Discovering Process Models from Unlabelled Event Logs

Diogo R. Ferreira[1] and Daniel Gillblad[2]

[1] IST – Technical University of Lisbon
[2] Swedish Institute of Computer Science (SICS)
`diogo.ferreira@ist.utl.pt, dgi@sics.se`

Abstract. Existing process mining techniques are able to discover process models from event logs where each event is known to have been produced by a given process instance. In this paper we remove this restriction and address the problem of discovering the process model when the event log is provided as an unlabelled stream of events. Using a probabilistic approach, it is possible to estimate the model by means of an iterative Expectaction–Maximization procedure. The same procedure can be used to find the *case id* in unlabelled event logs. A series of experiments show how the proposed technique performs under varying conditions and in the presence of certain workflow patterns. Results are presented for a running example based on a technical support process.

1 Introduction

One of the fundamental principles of workflow and BPM systems is the ability to execute multiple instances of a business process where the behaviour of those instances is governed by a predefined process model [1, 2]. The goal of *process mining* [3] is to rediscover the process model from the run-time behaviour of process instances, assuming that it is possible, namely: to record events as tasks are performed, and to identify the process instance that produced each event.

These requirements are usually met when the run-time behaviour is recorded in an event log containing a sequence of entries in the form <*case id*, *task id*> where *case id* identifies the process instance and *task id* specifies the task that has been performed [3]. Such event logs can be obtained from workflow and case-handling systems, but in applications where there is limited support from process-aware systems it may become difficult to retrieve log data in that form.

In general, it may be possible to record a vast array of events without being able to correlate them to specific process instances. In this scenario, the *case id* attribute is absent and the event log becomes an unlabelled stream of events. Within this stream of events it becomes uncertain whether two events are related or not, as consecutive events may come from different process instances. Also, the number of process instances is unknown.

Our goal is to investigate whether it is possible to discover the process models from such unlabelled event logs. Clearly, the problem of finding the process model in these circumstances is under-defined. However, business processes have distinctive sequential patterns [4] and process instances essentially repeat these patterns over and over again. Based on these premises, it is possible to estimate a probabilistic model that is likely to explain the observed behaviour. In this paper we develop a probabilistic framework for that purpose (section 3).

Similar approaches are not common. In [4] the authors describe two experiments where they had to deal with events without an associated *case id*. In both experiments they resorted to application-specific techniques such as context and data attributes to establish the connection between events. In [5] the authors propose an *iterative workflow mining* approach that could be regarded as being related to the expectation–maximization approach we describe here, but it is used for a different purpose, which is to associate low-level events with high-level tasks. In this work we focus on the specific problem of finding the *case id* in unlabelled event logs (section 4). Section 5 discusses the working assumptions and section 6 concludes the paper.

2 Running example

Fig. 1 illustrates the technical support process for a software product, adapted from a real case study [6]. Basically, the customer calls the vendor to report a problem, the call-center checks if there is an existing contract and records the complaint to be analyzed by the technical support team. The support engineer that is handling the case may either provide a solution or request the development team to fix some bugs. Should the latter become necessary, the development team will have to schedule the release of the bug fix in one of the forthcoming product versions.

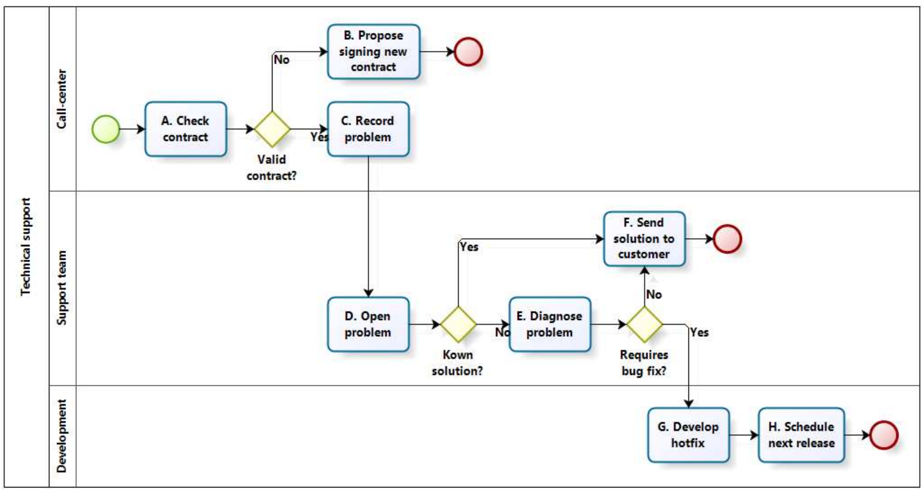


Fig. 1. Technical support process

Whenever the call-center receives a call, a new instance of this process is created. For easier reference, the activities have been labelled with symbol letters from A to H. Let us assume that these activities are recorded in such a way that whenever an activity is completed, the corresponding symbol is recorded in an event log. According to the process model depicted in fig. 1, there are four kinds of possible behaviour: AB, $ACDF$, $ACDEF$ and $ACDEGH$. Each process instance will generate one of these sequences, and the sequences from several process instances may become interleaved since, at any point in time, there may be a number of cases running concurrently.

Fig. 2 shows the events recorded during the execution of 20 process instances. A total of 86 symbols have been recorded, as shown in the first two lines. The complete sequence – called the *symbol sequence* – is shown in the third line, and again below the 20 separate instances. The last three lines display two distinct features: the two lines before the last contain the instance number for each recorded event – this is called the *source sequence* – and it refers to the same set of numbers as printed in the leftmost column of the figure. The very last line displays a count of the total number of instances running concurrently at the time when each event was recorded. In this example it can be seen that there were at most 5 instances running concurrently at different points in time.

```
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
   ACDAEACCDDAEFFAFCCDFABAADACCCDDAFEDGABEGFCADEHBHACAGHACDEDECDAFAFAFCACDCDEAFCDFCGHDDFF
 1 ACD.E..........F......................................................................
 2 ...A...CD...F.........................................................................
 3 .....AC..D.E.F........................................................................
 4 ..........A.....C.DF..................................................................
 5 ..............A..C......D.............EG.......H......................................
 6 ....................AB................................................................
 7 ......................A.....C.D..E.G.........H........................................
 8 .......................A..C..D..F.....................................................
 9 .........................A.C......D.....F.............................................
10 ...............................A.....B................................................
11 ....................................A....C.DE......GH.................................
12 ..........................................A...B.......................................
13 ................................................AC.....DE.....F.......................
14 ..................................................A........CD.....F...................
15 .....................................................AC..DE.....F.....................
16 .............................................................A.....C..D....F..........
17 ...............................................................A.....C..DE......GH....
18 .................................................................A.....C.....DF.......
19 ....................................................................A.......C.....D.F.
20 ..........................................................................A....C...D.F
   ACDAEACCDDAEFFAFCCDFABAADACCCDDAFEDGABEGFCADEHBHACAGHACDEDECDAFAFAFCACDCDEAFCDFCGHDDFF
                                  1    11   1111 1 11111111111111111111111111211112111212
   11121332234323514544667859897870879710559121172533411553355446375846976877069880779090
   11122333334443332222222233444444554445544434444322233333333333344444434444445544433322 21
```

