{s_{cov}}{s_{all}} \times 100\%$
  return p

2.5 Coverage vs. range of a mapping

Ideally, it would be interesting to find the mapping λ with the highest possible coverage. This could be one of the mappings λ_k in T, or it could be a merge of several such mappings (like λ_3 in Example 7). If the coverage of λ would be 100%, then this would mean that the mapping λ is able to cover all the micro-sequences in the event log.

However, it turns out that coverage is not the only factor that should be considered. In fact, it is possible to get a mapping with 100% coverage, but this mapping may not be interesting at all, as the following example illustrates.

Example 8 *For the micro-sequence UVWXXYZYZ and the macro-sequence ABC, one possible mapping is:*

$$\lambda_1 = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$$

Now suppose that we observe the micro-sequence UV in the event log. There is no complete mapping between the micro-sequence UV and the macro-sequence ABC. However, λ_1 suggests that UV might have been produced by a macro-sequence with a single activity A. If we add this macro-sequence to the macro-model, then Algorithm 2 comes up with the following mapping:

$$\lambda_2 = \{U \mapsto A, V \mapsto A, W \mapsto A, X \mapsto A, Y \mapsto A, Z \mapsto A\}$$

which covers both UVWXXYZYZ and UV, and therefore has a coverage of 100% (assuming that there are no other micro-sequences in the event log).

The problem is that both UVWXXYZYZ and UV have been mapped to the macro-sequence A, when UVWXXYZYZ should have been mapped to ABC and UV should have been mapped to A. Fortunately, by trying all pairings of μ_i and σ_j, Algorithm 2 also comes up with λ_1 again:

$$\lambda_1 = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$$

which (now with macro-sequences ABC and A) covers both micro-sequences as well.

Clearly, λ_1 is preferable to λ_2. The reason for this is that the range of λ_1 includes activities A, B and C, whereas the range of λ_2 includes only activity A.

Definition 15 (Range) *The range of a mapping λ is the set of activities that appear in λ. It is defined as*

$$R(\lambda) = \{a \mid (u \mapsto a) \in \lambda\}$$

To compare two given mappings λ' and λ'' it is often useful to use the size of their ranges, i.e. $|R(\lambda')|$ and $|R(\lambda'')|$. Therefore, we define the function RANGE(λ) *as* RANGE(λ) $\leftarrow |R(\lambda)|$.

As in Example 8, we will be looking for mappings with the largest possible range (ideally, this range should contain all the activities in the macro-model M). Then, from all those mappings with the largest possible range, we pick the mapping (or mappings) with highest coverage. The next section explain the mining algorithm in more detail.

3 Mining approach

In general terms, the problem to be addressed here consists in finding a mapping between the low-level events in an event log $L = [\mu_1^{q_1}, \mu_2^{q_2}, \mu_3^{q_3}, \ldots]$ and the high-level activities in a macro-model $M = \{\sigma_1, \sigma_2, \ldots\}$. Such mapping should have the following desired characteristics:

1. Range – the range of the mapping should include, to the furthest extent possible, all the activities that appear in the macro-model M.
2. Coverage – the mapping should cover, to the furthest extent possible, all the micro-sequences that appear in the event log L.

The first approach that comes to mind would be to generate all complete mappings through Algorithm 2 and then pick the mapping (or mappings) that best fit the characteristics above. However, there are additional possibilities to be considered. In Example 7, we have seen that it is possible to generate new mappings by merging existing ones, and those new mappings should be considered as well.

Suppose that for a given micro-sequence μ_i and a given macro-sequence σ_j there are k complete mappings. Then if there are n micro-sequences and m macro-sequences, the total number of complete mappings is of the order of $n \times m \times k$. In addition, we have to consider merges of two mappings (for those pairs of mappings which are compatible), merges of three mappings (for those triples of mappings which are compatible), and so on.

For practical purposes, the set of mappings to be considered is simply too large to be searched exhaustively. Therefore, we use a greedy approach to search for a mapping $\hat{\lambda}$ that maximizes range and coverage. Such greedy approach is described in Algorithm 4. To make things clearer (even if at the expense of some rigor), this algorithm is described in plain words whenever possible.

In essence, the idea of Algorithm 4 is to keep merging compatible mappings into $\hat{\lambda}$ (which is initially empty). At each iteration, a new mapping λ'_k is merged into $\hat{\lambda}$ (step 6), so that all micro-sequences that are covered by λ'_k are now covered by $\hat{\lambda}$ as well. This mapping λ'_k is chosen according to three selection criteria:

- it must be compatible with $\hat{\lambda}$ (step 3);
- it must be the mapping (or one of the mappings) which, when merged with $\hat{\lambda}$, produces a new mapping with the largest possible range (step 4);

Algorithm 4 Find a mapping $\hat{\lambda}$ from a given event log $L = [\mu_1^{q_1}, \mu_2^{q_2}, \mu_3^{q_3}, \ldots]$ and macro-model $M = \{\sigma_1, \sigma_2, \ldots\}$

1. Initialize $\hat{\lambda} \leftarrow \{\}$ as an empty mapping.
2. Use Algorithm 2 to generate the table T of mappings for all micro-sequences in L and all macro-sequences in M. (If T is empty, then stop and return $\hat{\lambda}$.)
3. For each tuple $(\mu_i, \sigma_j, \lambda_k) \in T$ select those tuples where λ_k is compatible with $\hat{\lambda}$ (by Definition 13). Let T' be the set of tuples where λ_k is compatible with $\hat{\lambda}$. (If T' is empty, return $\hat{\lambda}$.)
4. From each mapping λ_k in T', try merging λ_k with $\hat{\lambda}$ (Definition 14) and compute the range of the resulting mapping (Definition 15). Select those tuples in T' where the resulting mapping has the largest range. Let T'' be the set of tuples where λ_k, if merged with $\hat{\lambda}$, leads to the largest possible range.
5. Use Algorithm 3 to compute the coverage of each mapping λ_k in T'' and pick the mapping λ'_k with the highest coverage. (In case there are several such mappings, pick one arbitrarily).
6. Merge λ'_k into $\hat{\lambda}$, i.e. $\hat{\lambda} \leftarrow \textsc{Merge}(\lambda'_k, \hat{\lambda})$ (Definition 14). The fact that λ'_k and $\hat{\lambda}$ are compatible and can be merged is ensured by step 3.
7. Let S_{cov} be the set of micro-sequences that are covered by λ'_k (S_{cov} is a by-product of Algorithm 3). Since the micro-sequences in S_{cov} have now been covered by λ'_k, removed them from the event log L.
8. As a result of step 7, L has been modified to contain just those micro-sequences which have not been covered up to this point. If L is empty, then return $\hat{\lambda}$. Otherwise, go back to step 2 with the modified L.

– it must be the mapping (or one of the mappings) with with highest coverage (step 5).

After running steps 2–8 iteratively, Algorithm 4 ends in one of three possible ways:

– either there are no mappings for the remaining (i.e. non-covered) micro-sequences in L (T is empty in step 2),
– or there are mappings, but none of them are compatible with $\hat{\lambda}$ (T' is empty in step 3),
– or there are no micro-sequences left to be covered in L (L is empty in step 8).