Fig. 2. Events recorded for 20 instances of the technical support process

If both the symbol sequence and the source sequence are known then we have the equivalent of an event log with *task id* and *case id*, respectively, and it is possible to discover the process model using existing process mining techniques.

What we want to investigate is whether it is possible to discover the process model when only the symbol sequence is known.

The source sequence may be unknown for a number of reasons, including the fact that the business process may lack an appropriate support system – this is especially true for organizations with a fragmented IT infrastructure comprising several disparate tools and applications. The source sequence may also have to be removed from the event log for privacy reasons, for example to avoid identifying customers, citizens or medical patients. Finally, it could be the case that the event log is captured by systems that forward tasks without being aware of the process logic. In these scenarios it becomes useful to have a technique that is able to discover the hidden logic behind an unlabelled stream of events.

3 Probabilistic approach

Let K be the number of sources[3] that produce symbols according to the same underlying process model. The output of all sources is recorded in a common event log, where symbols produced by any source may become interleaved with symbols produced by other sources. Let $\boldsymbol{x} = \{x_1, x_2, \ldots, x_N\}$ be the symbol sequence of length N where each symbol x_n comes from one of the available K sources. Let $\boldsymbol{s} = \{s_1, s_2, \ldots, s_N\}$ be the unknown source sequence where each element s_n says which source produced the symbol x_n.

For the purpose of estimating the process model from the symbol sequence $\boldsymbol{x}$ alone, we will use a probabilistic approach based on a first-order Markov chain augmented with special start ($\circ$) and stop ($\bullet$) states. Fig. 3 shows one possible representation for the technical support process shown earlier. The transition matrix $\boldsymbol{M}$ specifies the conditional probabilities for the transition between any two symbols; for example, the probability of producing symbol E after symbol D is given by $p(E|D) = \boldsymbol{M}(D, E) = 0.47$. The conditional probabilities in each row add up to 1.0 except in the last row that represents the stop state.

$$\boldsymbol{M} = \begin{array}{c|cccccccccc}
 & \circ & A & B & C & D & E & F & G & H & \bullet \\
\hline
\circ & - & 1.0 & - & - & - & - & - & - & - & - \\
A & - & - & 0.15 & 0.85 & - & - & - & - & - & - \\
B & - & - & - & - & - & - & - & - & - & 1.0 \\
C & - & - & - & - & 1.0 & - & - & - & - & - \\
D & - & - & - & - & - & 0.47 & 0.53 & - & - & - \\
E & - & - & - & - & - & - & 0.5 & 0.5 & - & - \\
F & - & - & - & - & - & - & - & - & - & 1.0 \\
G & - & - & - & - & - & - & - & - & 1.0 & - \\
H & - & - & - & - & - & - & - & - & - & 1.0 \\
\bullet & - & - & - & - & - & - & - & - & - & -
\end{array}$$

Fig. 3. Transition matrix for the technical support process

[3] From this point onwards, we will refer to *process instances* as *sources*.

The special start and stop states do not produce symbols; instead, they are used solely for the purpose of representing the probability of the process beginning or ending with certain symbols. Note that $p(\circ|...) = p(...|\bullet) = p(\bullet|\circ) \triangleq 0$, i.e., there are no transitions to the start state, no transitions from the stop state, and no direct transitions from the start to the stop state, respectively.

The following sections describe how to estimate the transition matrix $\boldsymbol{M}$ and the source sequence $\boldsymbol{s}$ from a given symbol sequence $\boldsymbol{x}$. Since the estimation involves several steps, we proceed incrementally by first explaining how to compute $\boldsymbol{M}$ from both $\boldsymbol{x}$ and $\boldsymbol{s}$ (section 3.1), then how to estimate $\boldsymbol{s}$ from $\boldsymbol{x}$ and $\boldsymbol{M}$ (section 3.2) and finally how to use the two previous steps to iteratively estimate both $\boldsymbol{M}$ and $\boldsymbol{s}$ from $\boldsymbol{x}$ alone (section 3.3). Note that when only $\boldsymbol{x}$ is given, there must be some way to initialize $\boldsymbol{M}$ in order to get the procedure running. This leads to the concept of $\boldsymbol{M}^+$ in section 3.3.

3.1 Estimating $\boldsymbol{M}$ given $\boldsymbol{x}$ and $\boldsymbol{s}$

If both the symbol sequence $\boldsymbol{x}$ and the source sequence $\boldsymbol{s}$ are given, it becomes straightforward to estimate the transition matrix shown in Fig. 3. With both $\boldsymbol{x}$ and $\boldsymbol{s}$ it is possible to separate $\boldsymbol{x}$ into the symbol sequences produced by each source. We define $\boldsymbol{y}^{(k)} = \{y_1^{(k)}, y_2^{(k)}, \ldots, y_{m_k}^{(k)}\}$ of length m_k as the symbol sequence produced by source k alone, where $y_1^{(k)} = \circ$ and $y_{m_k}^{(k)} = \bullet$. Each sequence $\boldsymbol{y}^{(k)}$ can be easily compiled by picking up the symbols x_n for which $s_n = k$ and adding the special start and stop states.

In its simplest form, the joint probability of $\boldsymbol{x}$ and $\boldsymbol{s}$ can be expressed as:

$$p(\boldsymbol{x};\boldsymbol{s}) = \prod_{k=1}^{K} \prod_{i=1}^{m_k-1} \boldsymbol{M}(y_i^{(k)}, y_{i+1}^{(k)}) \tag{1}$$

For any given pair of symbols a and b, it can be shown that the estimator $\hat{\boldsymbol{M}}(a,b)$ that maximizes $p(\boldsymbol{x};\boldsymbol{s})$ is given by[4]:

$$\hat{\boldsymbol{M}}(a,b) = \frac{\sum_k \eta_{(a,b)}(\boldsymbol{y}^{(k)})}{\sum_k \sum_{b'} \eta_{(a,b')}(\boldsymbol{y}^{(k)})} \tag{2}$$

where a and b are symbols and $\eta_{(a,b)}(\boldsymbol{y}^{(k)})$ is the number of times that the transition from a to b occurs in sequence $\boldsymbol{y}^{(k)}$.

From the event log in Fig. 2 we have:

$\boldsymbol{y}^{(1)} = \boldsymbol{y}^{(3)} = \boldsymbol{y}^{(13)} = \boldsymbol{y}^{(15)} = \circ ACDEF \bullet$

$\boldsymbol{y}^{(2)} = \boldsymbol{y}^{(4)} = \boldsymbol{y}^{(8)} = \boldsymbol{y}^{(9)} = \boldsymbol{y}^{(14)} = \boldsymbol{y}^{(16)} = \boldsymbol{y}^{(18)} = \boldsymbol{y}^{(19)} = \boldsymbol{y}^{(20)} = \circ ACDF \bullet$

[4] For this purpose it is convenient to use the log-likelihood $L(\boldsymbol{M}) \triangleq \log p(\boldsymbol{x};\boldsymbol{s} \mid \boldsymbol{M}) = \sum_k \sum_i \log \boldsymbol{M}(y_i^{(k)}, y_{i+1}^{(k)})$. Maximizing this expression in terms of $\boldsymbol{M}(a,b)$ requires the use of a Lagrange multiplier to find the solution of $\partial L/\partial \boldsymbol{M}(a,b) = 0$ subject to the constraint $\sum_{b'} \boldsymbol{M}(a,b') = 1$. The solution is the maximum likelihood estimator (MLE) shown in equation (2).

$$\boldsymbol{y}^{(5)} = \boldsymbol{y}^{(7)} = \boldsymbol{y}^{(11)} = \boldsymbol{y}^{(17)} = \circ ACDEGH \bullet$$
$$\boldsymbol{y}^{(6)} = \boldsymbol{y}^{(10)} = \boldsymbol{y}^{(12)} = \circ AB\bullet$$

and therefore $\hat{\boldsymbol{M}}(D,E) = \frac{4\times1+9\times0+4\times1+3\times0}{4\times1+9\times1+4\times1+3\times0} \simeq 0.47$ as before.

3.2 Estimating $\boldsymbol{s}$ given $\boldsymbol{x}$ and $\boldsymbol{M}$

If $\boldsymbol{M}$ would be known, then it would be possible to estimate the source sequence $\boldsymbol{s}$ for a given symbol sequence $\boldsymbol{x}$. In principle we would be interested in finding the optimal source sequence $\hat{\boldsymbol{s}} = \arg\max_{\boldsymbol{s}}\{p(\boldsymbol{x};\boldsymbol{s})\}$ that maximizes the joint probability of $\boldsymbol{x}$ and $\boldsymbol{s}$. Unfortunately, finding $\hat{\boldsymbol{s}}$ is a combinatorial optimization problem where one would have to test all possible source sequences in order to find the set of sequences $\boldsymbol{y}^{(k)}$ that maximize $p(\boldsymbol{x};\boldsymbol{s})$ according to equation (1).

In practice, it is possible to obtain an approximation of $\hat{\boldsymbol{s}}$, denoted by $\tilde{\boldsymbol{s}}$, by following a greedy procedure to pick the most likely source for each symbol in $\boldsymbol{x}$. This procedure is based on the idea that if we know the previous symbol ε_k for every source k, then symbol x_n should be assigned to the source k that is able to produce x_n with the highest transition probability. That is, we choose to make $\tilde{s}_n \leftarrow \arg\max_k\{\boldsymbol{M}(\varepsilon_k, x_n)\}$.

After assigning symbol x_n to source k, the previous symbol ε_k for source k is updated to x_n (i.e. $\varepsilon_k \leftarrow x_n$) and we move on to the next symbol x_{n+1}. We find source $\tilde{s}_{n+1}$ by the same procedure, i.e. $\tilde{s}_{n+1} \leftarrow \arg\max_k\{\boldsymbol{M}(\varepsilon_k, x_{n+1})\}$, and so on, until all symbols in $\boldsymbol{x}$ have been assigned to some source.

Whenever symbol x_n is such that $\boldsymbol{M}(\circ, x_n)$ is higher than $\boldsymbol{M}(\varepsilon_k, x_n)$ for every source k, then a new source is *activated* and x_n is assigned to that newly created source. On the other hand, whenever symbol x_n is such that the transition probability to the stop state $\boldsymbol{M}(x_n, \bullet)$ is higher than the transition probability to any other symbol, then source s_n is *deactivated* and removed from the set of active sources.

The following examples are based on the event log shown in Fig. 2 and the transition matrix in Fig. 3:

- At position 40 there are four active sources whose previous symbols are $\varepsilon_5 = E$, $\varepsilon_7 = G$, $\varepsilon_9 = D$ and $\varepsilon_{11} = A$. The present symbol is $x_{40} = G$ and the probabilities of each active source producing this symbol are $\boldsymbol{M}(E,G) = 0.5$, $\boldsymbol{M}(G,G) = 0$, $\boldsymbol{M}(D,G) = 0$ and $\boldsymbol{M}(A,G) = 0$, respectively. Hence, symbol x_{40} gets assigned to source 5, which sets $\tilde{s}_{40} \leftarrow 5$ and $\varepsilon_5 \leftarrow G$. Note that $\boldsymbol{M}(\circ, G) = 0$, so activating a new source is not an option at this point.
- At position 41 the present symbol is $x_{41} = F$ and the best candidate is source 9 with $\boldsymbol{M}(D,F) = 0.53$, which sets $\tilde{s}_{41} \leftarrow 9$ and $\varepsilon_9 \leftarrow F$. After this, source 9 gets deactivated because $\boldsymbol{M}(F,\bullet) = 1$ and therefore it cannot produce any additional symbols.
- At position 42 there are only 3 active sources with previous symbols $\varepsilon_5 = G$, $\varepsilon_7 = G$ and $\varepsilon_{11} = A$. Symbol $x_{42} = C$ gets assigned to source 11 which has the highest transition probability $\boldsymbol{M}(A,C) = 0.85$. We have $\tilde{s}_{42} \leftarrow 11$ and $\varepsilon_{11} \leftarrow C$.