In the first two cases, there will be some micro-sequences left in L that cannot be covered by the mapping $\hat{\lambda}$ and therefore the coverage of $\hat{\lambda}$ will be lower than 100%. In the last case, $\hat{\lambda}$ has 100% coverage, and we can say that the macro-model M seems to be a perfect description for the behavior observed in the event log.

Example 9 *Let $L = [\mu_1^{q_1}, \mu_2^{q_2}]$ be an event log with the following micro-sequences:*

$$\mu_1 = \mathit{UVWXXYZYZ} \qquad q_1 = 5$$
$$\mu_2 = \mathit{UVYZYZ} \qquad q_2 = 3$$

Let $M = \{\sigma_1, \sigma_2\}$ be a macro-model with the macro-sequences $\sigma_1 = \mathit{ABC}$ and $\sigma_2 = \mathit{AC}$. Given L and M, Algorithm 4 runs as follows:

1. *$\hat{\lambda}$ is initialized as an empty mapping.*
2. *Algorithm 2 produces the same table as in Example 5 on page 9.*
3. *Since $\hat{\lambda}$ is empty, every mapping in T is compatible with $\hat{\lambda}$, so $T' = T$.*
4. *The mappings which yield the largest range are λ_1 through λ_6, and also λ_{11}.*
5. *The mappings with highest coverage are λ_1 through λ_6, because they cover μ_1 (with $q_1 = 5$), whereas λ_{11} covers μ_2 (with $q_2 = 3$). We can pick one of the mappings λ_1 through λ_6 arbitrarily, so let us pick, for example:*
   $\lambda_2 = \{\mathit{U} \mapsto \mathit{A}, \mathit{V} \mapsto \mathit{A}, \mathit{W} \mapsto \mathit{B}, \mathit{X} \mapsto \mathit{B}, \mathit{Y} \mapsto \mathit{C}, \mathit{Z} \mapsto \mathit{C}\}$.

6. *Since $\hat{\lambda}$ is empty, merging λ_2 into $\hat{\lambda}$ yields $\hat{\lambda} = \lambda_2$.*
7. *Since μ_1 has been covered, it is removed from L, which becomes $L = [\mu_2^{q_2}]$.*
8. *Since L is not empty, go back to step 2.*
2. *Table T is now the following:*

μ_i	σ_j	λ_k
$\mu_2 = UVYZYZ$	$\sigma_1 = ABC$	$\lambda_{11} = \{U \mapsto A, V \mapsto B, Y \mapsto C, Z \mapsto C\}$
$\mu_2 = UVYZYZ$	$\sigma_2 = AC$	$\lambda_{12} = \{U \mapsto A, V \mapsto A, Y \mapsto C, Z \mapsto C\}$
$\mu_2 = UVYZYZ$	$\sigma_2 = AC$	$\lambda_{13} = \{U \mapsto A, V \mapsto C, Y \mapsto C, Z \mapsto C\}$

3. *Only λ_{12} is compatible with $\hat{\lambda}$.*
4. *There is only λ_{12} to choose from.*
5. *There is only λ_{12} to choose from.*
6. *Merging λ_{12} into $\hat{\lambda}$ yields the same mapping $\hat{\lambda}$ as before, because $\hat{\lambda}$ subsumes λ_{12}.*
7. *Since μ_2 has been covered, it is removed from L, which becomes empty.*
8. *Since L is empty, $\hat{\lambda} = \{U \mapsto A, V \mapsto A, W \mapsto B, X \mapsto B, Y \mapsto C, Z \mapsto C\}$ is returned.*

In Algorithm 4 and in the example above there is an arbitrary choice of mapping λ'_k in step 5. In this particular example, any of the available choices would lead to a mapping $\hat{\lambda}$ with the same coverage, but in a general case this choice could have an impact in the final coverage of $\hat{\lambda}$. This is one of the problems of using a greedy approach: at each iteration we are choosing a mapping that appears to maximize coverage immediately, but it could happen that another mapping with equal coverage (or even lower coverage) would yield a higher coverage in the long run (i.e. in subsequent iterations).

On the other hand, it is possible that a mapping λ'_k which has not been chosen in the current iteration will end up being chosen in a subsequent iteration. (The only reason why this may not occur is if an incompatible mapping has been merged into $\hat{\lambda}$ in the meantime.) In any case, regardless of whether λ'_k or another mapping is chosen, the coverage of $\hat{\lambda}$ will increase, and it will keep increasing for as long as it is possible to find mappings that are compatible with (and therefore that can be merged into) $\hat{\lambda}$.

For this reason, Algorithm 4 is usually capable of finding a mapping with good coverage (i.e. higher than 50%). This is also helped by the fact that the distribution of sequences in an event log is typically unbalanced (with a few, very frequent sequences having a lot of occurrences and many sequences having just a few occurrences). By focusing on the mappings with highest coverage, Algorithm 4 can usually find a mapping that covers most of the event log. For those micro-sequences that are left to be covered, it is usually possible to come up with a suggestion, as described below.

3.1 Generating suggestions for improvement

As stated above, when Algorithm 4 gets to a point where it cannot cover the micro-sequences that are left in L, it stops and returns the current mapping $\hat{\lambda}$. However, in practice it is often the case that the mapping cannot cover the remaining micro-sequences not because the mapping itself is wrong, but because the macro-model does not include the behavior that could generate such micro-sequences. The following example illustrates this point.

Example 10 *Let $L = [\mu_1^{q_1}, \mu_2^{q_2}, \mu_3^{q_3}]$ be an event log with the following micro-sequences:*

$$\mu_1 = UVWXXYZYZ \qquad q_1 = 5$$
$$\mu_2 = UVWXUVYZ \qquad q_2 = 2$$

$$\mu_3 = \mathsf{UV} \qquad\qquad q_3 = 1$$

Let $M = \{\sigma_1\}$ be a macro-model with a single macro-sequence $\sigma_1 = \mathsf{ABC}$.

Given L and M, Algorithm 4 finds the mapping:

$$\hat{\lambda} = \{\mathsf{U} \mapsto \mathsf{A}, \mathsf{V} \mapsto \mathsf{A}, \mathsf{W} \mapsto \mathsf{B}, \mathsf{X} \mapsto \mathsf{B}, \mathsf{Y} \mapsto \mathsf{C}, \mathsf{Z} \mapsto \mathsf{C}\}$$

which covers μ_1. Therefore, $\hat{\lambda}$ covers $5/8 = 62.5\%$ of the micro-sequences in the event log.

The most frequent micro-sequence which is not covered is $\mu_2 = \mathsf{UVWXUVYZ}$. However, if we believe $\hat{\lambda}$ to be correct, then $\hat{\lambda}$ suggests that the micro-sequence $\mu_2 = \mathsf{UVWXUVYZ}$ was generated by a macro-sequence $\sigma_2 = \mathsf{ABAC}$. If we add this macro-sequence to the macro-model, then $\hat{\lambda}$ increases its coverage to $7/8 = 87.5\%$.