– At position 43 the present symbol is $x_{43} = A$ and the transition probability from the previous symbol to symbol A is zero for all active sources: sources 5 and 7 have $\varepsilon_5 = \varepsilon_7 = G$ and $\boldsymbol{M}(G, A) = 0$; source 11 has $\varepsilon_{11} = C$ and $\boldsymbol{M}(C, A) = 0$. However, $\boldsymbol{M}(\circ, A) = 1$ and therefore a new source with number 12 is created, setting $\tilde{s}_{43} \leftarrow 12$ and $\varepsilon_{12} \leftarrow A$.

More formally, this procedure can be described as shown in Algorithm 1. Note that at line 4 the set of candidate sources becomes the set of all active sources except those that have previously produced a symbol equal to x_n. In other words, we are assuming that each source does not produce the same symbol more than once (this assumption will be discussed ahead in section 5).

Algorithm 1 Greedy algorithm to compute $\tilde{\boldsymbol{s}} = \{\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_N\}$

```
Input: symbol sequence $\boldsymbol{x}$ and transition matrix $\boldsymbol{M}$
  Let $\Psi$ be the set of currently active sources
  Let $\psi$ be the set of candidate sources ($\psi \subseteq \Psi$)
  Let $K$ be the total number of sources used
  Let $\Omega$ be the set of distinct symbols in $\boldsymbol{x}$
 1: $\Psi \leftarrow \emptyset$
 2: $K \leftarrow 0$
 3: for $n = 1$ to $N$ do
 4:   $\psi \leftarrow \Psi \backslash \{k : x_n \in \boldsymbol{y}^{(k)}\}$
 5:   if $(\psi = \emptyset) \vee (\forall_{k \in \psi} : \boldsymbol{M}(\circ, x_n) > \boldsymbol{M}(\varepsilon_k, x_n))$ then
 6:     $K \leftarrow K + 1$
 7:     $\Psi \leftarrow \Psi \cup \{K\}$        // activate a source
 8:     $\boldsymbol{y}^{(K)} \leftarrow \{\circ\}$
 9:     $\tilde{s}_n \leftarrow K$
10:   else
11:     $\tilde{s}_n \leftarrow \arg\max_{k \in \psi}\{\boldsymbol{M}(\varepsilon_k, x_n)\}$
12:   end if
13:   $\varepsilon_{\tilde{s}_n} \leftarrow x_n$
14:   $\boldsymbol{y}^{(\tilde{s}_n)} \leftarrow \boldsymbol{y}^{(\tilde{s}_n)} \cup \{x_n\}$
15:   if $(\forall_{b \in \Omega} : \boldsymbol{M}(x_n, \bullet) > \boldsymbol{M}(x_n, b))$ then
16:     $\Psi \leftarrow \Psi \backslash \{\tilde{s}_n\}$        // deactivate a source
17:     $\boldsymbol{y}^{(\tilde{s}_n)} \leftarrow \boldsymbol{y}^{(\tilde{s}_n)} \cup \{\bullet\}$
18:   end if
19: end for
Output: source sequence $\tilde{\boldsymbol{s}}$ and separate source sequences $\boldsymbol{y}^{(1 \ldots K)}$
```

3.3 Estimating $\boldsymbol{M}$ and $\boldsymbol{s}$ from $\boldsymbol{x}$ alone

Equipped with equation (2) and Algorithm 1 it is possible to devise an iterative procedure to estimate both $\boldsymbol{M}$ and $\boldsymbol{s}$ when only the symbol sequence $\boldsymbol{x}$ is known. Provided with an initial estimate for $\boldsymbol{M}$ we use Algorithm 1 to obtain $\tilde{\boldsymbol{s}}$; then we use $\tilde{\boldsymbol{s}}$ to separate $\boldsymbol{x}$ into $\boldsymbol{y}^{(k)}$ and by equation (2) we compute $\hat{\boldsymbol{M}}$; these two steps complete one iteration. By repeating these steps, we continuously improve $\hat{\boldsymbol{M}}$ and $\tilde{\boldsymbol{s}}$ until finally none of them changes anymore; at this point a solution has been found. This procedure is described in Algorithm 2.

Algorithm 2 Expectation–Maximization procedure to estimate $\hat{\boldsymbol{M}}$ and $\tilde{\boldsymbol{s}}$

```
Input: symbol sequence $\boldsymbol{x}$
1: initialize $\hat{\boldsymbol{M}} \leftarrow \boldsymbol{M}^{+}$
2: repeat
3:     (E-step) use $\hat{\boldsymbol{M}}$ in Algorithm 1 to obtain $\tilde{\boldsymbol{s}}$ and $\boldsymbol{y}^{(1 \ldots K)}$
4:     (M-step) use $\boldsymbol{y}^{(1 \ldots K)}$ in equation (2) to update $\hat{\boldsymbol{M}}$
5: until ($\hat{\boldsymbol{M}}$ does not change)
Output: transition matrix $\hat{\boldsymbol{M}}$ and source sequence $\tilde{\boldsymbol{s}}$
```

Algorithm 2 is essentially an Expectaction–Maximization technique [7] to estimate the model parameters $\boldsymbol{M}$ from the incomplete data $\boldsymbol{x}$, where $\boldsymbol{s}$ is the missing data. The question now is how to initialize $\hat{\boldsymbol{M}}$ in order to get the procedure running. The simplest way to do this is to randomize $\hat{\boldsymbol{M}}$ subject to the stochastic constraints $\sum_{b'} \boldsymbol{M}(a, b') = 1$. However, this random initialization will, in general, lead to a sub-optimal solution as there are many local maxima of the likelihood function where Algorithm 2 will converge. Instead, we need a better way to initialize $\hat{\boldsymbol{M}}$ in order to have a starting point that is actually closer to an optimal solution.