Now there is only one micro-sequence left to be covered: $\mu_3 = \mathsf{UV}$. If we keep to mapping $\hat{\lambda}$, then $\hat{\lambda}$ suggests that μ_3 is generated by the macro-sequence $\sigma_3 = \mathsf{A}$. By adding σ_3 to M, we achieve 100% coverage for $\hat{\lambda}$.

In this example, the macro-sequences $\sigma_2 = \mathsf{ABAC}$ and $\sigma_3 = \mathsf{A}$ can be regarded as suggestions for improving the macro-model in order to increase the coverage of mapping $\hat{\lambda}$. Of course, this mapping could be wrong from the perspective of a business analyst, and in this case these suggestions should not be followed. In any case, they are just suggestions; if the analyst believes that the mapping $\hat{\lambda}$ makes sense, then the additional macro-sequences can be incorporated in the macro-model, resulting in a new, improved version.

A suggestion can be generated by applying the mapping $\hat{\lambda}$ to a given non-covered micro-sequence μ_i. In the example above, $\hat{\lambda}$ has been applied to μ_2 and to μ_3. When $\hat{\lambda}$ is applied to $\mu_2 = \mathsf{UVWXUVYZ}$ the result is $\sigma_2^\star = \mathsf{AABBAACC}$. By eliminating all consecutive repetitions of activities, we get $\sigma_2 = \mathsf{ABAC}$. Similarly, when $\hat{\lambda}$ is applied to $\mu_3 = \mathsf{UV}$ the result is $\sigma_3^\star = \mathsf{AA}$ and $\sigma_3 = \mathsf{A}$. The following definitions establish the notation for these operations.

Definition 16 (Apply) *A mapping $\lambda = \{u_i \mapsto f(u_i)\}$ can be applied to a micro-sequence $\mu = \langle u_1, u_2, \ldots, u_n \rangle$, resulting in a new sequence $\sigma^\star = \langle f(u_1), f(u_2), \ldots, f(u_n) \rangle$. This operation is denoted as $\sigma^\star \leftarrow \textsc{Apply}(\lambda, \mu)$.*

Definition 17 (Collapse) *A sequence of symbols:*

$$\sigma^\star = \langle \underbrace{a_1, a_1, ..., a_1}_{a_1}, \underbrace{a_2, a_2, ..., a_2}_{a_2}, \quad \ldots \quad , \underbrace{a_n, a_n, ..., a_n}_{a_n} \rangle$$

can be collapsed into:

$$\sigma = \langle a_1, a_2, \ldots, a_n \rangle$$

by eliminating all consecutive repetitions of the same symbol in $\sigma^\star$. This operation is denoted as $\sigma \leftarrow \textsc{Collapse}(\sigma^\star)$.

To generate a suggested macro-sequence σ_j for a non-covered micro-sequence μ_i, first we apply the mapping $\hat{\lambda}$ to the micro-sequence μ_i in order to generate the sequence $\sigma_j^\star$, i.e. $\sigma_j^\star \leftarrow \textsc{Apply}(\hat{\lambda}, \mu_i)$. Then we eliminate all consecutive repetitions of the same symbol in $\sigma_j^\star$ in order to obtain the suggested macro-sequence σ_j, i.e. $\sigma_j \leftarrow \textsc{Collapse}(\sigma_j^\star)$.

Example 11 *Given the micro-sequence $\mu_4 = \mathsf{UVYZYZ}$ and the mapping $\hat{\lambda} = \{\mathsf{U} \mapsto \mathsf{A}, \mathsf{V} \mapsto \mathsf{A}, \mathsf{W} \mapsto \mathsf{B}, \mathsf{X} \mapsto \mathsf{B}, \mathsf{Y} \mapsto \mathsf{C}, \mathsf{Z} \mapsto \mathsf{C}\}$ from Example 10, it is possible to generate a suggested macro-sequence in the form $\sigma_4^\star = \mathsf{AACCCC}$ and $\sigma_4 = \mathsf{AC}$.*

3.2 Visualizing the (new) process model

In Example 10 we started with a single macro-sequence $\sigma_1 = \mathsf{ABC}$ and then we added $\sigma_2 = \mathsf{ABAC}$ and $\sigma_3 = \mathsf{A}$ to the macro-model. How has the process model changed when these new macro-sequences were added? Clearly, the point of adding σ_2 and σ_3 was to be able to map μ_1 to $\sigma_1 = \mathsf{ABC}$, μ_2 to $\sigma_2 = \mathsf{ABAC}$, and μ_3 to $\sigma_3 = \mathsf{A}$. However, once we added σ_2 and σ_3 to the macro-model, the process itself is no longer a simple sequence of activities ABC.

In Example 10, there are 5 instances of μ_1, 2 instances of μ_2, and 1 instance of μ_3. In total, there are 8 instances of this process. Since μ_1, μ_2 and μ_3 are being mapped to σ_1, σ_2 and σ_3 respectively, it is as if the process has been executed 8 times in the following way: 5 instances in the form $\sigma_1 = \mathsf{ABC}$; 2 instances in the form $\sigma_2 = \mathsf{ABAC}$; and 1 instance in the form $\sigma_3 = \mathsf{A}$. With this information, we can calculate the transition probabilities between the activities in the process in order to visualize the process model as a Markov chain.

Of course, another kind of model could be used to visualize the process, such as a dependency graph [30] or a Petri net [27]. Our goal here is just to illustrate the process (and its changes from one version to another) in an intuitive way, and for that purpose we use the simplest possible framework. In particular, the type of Markov chain that we use here, which is extended with special "open" and "closed" states, is somewhat similar to the concept of WF-net, as defined in [27], which has special "input" and "output" places.

Example 12 *Let $\circ$ ("open") and $\bullet$ ("closed") be two special symbols to denote the beginning and the end of the process, respectively. With $\sigma_1 = \circ\mathsf{ABC}\bullet$, $\sigma_2 = \circ\mathsf{ABAC}\bullet$, and $\sigma_3 = \circ\mathsf{A}\bullet$, we can say that the process always begins with activity A. We write $P(\circ, \mathsf{A}) = 1.0$ to say that the transition probability between $\circ$ and A is 1.0.*

With regard to the transition probabilities from A to other activities, first we note that there is one A in σ_1, two A's in σ_2, and one A in σ_3. Since there are 5 instances of σ_1, 2 instances of σ_2, and 1 instance of σ_3, we get a total number of $5\times1 + 2\times2 + 1\times1 = 10$ occurrences of A. The transition probabilities are as follows:

- *in $5 \times 1 + 2 \times 1 = 7$ of those occurrences, the process goes from A to B, therefore $P(\mathsf{A}, \mathsf{B}) = 7/10$;*
- *in $2 \times 1 = 2$ of those occurrences, the process goes from A to C, therefore $P(\mathsf{A}, \mathsf{C}) = 2/10$;*
- *in $1 \times 1 = 1$ of those occurrences, the process ends with A, therefore $P(\mathsf{A}, \bullet) = 1/10$.*

With regard to the transition probabilities from B to other activities, we note that there is one B in σ_1, one B in σ_2, and none in σ_3. Therefore, there is a total number of $5\times1 + 2\times 1 + 1\times0 = 7$ occurrences of B. The transition probabilities are:

- *in $5\times1 = 5$ of those occurrences, the process goes from B to C, therefore $P(\mathsf{B}, \mathsf{C}) = 5/7$;*
- *in $2\times1 = 2$ of those occurrences, the process goes from B to A, therefore $P(\mathsf{B}, \mathsf{A}) = 2/7$.*

With regard to the transition probabilities from C to other activities, we note that there is one C in σ_1 and one C in σ_2. Therefore, there is a total number of $5\times1+2\times1 = 7$ occurrences of C. In all of those occurrences, the process ends with C. Therefore, $P(\mathsf{C}, \bullet) = 1.0$.

All other transitions probabilities are zero. Therefore, the transition matrix for this process is:

$$\mathcal{M} = \begin{array}{c|ccccc} & \circ & A & B & C & \bullet \\ \hline \circ & 0 & 1 & 0 & 0 & 0 \\ A & 0 & 0 & \frac{7}{10} & \frac{2}{10} & \frac{1}{10} \\ B & 0 & \frac{2}{7} & 0 & \frac{5}{7} & 0 \\ C & 0 & 0 & 0 & 0 & 1 \\ \bullet & 0 & 0 & 0 & 0 & 0 \end{array}$$

This transition matrix can be depicted as a graphical model, as shown in Figure 2.

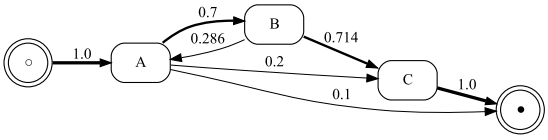


Fig. 2: A graphical depiction of the transition matrix for a macro-model with three macro-sequences

As this example illustrates, calculating the transition matrix for the process is a matter of counting the transitions of one activity to another, and dividing by the total number of occurrences of the first activity. The procedure is relatively simple, as is the conversion of the transition matrix to the type of graphical model shown in Figure 2. Therefore, we do not provide a formal description of the procedure that has just been illustrated in Example 12.

However, it is important to mention how the required data for computing the transition matrix can be obtained. For each micro-sequence $\hat{\mu}$ that is covered by $\hat{\lambda}$, we can lookup the corresponding macro-sequence $\hat{\sigma}$ in table T by selecting the tuple $(\mu_i, \sigma_j, \lambda_k)$ where $\mu_i = \hat{\mu}$ and $\lambda_k \sqsubseteq \hat{\lambda}$. As for the number of instances that should be assigned to σ_j, this is equal to the sum of the support $(\mu_i @ L)$ for every micro-sequence μ_i that ends up being mapped to σ_j.

4 Case study: Volvo IT Belgium

This case study is based on a real-world event log collected from an incident management system at Volvo IT Belgium. Originally, this event log was made publicly available for the *BPI Challenge 2013*.[2] In that challenge, the goal was to analyze the event log in order to answer a number of specific questions posed by the process owner. Here we focus on arriving at a high-level process model that captures the low-level behavior recorded in the event log.

For the purpose of *BPI Challenge 2013*, there were actually three event logs available for two different processes: incident management, and problem management. Here, we illustrate the application of our mining approach to the incident management process, which is the one that is described in more detail in existing documentation.

[2] http://www.win.tue.nl/bpi/2013/challenge

SR number	Status	Sub-status	Involved ST	timestamp
1-364285768	Accepted	In Progress	V30	2010-03-31 15:59:42
1-364285768	Accepted	In Progress	V30	2010-03-31 16:00:56
1-364285768	Queued	Awaiting Assignment	V5 3rd	2010-03-31 16:45:48
1-364285768	Accepted	In Progress	V5 3rd	2010-04-06 15:44:07
1-364285768	Queued	Awaiting Assignment	V30	2010-04-06 15:44:38
1-364285768	Accepted	In Progress	V13 2nd 3rd	2010-04-06 15:44:47
1-364285768	Completed	Resolved	V13 2nd 3rd	2010-04-06 15:44:51
...	...	...	...	...
1-467153946	Completed	Closed	S42	2012-05-23 00:22:25

Table 1: An excerpt of the event log from Volvo IT Belgium

4.1 The event log

In the event log[3], we find the events that correspond to all changes in the status of each incident (an incident is also referred to as a *service request*). Each event has:

- an *SR number*, which is a unique identifier for the service request;
- a *Status*, which can be *Accepted*, *Queued* or *Completed*;
- a *Sub-status*, which is one of *In Progress*, *Awaiting Assignment*, *Assigned*, *Resolved*, *Closed*, etc.;
- the *Involved ST*, which is the support team working on the incident and which changed its status or sub-status;
- a *timestamp* for the event, in the form of a date and time;
- other fields, such as the impact of the incident (high, medium, low), the product that the incident refers to, and the country that the support team belongs to.

An excerpt of the event log with the fields *SR number*, *Status*, *Sub-status*, *Involved ST* and *timestamp* is shown in Table 1. Here it is possible to see that the first event in the log is from March 2010 and the last event is from May 2012. There are 7554 cases (i.e. incidents) recorded in the event log, and a total of 65533 events. From these 7554 cases, there are 2278 different traces (i.e. micro-sequences).

The *Involved ST* field contains the identifier of the support team, sometimes with a suffix "2nd" or "3rd". This means that the support team belongs to the second line and/or third line, rather than to the first line. The first line is the service helpdesk that serves as a frontend to the user or customer who submitted the service request. The second and third lines refer to support teams that take care of those incidents which cannot be solved by the first line. In particular, the third line is comprised of experts in product development which should only be involved if the incident cannot be solved within the first or second line.

The act of passing work from the first line to the second or even to the third line is known as *escalation* in the popular ITIL framework [22]. One of the issues to be analyzed in the *BPI Challenge 2013* was whether escalation of service requests takes place too often or too soon. Here we will be focusing on the *Status* and *Sub-status* columns, in order to map these statuses to a set of high-level activities.

[3] The event log has the following DOI reference: http://dx.doi.org/10.4121/uuid:500573e6-accc-4b0c-9576-aa5468b10cee

4.2 The process model

An interesting feature of *BPI Challenge 2013* is that the event log was made available together with some documentation about the process of handling service requests. A BPMN model for this process is shown in Figure 3.