Let

$$\boldsymbol{M}^{+}(a, b) \triangleq \frac{\eta_{(a,b)}(\boldsymbol{x})}{\sum_{b'} \eta_{(a,b')}(\boldsymbol{x})} \tag{3}$$

be the *global model* where $\eta_{(a,b)}(\boldsymbol{x})$ is the number of times that transition a to b occurs in sequence $\boldsymbol{x}$ (i.e. where a and b are consecutive symbols). This global model captures the transition probabilities as if the symbol sequence $\boldsymbol{x}$ had been produced by a single source. Even if $\boldsymbol{x}$ is the result of interleaving a number of sources, their underlying behaviour will be present in $\boldsymbol{M}^{+}$ since consistent behaviour will stand out with stronger transition probabilities than the spurious effects of random interleaving. Therefore, $\boldsymbol{M}^{+}$ is a good initial guess for the estimation of $\boldsymbol{M}$.

3.4 Example

From the process shown in Fig. 1, an event log of 300 sources was generated, having at most 5 overlapping sources. The event log was generated via simulation, using the same ratios as in Fig. 2, i.e. about $9/20 = 45\%$ of $ACDF$, $4/20 = 20\%$ of $ACDEF$, $4/20 = 20\%$ of $ACDEGH$, and $3/20 = 15\%$ of AB. After running Algorithm 2 on the symbol sequence, the transition matrix in Fig. 4 was obtained[5].

[5] Source code for the algorithms and instructions for running examples similar to this one can be found at: `http://web.tagus.ist.utl.pt/~diogo.ferreira/mimcode/`

$$\hat{\boldsymbol{M}} = \begin{array}{c|cccccccccc} & \circ & A & B & C & D & E & F & G & H & \bullet \\ \hline \circ & - & 0.97 & - & - & - & 0.03 & - & - & - & - \\ A & - & - & 0.16 & 0.84 & - & - & - & - & - & - \\ B & - & - & - & - & - & - & - & - & - & 1.0 \\ C & - & - & - & - & 1.0 & - & - & - & - & - \\ D & - & - & - & - & - & 0.41 & 0.59 & - & - & - \\ E & - & - & - & - & - & - & 0.38 & 0.55 & - & 0.07 \\ F & - & - & - & - & - & - & - & - & - & 1.0 \\ G & - & - & - & - & - & - & - & - & 1.0 & - \\ H & - & - & - & - & - & - & - & - & - & 1.0 \\ \bullet & - & - & - & - & - & - & - & - & - & - \end{array}$$

Fig. 4. Estimated transition from an unlabelled a symbol sequence.

Also, from $\boldsymbol{y}^{(1\ldots K)}$ the algorithm estimates that 48.1% of sources produce $ACDF$, 19.8% produce $ACDEGH$, 15.9% produce AB, 13.6% produce $ACDEF$, and there is a small fraction (2.6%) of a single-step sequence E. This last case has some effects on $\hat{\boldsymbol{M}}$, where $\hat{\boldsymbol{M}}(\circ, E) = 0.03$ and $\hat{\boldsymbol{M}}(E, \bullet) = 0.07$; also, this is the reason why the number of sources $K = 308$ was found to be slightly higher than 300.

The transition matrix $\hat{\boldsymbol{M}}$ is shown in graphical form on Fig. 5, where the width of each arc is made proportional to the transition probability. Except for the arcs labeled 0.03 and 0.07 that involve symbol E, the graph depicts the same behaviour as the process model in Fig. 1.

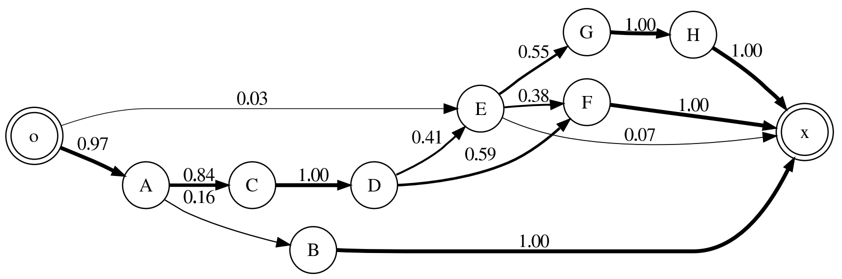


Fig. 5. Estimated model for the technical support process

4 Finding the *case id* in unlabelled event logs

Up to this point we have focused on $\boldsymbol{M}$ as the main outcome of Algorithm 2. However, it is clear that the separate source sequences $\boldsymbol{y}^{(1\ldots K)}$ represent, on their own, a complete event log with both *case id* and *task id*. This suggests that Algorithm 2 can be used as a means to find the *case id* for the activities recorded in an unlabelled event log.

Let $\mathbb{Z}$ be the set of *distinct sequences* in $\boldsymbol{y}^{(1\cdots K)}$. We associate the probability $q(\boldsymbol{z})$ of a sequence $\boldsymbol{z} \in \mathbb{Z}$ with the number of times $\boldsymbol{z}$ occurs in $\boldsymbol{y}^{(1\cdots K)}$. Basically, $q(\boldsymbol{z})$ is the percentage of sequences equal to $\boldsymbol{z}$ in $\boldsymbol{y}^{(1\cdots K)}$. For example, from the results in the example of section 3.4 we have: $q(ACDF) = 0.481$, $q(ACDEGH) = 0.198$, $q(AB) = 0.159$, $q(ACDEF) = 0.136$ and $q(E) = 0.026$.