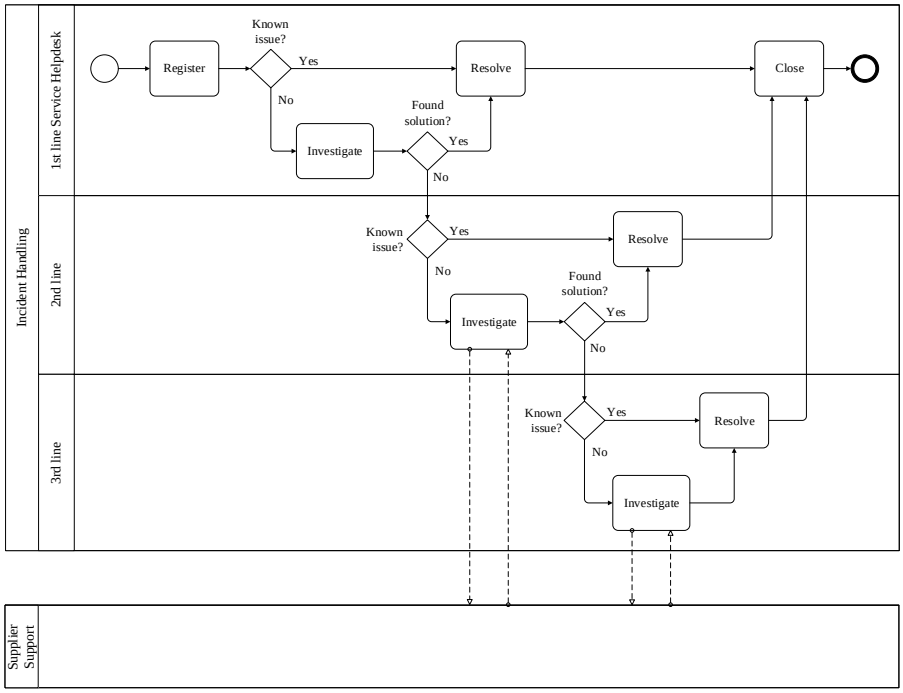


Fig. 3: A BPMN model of the high-level process associated with handling service requests

The model shows how an incident that is being "investigated" within the first line can escalate to the second line, and possibly to the third line. This happens in a similar way across lines: if no solution can be found within the current line, the incident escalates to the next line, where it will be matched against a database of known issues; if the incident is recognized as a known issue, then an existing solution is applied to resolve it; otherwise, the support team will try to come up with a solution to the problem. Eventually, if the problem is traced back to a certain component, the support team may get in contact with the respective supplier in order to fix the problem.

Our goal is to map the low-level events in Table 1 to the high-level activities in Figure 3. For this purpose, we define a low-level event as being the concatenation of the *Status* and *Sub-status* fields in Table 1. For example, if the status is *Accepted* and the sub-status is *In Progress*, we concatenate these fields into the string AcceptedInProgress. Then the mapping we are looking for will contain pairs of event and activity, such as:

$$\mathsf{AcceptedInProgress} \mapsto \mathsf{Investigate}$$

In fact, in the event log we find that about 84.4% of cases begin with AcceptedInProgress, 15.3% begin with QueuedAwaitingAssignment, and the remaining 0.3% begin with other events. The fact that such a large percentage of cases begin with AcceptedInProgress appears to suggest that this event should be mapped to the first activity in the process, i.e. Register. However, Table 1 shows that the event AcceptedInProgress can appear multiple times in each trace, so it cannot belong to the activity Register.

See for example the escalation from first line (V30) directly(!) to third line (V5 3rd) that takes place from the second to the third event in Table 1. According to Figure 3, there is no Register activity in the third line, so AcceptedInProgress cannot be mapped to Register. Therefore, we conclude that the observable behavior recorded in the event log begins with the activity Investigate.

Before proceeding with the analysis, we should mention that no form of filtering or preprocessing was applied to the event log; we simply take the micro-sequences as they are, with duplicate events included. This is important to mention because, in practice, it is quite difficult to analyze real-world event logs without some sort of preprocessing [16]. However, with the present approach we are able to analyze the event log in its entirety, without worrying about *noise* [24].

4.3 Analysis

We start with the event log and a macro-model with a single macro-sequence:

$$\sigma_1 = \langle \mathsf{Investigate}, \mathsf{Resolve}, \mathsf{Close} \rangle$$

On a Linux laptop with an Intel Core i5-4200U processor, an implementation of Algorithm 4 in Python 2.7 takes about 1.26 seconds to run, and produces the following mapping with 68.7% coverage:

$$\begin{aligned}
\mathsf{AcceptedAssigned} &\mapsto \mathsf{Investigate} \\
\mathsf{AcceptedInProgress} &\mapsto \mathsf{Investigate} \\
\mathsf{AcceptedWait} &\mapsto \mathsf{Investigate} \\
\mathsf{AcceptedWaitCustomer} &\mapsto \mathsf{Investigate} \\
\mathsf{AcceptedWaitImplementation} &\mapsto \mathsf{Investigate} \\
\mathsf{AcceptedWaitUser} &\mapsto \mathsf{Investigate} \\
\mathsf{AcceptedWaitVendor} &\mapsto \mathsf{Investigate} \\
\mathsf{CompletedClosed} &\mapsto \mathsf{Close} \\
\mathsf{CompletedInCall} &\mapsto \mathsf{Investigate} \\
\mathsf{CompletedResolved} &\mapsto \mathsf{Resolve} \\
\mathsf{QueuedAwaitingAssignment} &\mapsto \mathsf{Investigate}
\end{aligned}$$

Since at this stage there is only one macro-sequence, all covered micro-sequences are mapped to σ_1. The current process model is shown in Figure 4.

The most frequent micro-sequence that cannot be covered is: $\mu_1 = \langle$AcceptedInProgress, AcceptedInProgress, CompletedInCall$\rangle$. This corresponds to those incidents which end up being resolved during a call with the customer. The mapping above suggests that this micro-sequence μ_1 might have been generated by a macro-sequence in the form:

Fig. 4: Process model, version 1

$$\sigma_2 = \langle \mathsf{Investigate} \rangle$$

Therefore, the suggestion is to add this macro-sequence to the macro-model. If we accept this suggestion and run Algorithm 4 again, it takes 2.59 seconds to generate a mapping with 93.7% coverage. The new mapping is equal to the previous one, with the addition of the following pair:

$$\mathsf{CompletedCancelled} \mapsto \mathsf{Investigate}$$

This means that, provided with σ_1 and σ_2, Algorithm 4 was able to cover not only the micro-sequence μ_1 but also other traces containing the status CompletedCancelled.

Based on the macro-sequences which the covered micro-sequences have been mapped to, a new version of the process model is shown in Figure 5.

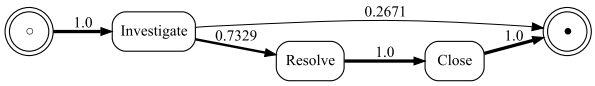


Fig. 5: Process model, version 2

The most frequent micro-sequence that cannot be covered is now: $\mu_2 = \langle$AcceptedInProgress, AcceptedInProgress, CompletedResolved, AcceptedInProgress, CompletedResolved, CompletedClosed$\rangle$. This corresponds to those incidents which were thought to have been resolved on a first attempt, but were resolved only after a second attempt. The current mapping suggests that this micro-sequence μ_2 might have been generated by a macro-sequence in the form:

$$\sigma_3 = \langle \mathsf{Investigate}, \mathsf{Resolve}, \mathsf{Investigate}, \mathsf{Resolve}, \mathsf{Close} \rangle$$

After inserting this suggestion in the macro-model and running Algorithm 4 again, it takes 9.66 seconds to generate a mapping with 96.9% coverage. The new mapping adds the following pair:

$$\mathsf{UnmatchedUnmatched} \mapsto \mathsf{Resolve}$$

This means that, provided with σ_1, σ_2 and σ_3, Algorithm 4 was able to cover also those micro-sequences with the (undocumented) status UnmatchedUnmatched.

Based on the macro-sequences which the covered micro-sequences have been mapped to, a third version of the process model is shown in Figure 6.

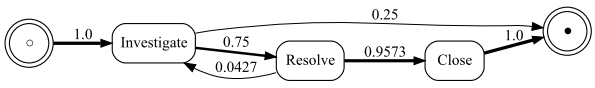


Fig. 6: Process model, version 3

The most frequent micro-sequence that is not covered is now: $\mu_3 = \langle$AcceptedInProgress, AcceptedInProgress, AcceptedWaitUser, CompletedResolved$\rangle$. The current mapping suggests that this micro-sequence μ_3 might have been generated by a macro-sequence in the form:

$$\sigma_4 = \langle \text{Investigate}, \text{Resolve} \rangle$$

We add this micro-sequence to the macro-model and run Algorithm 4 again, which takes 18.05 seconds to find out that the current mapping, without any further change, is now able to cover 98.0% of the traces in the event log. However, because we added σ_4, there has been a slight change in the process model, as shown in Figure 7.

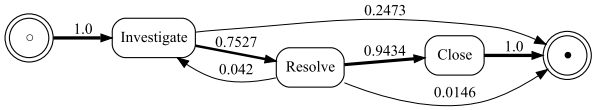


Fig. 7: Process model, version 4

By carrying out the same procedure repeatedly – i.e. by adding the suggested macro-sequences and running Algorithm 4 again – the mapping does not change anymore, but the new macro-sequences keep refining the process model, until eventually all the micro-sequences in the event log are covered. Figures 8–19 show the successive versions of the process model, as each new macro-sequence is added.

Starting with version 6 in Figure 9 it can be seen that the coverage is already very close to 100%. From that point onwards, the new macro-sequences that are added to the macro-model are intended to cover some very particular cases which represent a very small fraction of the event log (of the order of 0.1% or even less).

For example, from version 8 (Figure 11) to version 9 (Figure 12) the macro-sequence $\sigma_9 = \langle$Resolve, Investigate, Resolve$\rangle$ was added. This is an interesting point in the analysis because σ_9 is the first macro-sequence to be added that begins with an activity other than Investigate. However, as can be seen in the model of Figure 12, only 0.03% of cases are actually covered by that macro-sequence.

Eventually, by adding the suggested macro-sequences, the mapping is able to cover all the cases in the event log. The resulting process model is shown in Figure 19. Even if the analyst does not follow the suggestions and prefers to add different macro-sequences at each step, it will still be possible to achieve a macro-model with 100% coverage, albeit a different one from that of Figure 19.

4.4 Execution time and coverage

Figure 20 plots the execution time and coverage achieved by Algorithm 4 as each new macro-sequence is added to the macro-model. These plots illustrate two important facts:

- Even though Algorithm 2 computes all possible mappings between each micro-sequence μ_i and each macro-sequence σ_j, the total execution time of Algorithm 4 does not appear to rise exponentially with the number of macro-sequences. This provides confidence that, in practice, it will be possible to add an arbitrary number of macro-sequences and still carry out the analysis within a reasonable amount of time.

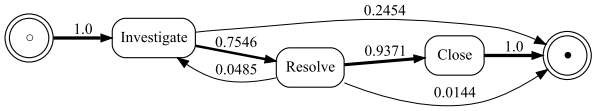


Fig. 8: Process model version 5, after adding $\sigma_5 = \langle$Investigate, Resolve, Investigate, Resolve, Investigate, Resolve, Close$\rangle$, coverage 98.29%

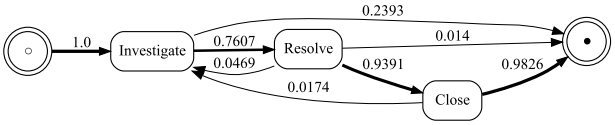


Fig. 9: Process model version 6, after adding $\sigma_6 = \langle$Investigate, Resolve, Close, Investigate, Resolve, Close$\rangle$, coverage 99.59%

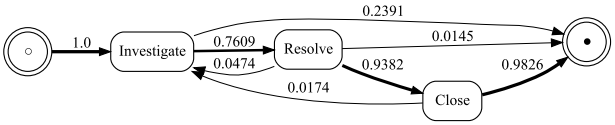


Fig. 10: Process model version 7, after adding $\sigma_7 = \langle$Investigate, Resolve, Investigate, Resolve$\rangle$, coverage 99.63%

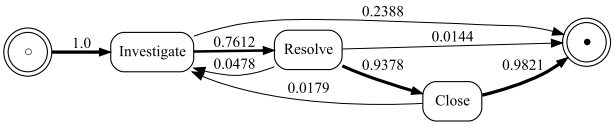


Fig. 11: Process model version 8, after adding $\sigma_8 = \langle$Investigate, Resolve, Close, Investigate, Resolve, Investigate, Resolve, Close$\rangle$, coverage 99.67%

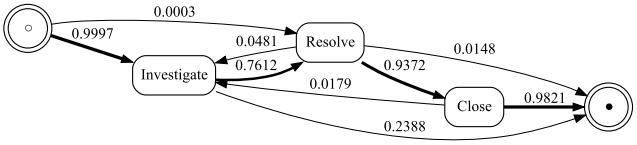


Fig. 12: Process model version 9, after adding $\sigma_9 = \langle$Resolve, Investigate, Resolve$\rangle$, coverage 99.69%

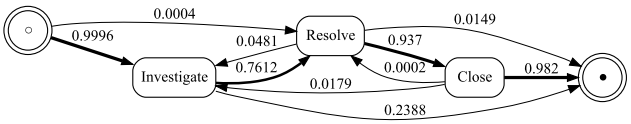


Fig. 13: Process model version 10, after adding $\sigma_{10} = \langle$Resolve, Close, Resolve$\rangle$, coverage 99.71%

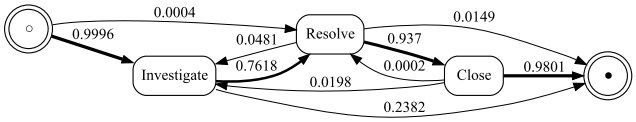


Fig. 14: Process model version 11, after adding $\sigma_{11} = \langle$Investigate, Resolve, Investigate, Resolve, Close, Investigate, Resolve, Close$\rangle$, coverage 99.79%

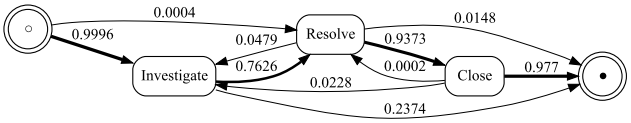


Fig. 15: Process model version 12, after adding $\sigma_{12} = \langle$Investigate, Resolve, Close, Investigate, Resolve, Close, Investigate, Resolve, Close$\rangle$, coverage 99.91%

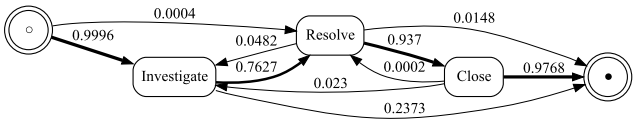


Fig. 16: Process model version 13, after adding $\sigma_{13} = \langle$Investigate, Resolve, Investigate, Resolve, Investigate, Resolve, Close, Investigate, Resolve, Close$\rangle$, coverage 99.92%