Let $p(\boldsymbol{z})$ denote the actual distribution one would get if both $\boldsymbol{x}$ and the true source sequence $\boldsymbol{s}$ were known. Then the following metric based on the geometric mean of both distributions can be used to determine how good Algorithm 2 is as a labelling mechanism:

$$G(p \parallel q) \triangleq \sum_{\boldsymbol{z} \in \mathbb{Z}} \sqrt{p(\boldsymbol{z}) \cdot q(\boldsymbol{z})} \tag{4}$$

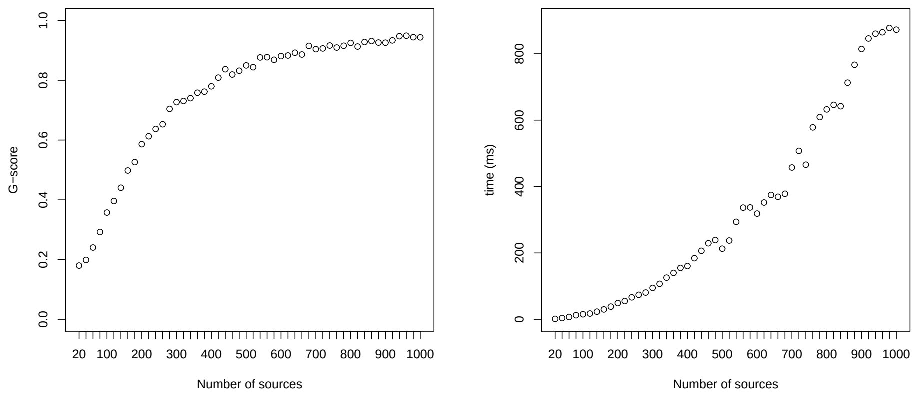
From the example in section 3.4 we have: $G(p \parallel q) = \sqrt{0.45 \times 0.481} + \sqrt{0.2 \times 0.198} + \sqrt{0.15 \times 0.159} + \sqrt{0.2 \times 0.136} + \sqrt{0 \times 0.026} \cong 0.98$, i.e. in this example the algorithm was able to achieve about 98% accuracy in labelling the symbol sequence $\boldsymbol{x}$ with the estimated source sequence $\tilde{\boldsymbol{s}}$.

Once the log is labelled, then it becomes possible to apply process mining techniques such as the α-algorithm [8], the heuristics miner [9], the genetic miner [10], the fuzzy miner [11], or other techniques available in the ProM framework [12]. In general, all these techniques require a labelled event log. If only an unlabelled log is available, then Algorithm 2 can be used as a first pre-processing stage. Also, Algorithm 2 is able to produce a model $\hat{\boldsymbol{M}}$ in the form of a transition matrix but once the log is labelled other process mining techniques can be used to extract other kinds of models such as Petri nets, heuristic nets, etc.

4.1 Accuracy and performance

The metric $G(p \parallel q)$ provides a scoring measure which evaluates the degree of similarity between a complete event log, where both $\boldsymbol{x}$ and $\boldsymbol{s}$ are known, and an incomplete event log $\boldsymbol{x}$ that has been labelled by the estimated source sequence $\tilde{\boldsymbol{s}}$. We will call this metric the G-score; it is a measure of the accuracy of Algorithm 2 as a labelling mechanism for incomplete event logs. In general, this accuracy will depend on the total number of sources in the event log, and on the number of overlapping sources. In principle, the higher the number of sources, the easier it becomes to discover consistent behaviour in the event log. On the other hand, the higher the number of overlapping sources, the more difficult it is to separate the events belonging to different sources.

Fig. 6 shows the results of running Algorithm 2 over symbol sequences with varying number of sources, all having at most 5 overlapping sources. From Fig. 6(a) it is clear that accuracy tends asymptotically to 1.0 as the number of sources (and hence the length of sequence $\boldsymbol{x}$) increases. Fig. 6(b) suggests that execution time evolves quadratically with sequence length; however, it should be noted that the average time per run is below 1 sec. in all experiments.



(a) (b)

Fig. 6. Average G-score (a) and average runtime (b) for input event logs with varying number of sources and having at most 5 overlapping sources. Each point has been obtained by averaging over 1000 synthetically-generated logs.

Fig. 7 shows the results of running Algorithm 2 over symbol sequences of 300 sources with varying number of overlapping sources. From Fig. 7(a) it is apparent that accuracy is exceedingly good when there is little or no overlap at all; it drops dramatically as the number of overlapping sources approaches the length of the sequences produced by each source; but it remains fairly constant and above 0.5 no matter how much the overlap is further increased. Fig. 7(b) suggests that execution time is rather independent of the amount of overlap.

4.2 Parallelism, loops and non-local dependencies

There are a number of workflow patterns [13] that business processes often contain and that may be difficult to capture using process mining techniques. In [14] the authors address the problem of discovering parallel behaviour; in [15] the authors address the problem of mining *short loops* of length one and two; and in [10] the authors present the *drivers license* example where there are non-local dependencies between log events, i.e. where the current symbol depends on a past symbol that has been produced before the immediately previous one.

These and other workflow patterns can become quite challenging to discover since first-order Markov models capture behaviour in terms of the previous state alone. However, experiments suggest that Algorithm 2 can still provide useful insight into the behaviour of processes that contain such patterns. Table 1 presents the results on simple experiments with these patterns.

In models with parallelism it is possible to capture the behaviour by a set of independent sequences. As shown in the last column of Table 1, the top

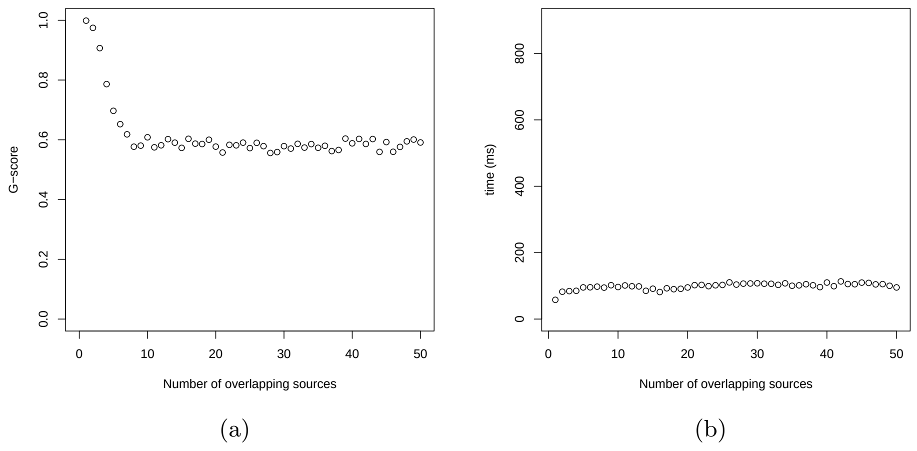


Fig. 7. Average G-score (a) and average runtime (b) for input event logs with 300 sources and varying number of overlapping sources. Each point has been obtained by averaging over 1000 synthetically-generated logs.

three sequences match the original behaviour in $p(\boldsymbol{z})$. There is, however, some amount of mislabelling in the remaining sequences. In particular, the algorithm finds it difficult to establish a relationship between symbol E and the remaining symbols. This explains why $ABCDF$ becomes the fourth strongest sequence and why there are so many sequence variations involving symbol E.

Models with loops pose special problems, as they involve a repetition of symbols. Since Algorithm 1 does not assign repeating symbols to the same source, the solution provided by Algorithm 2 tends to isolate loop behaviour into separate sources, i.e. each loop iteration is assigned to a different source. This is apparent in the loop experiments reported in Table 1, where the second strongest source corresponds to the loop body: BCD for the loop of length 3, CD for the loop of length 2, and C for the loop of length 1.

As for the strongest source, this corresponds to the linear sequence without looping. There is, however, a mismatch between this sequence and the first sequence in the original model: the sequence seems to have been shifted-left with respect to the original behaviour. This can be explained by the fact that the looping behaviour increases the probability of the start symbol being the first symbol in the loop, hence all sequences tend to be shifted to that symbol.

By looking at $p(\boldsymbol{z})$ and $q(\boldsymbol{z})$ in the loop experiments, it becomes apparent that $G(p \parallel q)$ is zero, since there are no common sequences between both distributions. To be able to determine the best solution in these experiments, we relax $G(p \parallel q)$ in order to include the shifting of sequences in $q(\boldsymbol{z})$. For example, in order to match the sequences of both distributions in the loop-3 experiment, we consider a new entity $q^*(\boldsymbol{z})$ where $q^*(BCDEA) = q^*(ABCDE) =$

Pattern	$p(\boldsymbol{z})$	No. symbol sequences	Average G^*-score	Best G^*-score	Best $q(\boldsymbol{z})$
Parallelism	ABCEDF : 0.5 ABECDF : 0.3 ABCDEF : 0.2	1000	0.716	0.854	ABCEDF : 0.398 ABCDEF : 0.180 ABECDF : 0.158 ABCDF : 0.062 ABCDE : 0.037 ABEDF : 0.034 ECDF : 0.031 ABCE : 0.028 ABCEF : 0.025 EDF : 0.019 ABEF : 0.009 CDF : 0.006 EF : 0.003 CEDF : 0.003 E : 0.003 CDEF : 0.003
Loop-3	ABCDE : 0.5 ABCDBCDE : 0.25 ABCDBCDBCDE : 0.125 ABCDBCDBCDBCDE : 0.125	1000	0.503	0.539	BCDEA : 0.581 BCD : 0.400 A : 0.010 BCDE : 0.010
Loop-2	ABCDE : 0.5 ABCDCDE : 0.25 ABCDCDCDE : 0.125 ABCDCDCDCDE : 0.125	1000	0.500	0.538	CDEAB : 0.578 CD : 0.402 CDE : 0.010 CDAB : 0.006 AB : 0.004
Loop-1	ABCE : 0.5 ABCCDE : 0.25 ABCCCDE : 0.125 ABCCCCDE : 0.125	1000	0.498	0.537	CDEAB : 0.578 C : 0.401 CDE : 0.010 CAB : 0.006 AB : 0.002 EAB : 0.002 CDAB : 0.002
Non-local dependency	ABCDE : 0.6 AFCGE : 0.4	1000	0.840	0.909	ABCDE : 0.507 AFCGE : 0.320 AFCDE : 0.087 ABCGE : 0.087

Table 1. Estimation results on different patterns. In all experiments, symbol sequences have been generated using 300 sources and at most 5 overlapping sources.

$q^*(EABCD) = q^*(DEABC) = q^*(CDEAB) = 0.581$. This leads to the definition of the G^*-score as $\sum_{\boldsymbol{z}} \sqrt{p(\boldsymbol{z}) \cdot q^*(\boldsymbol{z})}$ whose results are reported in Table 1. For the parallelism and non-local dependency experiments, the G^*-score results are equal to the G-score values.

For non-local dependencies, the algorithm is able to capture the most recurring sequences with relative ease, with only a small percentage of incorrect sequences.

5 Working assumptions

While choosing the source $\tilde{s}_n$ for each symbol x_n Algorithm 1 considers all active sources except those that have already produced symbol x_n earlier on. This means that no source is allowed to produce the same symbol more than once, and therefore the solutions found by Algorithm 2 will have this same characteristic. This restriction is intended to reduce the search space for the source sequence $\tilde{\boldsymbol{s}}$. Without this assumption, every active source remains as a possible candidate for any given symbol, so it becomes more difficult to assign the correct source to each symbol. Also, without this restriction there would be much more local maxima of the likelihood function, making it extremely difficult for Algorithm 2 to find to the optimal solution.

Nevertheless, the algorithm can still capture behaviour in the presence of repeating symbols, although it will forcefully disconnect a sequence when a repeated symbol occurs. Examples appear in the loop experiments in Table 1, where loop iterations are captured as a separate source, contributing to the overall probability of the second strongest sequence in all three experiments.

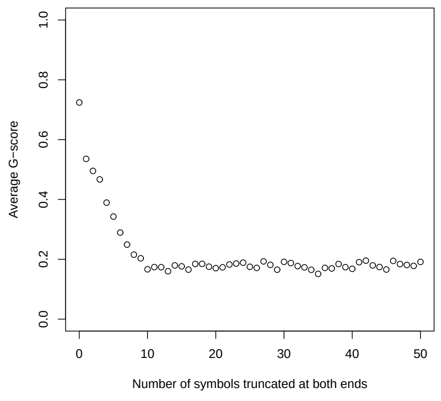
The same behaviour is due to happen in the presence of *duplicate tasks* [16], i.e. when two different activities are represented by the same symbol. Table 2 shows the results of an experiment using an event log (L3) taken from [16]. Some sequences in the original log have duplicate tasks. Algorithm 2 is able to capture some recurring patterns such as BDE, CHF, and BHF but the remaining sequences are broken due to the presence of repeating symbol A.