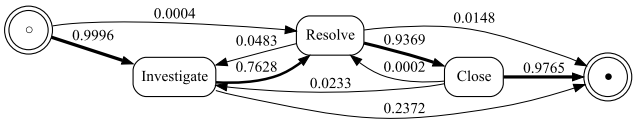


Fig. 17: Process model version 14, after adding $\sigma_{14} = \langle$Investigate, Resolve, Close, Investigate, Resolve, Investigate, Resolve, Close, Investigate, Resolve, Close$\rangle$, coverage 99.93%

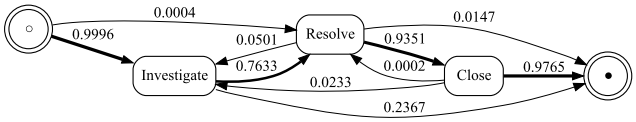


Fig. 18: Process model version 15, after adding $\sigma_{15} = \langle$Investigate, Resolve, Investigate, Resolve, Investigate, Resolve, Investigate, Resolve, Close$\rangle$, coverage 99.99%

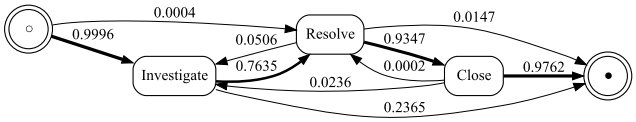
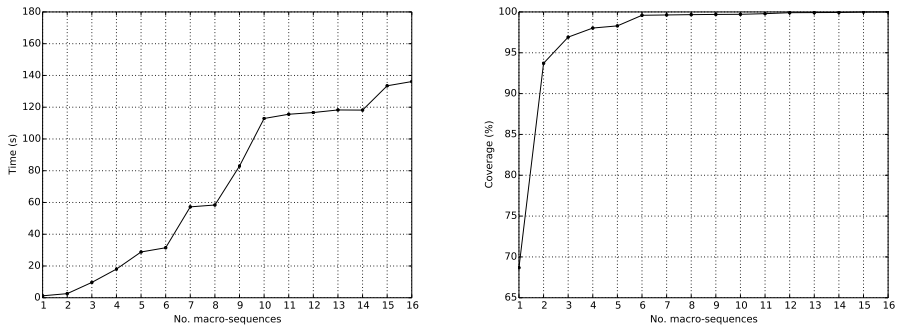


Fig. 19: Process model version 16, after adding $\sigma_{16} = \langle$Investigate, Resolve, Investigate, Resolve, Close, Investigate, Resolve, Investigate, Resolve, Investigate, Resolve, Close, Investigate, Resolve, Close$\rangle$, coverage 100.00%



(a) Time (sec.) (b) Coverage (%)

Fig. 20: Time and coverage vs. number of macro-sequences in the macro-model

– Even though it is possible to achieve 100% coverage by adding all of the suggested macro-sequences, for practical purposes it will be hardly necessary to do so since, at a certain point, the gain in coverage does not justify the extra execution time. By the time the mapping covers 99% of the event log (or another percentage that the analyst finds appropriate), the remaining traces can be dismissed as noise, and the current process model can be considered to be a sufficiently accurate description for the behavior observed in the event log.

5 Related work

The gap between high-level activities and low-level events is a well-known problem in the field of process mining [1, 10, 11, 12]. Despite the development of a wide range of process mining techniques, most of these techniques are able to discover process models that are at the same level of abstraction as the events recorded in the event log (i.e. when there is a one-to-one mapping between events and activities, and this mapping is known a priori). However, end users need solutions to analyze event data and to visualize the results at a higher level of abstraction, preferably at the same level of abstraction of their process models.

The research community has been looking into this problem and, while it is still a topic of ongoing research, some approaches have already been proposed to be able to produce more abstract models from an event log. These approaches can be divided into two main groups:

– First, there are techniques that work on the basis of models, by extracting a low-level model from the event log and then creating more abstract representations of that model. Examples are [10] and [11]. Basically, these techniques work by aggregating nodes in the model, in order to produce a simplified and more abstract picture of the process. In general, these approaches allow a stepwise simplification of the process until, in the limit, everything is aggregated into a single node. It is the end user who must know how far to carry the simplification in order to obtain meaningful results. A disadvantage of these approaches is that it is not possible to automatically identify aggregated nodes

as meaningful business activities (they are simply labeled as “Cluster A”, “Cluster B”, etc.), so it may be difficult for the end user to understand and analyze the results.

– Second, there are techniques that work on the basis of the events, by translating the event log into a more abstract sequence of events and then producing a model from that translated event log. Examples are [12] and [1]. Basically, these techniques work by identifying frequent patterns of events in the event log, and then substituting each of these patterns by a single, higher-level event. After all substitutions have been made, the event log becomes a sequence of more abstract events. As a final step, it is possible to extract a model by usual techniques, such as the α-algorithm [25], the heuristics miner [30], or the genetic miner [6]. As with other approaches, it is possible to perform this abstraction in multiple stages, but there is no guarantee that the patterns of events that are identified in the event log correspond to meaningful business activities, so it is up to the end user to determine whether such correspondence actually exists.

The problem with these approaches is that, although they are able to create new layers of abstraction over an event log, in general they do not take into account that there may already exist an abstract notion of the business process in the form of a process model, or in another form (e.g. a flowchart, or a procedure manual) that can be translated into a process model. This information can become a valuable input for the analysis, since it identifies the high-level activities that one should use when building abstractions over an event log. Our approach differs by taking this information into account.

On the other hand, there has been a surging interest in techniques that are based on aligning the events in an event log with the activities in a process model [2, 3, 4, 5, 28]. The concept of mapping that we use here can be somewhat related to that kind of alignment, in the sense that an alignment also establishes a correspondence between events and activities. Also, there is a search space for possible alignments that is somewhat analogous to the search space of possible mappings. However, in those alignments the process model is at the same level of abstraction of the event log, whereas in the present work we use the concept of mapping to bridge the gap between those two different levels. A further difference is that we focus on model improvement, whereas the works cited above focus either on discovery [1, 3, 10, 11, 12] or conformance [2, 4, 5, 28].

Also, another branch of work that can be related to the present approach is that of generating recommendations in the context of process modeling [13, 14, 15, 20, 29]. Of special interest is [20], where the recommendations are based on the past history of the process, which is recorded in an event log. However, once again there is a one-to-one mapping between the events in the log and the activities in the process. In contrast, our suggestions for new macro-sequences are based on a mapping which must be mined from the event log and from a higher-level model of the process. This model may be inaccurate at first, but through the addition of the suggested macro-sequences, or other macro-sequences that the analyst finds appropriate, the model is able to cover more and more of the event log.

6 Conclusion

In this work we have shown that it is possible to enhance a given process model with information about the run-time behavior of the process, as recorded in an event log. More importantly, we have shown that it is possible to do this even when the relationship between the activities in the process model and the events in the log is unknown.