Pattern	$p(\boldsymbol{z})$	No. symbol sequences	Average G-score	Best G-score	Best $q(\boldsymbol{z})$
Duplicate tasks	BDE : 24 / 61 $\simeq$ 0.393	1000	0.196	0.591	BDE : 0.381
	AABHF : 7 / 61 $\simeq$ 0.115				A : 0.355
	CHF : 15 / 61 $\simeq$ 0.246				CHF : 0.169
	ADBE : 6 / 61 $\simeq$ 0.098				BHF : 0.056
	ACBGDFAA : 1 / 61 $\simeq$ 0.016				BD : 0.010
	ABEDA : 8 / 61 $\simeq$ 0.131				B : 0.009
					F : 0.009
					G : 0.009
					DE : 0.002
					BE : 0.001

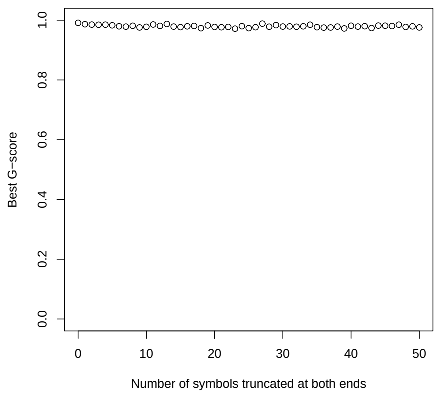
Table 2. Estimation results in an experiment involving log L3 of [16]. Symbol sequences have been generated using 1000 sources and at most 20 overlapping sources.

A second working assumption is that the symbol sequence $\boldsymbol{x}$ contains the complete sequences, from the first to the last symbol, produced by each source. In practice this assumption may not hold, since the symbol sequence $\boldsymbol{x}$ may be an excerpt of recorded behaviour during a period of time. It could be that at the beginning of sequence $\boldsymbol{x}$ some sources were already active, so the first symbols from these sources are missing; at the end of sequence $\boldsymbol{x}$, the last symbols from some of the active sources could also be missing.

To account for this possibility, we consider that $\boldsymbol{x}$ may be truncated at both ends by a certain amount of symbols. Fig. 8 shows the results of running Algorithm 2 on truncated symbol sequences. As the first symbols are truncated, the average G-score drops sharply and then stabilizes around 0.18 when the transient behaviour has been removed and $\boldsymbol{x}$ is left with steady-state behaviour. On the other hand, Fig. 8(b) shows that the best G-score attained remains fairly constant no matter how many symbols are truncated. This means that truncating $\boldsymbol{x}$ makes it more difficult, on average, to find the source for each event, but it does not diminish the ability of the algorithm to find the optimal solution.



(a)



(b)

Fig. 8. Average G-score (a) and best G-score (b) for input event logs generated from the technical support process with 300 sources, 5 overlapping sources, and varying number of symbols truncated at both ends.

6 Conclusion

In this paper we described an Expectation–Maximization approach to estimate the transition matrix $\boldsymbol{M}$ that represents the process model extracted from an unlabelled event log where the *case id* is missing. The probabilistic framework used for this purpose comprises a set of sources that are instances of $\boldsymbol{M}$ and that produce events which become randomly interleaved in the output symbol sequence $\boldsymbol{x}$. Finding the source for each event is a prerequisite for estimating $\boldsymbol{M}$, so the proposed approach can also be used as a labelling mechanism to find the *case id* in unlabelled event logs.

Since $\boldsymbol{M}$ is a first-order Markov model it may be unable to represent certain workflow patterns, but once the log is labelled it is possible to leverage the use of existing process mining techniques to obtain other kinds of models. The experiments reported in this paper show that the proposed approach is capable of labelling log events even in the presence of workflow patterns that $\boldsymbol{M}$ is unable to explicitly represent. This means that the proposed technique can become a valuable aid in the discovery of process models when log data is available as an unlabelled stream of events.

References

1. Hollingsworth, D.: The workflow reference model. Document Number TC00-1003, Workflow Management Coalition (1995)
2. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business process management: A survey. In: Proceedings of the 1st International Conference on Business Process Management (BPM 2003), Berlin, Springer (2003) 1–12
3. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data and Knowledge Engineering **47**(2) (2003) 237–267
4. Ferreira, D., Zacarias, M., Malheiros, M., Ferreira, P.: Approaching process mining with sequence clustering: Experiments and findings. In: Proceedings of the 5th International Conference on Business Process Management (BPM 2007). Volume 4714 of Lecture Notes in Computer Science., Berlin, Springer (2007) 360–374
5. Buffett, S., Geng, L.: Bayesian classification of events for task labeling using workflow models. In: Proceedings of the 4th Workshop on Business Process Intelligence (BPI 08). (2008)
6. Ferreira, D., Mira da Silva, M.: Using process mining for ITIL assessment: a case study with incident management. In: Proceedings of the 13th Annual UKAIS Conference, Bournemouth University (April 2008)
7. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society **39**(1) (1977) 1–38
8. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering **16**(9) (2004) 1128–1142
9. Weijters, A.J.M.M., van der Aalst, W.M.P., Alves de Medeiros, A.K.: Process mining with the heuristics miner algorithm. BETA Working Paper Series WP 166, Eindhoven University of Technology (2006)
10. Alves de Medeiros, A.K., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic process mining: An experimental evaluation. Data Mining and Knowledge Discovery **14**(2) (2007) 245–304
11. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In: Proceedings of the 5th International Conference on Business Process Management (BPM 2007). Volume 4714 of Lecture Notes in Computer Science., Berlin, Springer (2007) 328–343
12. van Dongen, B., de Medeiros, A.A., Verbeek, H., Weijters, A., van der Aalst, W.: The ProM framework: A new era in process mining tool support. In Ciardo, G., Darondeau, P., eds.: Application and Theory of Petri Nets 2005. Volume 3536 of Lecture Notes in Computer Science., Springer-Verlag, Berlin (2005) 444–454
13. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(3) (July 2003) 5–51
14. Cook, J.E., Du, Z., Liu, C., Wolf, A.L.: Discovering models of behavior for concurrent workflows. Computers in Industry **53** (2004) 297–319
15. Alves de Medeiros, A.K., van Dongen, B.F., van der Aalst, W.M.P., Weijters, A.J.M.M.: Process mining: Extending the α-algorithm to mine short loops. BETA Working Paper Series WP 113, Eindhoven University of Technology (2004)
16. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1) (2008) 64–95