Assuming that each event can be mapped to a single activity, we introduced the concept of mapping and we showed how to mine the set of possible mappings from a given micro-

sequence and macro-sequence (Algorithm 1). Then we generalized this procedure in order to be able to find the set of possible mappings between all the micro-sequences in the event log and all the macro-sequences in the process model (Algorithm 2).

In general, the set of possible mappings can be large, and one should focus on the mapping which has the highest coverage, meaning that the chosen mapping should be applicable to the largest possible number of traces in the event log (Algorithm 3). However, coverage is not the only factor to take into account, since a mapping where all events are mapped to the same activity will have 100% coverage but will be of no practical interest.

The range of a mapping, i.e. the set of activities that appear in the mapping, is an even more important factor. Therefore, first we select the mappings with largest range, and then we select the mappings with highest coverage from those with largest range. If possible, we combine (i.e. merge) mappings in order to create a new mappings with even larger range and higher coverage. There may be many such possible combinations, so Algorithm 4 adopts a greedy approach, where it tries to merge those mappings which, by themselves, already appear to have the largest range and highest coverage among their peers.

In practice, such greedy approach is enough to find a mapping that covers a large fraction of the event log. For those traces which are not covered by the mapping, it is possible to generate a suggestion in the form of a macro-sequence which, if added to the process model, will allow the mapping to increase its coverage. These suggestions, or whatever macro-sequences the analyst decides to add, are the basis for enhancing the process model.

In a case study involving a real-world event log from the BPI Challenge 2013, we have shown how the successive inclusion of the suggested macro-sequences has contributed to reach a model that captures the run-time behavior of the process in terms of the high-level activities used to document that process. We have also highlighted the fact that the approach scales reasonably well with the increase in the number of macro-sequences, and that it is able to deal with noise, by eventually covering all traces in the event log.

References

1. R. P. Jagadeesh Chandra Bose, E. H. M. W. Verbeek, and Wil M. P. van der Aalst. Discovering hierarchical process models using ProM. In *CAiSE Forum 2011*, volume 107 of *LNBIP*, pages 33–48. Springer, 2012.
2. Massimiliano de Leoni and Wil M. P. van der Aalst. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In *Business Process Management*, volume 8094 of *LNCS*, pages 113–129. Springer, 2013.
3. Massimiliano de Leoni and Wil M. P. van der Aalst. Data-aware process mining: discovering decisions in processes using alignments. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1454–1461. ACM, 2013.
4. Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. Aligning event logs and declarative process models for conformance checking. In *Business Process Management*, volume 7481 of *LNCS*, pages 82–97. Springer, 2012.
5. Massimiliano de Leoni, Wil M. P. van der Aalst, and Boudewijn F. van Dongen. Data- and resource-aware conformance checking of business processes. In *Business Information Systems*, volume 117 of *LNBIP*, pages 48–59. Springer, 2012.
6. A. K. Alves de Medeiros and A. J. M. M. Weijters. Genetic process mining. In *Applications and Theory of Petri Nets 2005*, volume 3536 of *LNCS*, pages 48–69. Springer, 2005.

7. A. K. Alves de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.
8. Diogo R. Ferreira, Fernando Szimanski, and Célia Ghedini Ralha. A hierarchical Markov model to understand the behaviour of agents in business processes. In *Business Process Management Workshops*, volume 132 of *LNBIP*, pages 150–161. Springer, 2013.
9. Diogo R. Ferreira, Fernando Szimanski, and Célia Ghedini Ralha. Mining the low-level behavior of agents in high-level business processes. *International Journal of Business Process Integration and Management*, 6(2):146–166, 2013.
10. Gianluigi Greco, Antonella Guzzo, and Luigi Pontieri. Mining hierarchies of models: From abstract views to concrete specifications. In *3rd International Conference on Business Process Management*, volume 3649 of *LNCS*, pages 32–47. Springer, 2005.
11. Christian W. Günther and Wil M. P. van der Aalst. Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In *5th International Conference on Business Process Management*, volume 4714 of *LNCS*, pages 328–343. Springer, 2007.
12. Christian W. Günther, Anne Rozinat, and Wil M. P. van der Aalst. Activity mining by global trace segmentation. In *BPM 2009 International Workshops*, volume 43 of *LNBIP*, pages 128–139. Springer, 2010.
13. Thomas Hornung, Agnes Koschmider, and Georg Lausen. Recommendation based process modeling support: Method and user experience. In *Conceptual Modeling - ER 2008*, volume 5321 of *LNCS*, pages 265–278. Springer, 2008.
14. Agnes Koschmider, Minseok Song, and Hajo A. Reijers. Advanced social features in a recommendation system for process modeling. In *Business Information Systems*, volume 21 of *LNBIP*, pages 109–120. Springer, 2009.
15. Agnes Koschmider, Thomas Hornung, and Andreas Oberweis. Recommendation-based editor for business process modeling. *Data & Knowledge Engineering*, 70(6):483–503, June 2011.
16. R.S. Mans, M.H. Schonenberg, M. Song, W.M.P. Aalst, and P.J.M. Bakker. Application of process mining in healthcare - a case study in a dutch hospital. In Ana Fred, Joaquim Filipe, and Hugo Gamboa, editors, *Biomedical Engineering Systems and Technologies*, volume 25 of *CCIS*, pages 425–438. Springer, 2009.
17. *Business Process Model and Notation (BPMN), Version 2.0.* OMG, 2011.
18. A. Rozinat and W.M.P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.
19. August-Wilhelm Scheer. *ARIS: Business Process Modeling*. Springer, 3rd edition, 2000.
20. Helen Schonenberg, Barbara Weber, Boudewijn van Dongen, and Wil van der Aalst. Supporting flexible processes through recommendations based on history. In *Business Process Management*, volume 5240 of *LNCS*, pages 51–66. Springer, 2008.
21. Fernando Szimanski, Célia G. Ralha, Gerd Wagner, and Diogo R. Ferreira. Improving business process models with agent-based simulation and process mining. In *Enterprise, Business-Process and Information Systems Modeling*, volume 147 of *LNBIP*, pages 124–138. Springer, 2013.
22. Jan van Bon, Mike Pieper, Annelies van der Veen, and Tieneke Verheijen, editors. *Foundations of IT Service Management: based on ITIL.* Van Haren Publishing, 2005.
23. W. M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
24. W. M. P. van der Aalst and A. J. M. M. Weijters. Process mining: a research agenda. *Computers in Industry*, 53(3):231–244, 2004.

25. W. M. P. van der Aalst, A. J. M. M. Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16:1128–1142, 2004.
26. Wil M. P. van der Aalst. Process mining. *Communications of the ACM*, 55(8):76–83, August 2012.
27. W.M.P. van der Aalst. The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
28. W.M.P. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying history on process models for conformance checking and performance analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
29. I. Vanderfeesten, H.A. Reijers, and W.M.P. van der Aalst. Product based workflow support: A recommendation service for dynamic workflow execution. BPM Center Report BPM-08-03, BPMcenter.org, 2008.
30. A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. Alves de Medeiros. Process mining with the HeuristicsMiner algorithm. Technical Report WP 166, Eindhoven University of Technology, 2006.