Correlation Mining: Mining Process Orchestrations Without Case Identifiers

Shaya Pourmirza(✉), Remco Dijkman, and Paul Grefen

Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{s.pourmirza,r.m.dijkman,p.w.p.j.grefen}@tue.nl

Abstract. Process discovery algorithms aim to capture process orchestration models from event logs. These algorithms have been designed for logs in which events that belong to the same case are related to each other - and to that case - by means of a unique case identifier. However, in service oriented systems these case identifiers are usually not stored beyond request-response pairs, which makes it hard to relate events that belong to the same case. This is known as the *correlation challenge*. This paper addresses the correlation challenge by introducing a new process discovery algorithm, called the *correlation miner*, that facilitates process discovery when events are not associated with a case identifier. Experiments performed on both synthetic and real-world event logs show the applicability of the correlation miner.

Keywords: Process mining · Process discovery · Event correlation

1 Introduction

Over the past decade, there has been an increasing interest in the area of process mining [2,10,16–18,20–22]. The goal of process mining is to extract information about processes from event logs, i.e., execution histories. One of the prominent branches of process mining is process discovery [20], which concerns itself with generating an orchestration model from an event log.

Process discovery techniques assume that an event log contains at least, for each recorded event: (i) a reference to the executed activity, (ii) a reference to the case for which the activity was executed, and (iii) the timestamp at which the activity was completed [21]. Table 1 shows an example of the event log involving 30 events in 10 cases. The main idea behind discovery algorithms is to merge, cluster and aggregate the different cases in an event log and generate a suitable orchestration model based on that. For example, considering the event log from Table 1, it is possible to capture three different traces of execution: $<A, B, E>$ for Cases 1, 4 and 5; $<A, D, E>$ for Cases 2, 3, 8 and 9; and finally, $<A, C, E>$ for Cases 6, 7 and 10. Figure 1 presents the corresponding orchestration model, as it would have been mined by the Disco process mining tool [11].

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant agreement 2012-318275 (GET Service).

A. Barros et al. (Eds.): ICSOC 2015, LNCS 9435, pp. 237–252, 2015.
DOI: 10.1007/978-3-662-48616-0_15

Table 1. An example event log

Case	Activity	Timestamp	Case	Activity	Timestamp	Case	Activity	Timestamp
1	A	00:20	4	B	05:04	7	E	09:17
1	B	02:04	4	E	07:26	8	A	06:20
1	E	02:32	5	A	03:40	8	D	08:36
2	A	02:15	5	B	05:59	8	E	10:03
2	D	03:14	5	E	07:49	9	A	06:41
2	E	05:06	6	A	04:18	9	D	08:56
3	A	02:27	6	C	07:08	9	E	10:20
3	D	04:17	6	E	09:05	10	A	07:13
3	E	06:51	7	A	05:54	10	C	09:10
4	A	03:06	7	C	07:30	10	E	10:26

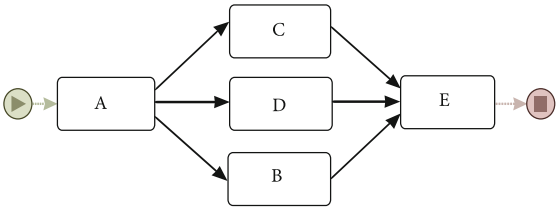


Fig. 1. Orchestration model generated from the example event log

While process mining techniques require an event to be associated with a case, making this association can be problematic. This is also referred to as the *correlation challenge* [18]. This challenge arises, for example, when information about the occurrence of activities is stored in separate service oriented systems and there is no obvious identifier to correlate the occurrences. It is considered especially problematic in service mining [19], because correlations between messages that are being exchanged for the same case, may not be stored beyond request-response pairs. When associations between events that belong to the same case are not present, process discovery becomes impossible, as illustrated by Table 2, which shows the event log from Table 1 without case identifiers. In this event log it is not possible to identify cases and, consequently, it is not possible to cluster and aggregate them into an orchestration model.

Therefore, the goal and contribution of this paper is to develop a process discovery technique that can discover orchestrations when events are neither correlated by case identifiers nor by additional data elements. Although extensive research has been carried out on both process discovery and event correlation techniques, to the best of our knowledge, only one [8] such process discovery technique exists that will be further discussed in this paper.

The remainder of this paper is structured as follows. Section 2 presents some preliminaries regarding the event logs and orchestration models. Section 3 introduces the actual technique that has been developed to capture orchestration

Table 2. An example event log without case identifiers

Activity	Timestamp	Activity	Timestamp	Activity	Timestamp
A	00:20	B	05:04	C	07:30
B	02:04	E	05:06	E	07:49
A	02:15	A	05:54	D	08:36
A	02:27	B	05:59	D	08:56
E	02:32	A	06:20	E	09:05
A	03:06	A	06:41	C	09:10
D	03:14	E	06:51	E	09:17
A	03:40	C	07:08	E	10:03
D	04:17	A	07:13	E	10:20
A	04:18	E	07:26	E	10:26

models from event logs without case identifiers. Section 4 discusses the evaluation setup and summarizes the obtained results. Section 5 compares our technique with related literatures. Finally, Sect. 6 concludes the paper by giving a brief overview and its findings.

2 Preliminaries

In the literature, an event log has been defined as a multiset of cases [18]. However in this paper, we cannot reuse this definition since we have no explicit case. Therefore, we define an event log as follows.

Let A be a set of activities and T be a set of timestamps. $L \subseteq A \times T$ is an *event log*. We use E_a to denote a set of events referring to the activity $a \in A$, such that $E_a = \{(a,t)|(a,t) \in L\}$. In addition, we represent the timestamp at which event $e \in E_a$ has happened by t_e. Note that, strictly speaking, multiple events for the same activity can happen at the same time. However, for simplicity, this is not covered by our definition. While the occurrence of identical events is sufficiently rare to not create problems with the algorithm, it does require some pre-processing of the log to make sure that identical events are distinguished from each other (e.g. by adding a suffix to the timestamp).

There exist various notations in which the orchestration model that is the result of applying a process discovery algorithm can be represented (e.g. Petri-Nets and BPEL) [18]. In this paper, we adopt a rather abstract definition of an orchestration model, which is in line with the notation that is used by the Disco tool for process discovery [11].

The orchestration model is a directed *weighted* graph $G = (\mathcal{V}, \mathcal{E}, \omega)$ such that:

- $\mathcal{V} = \{(a,n)|a \in A \wedge n = |E_a|\}$, where A is the set of activities and n indicates for each node the number of occurrences of that activity in the event log;
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges;

– $\omega : \mathcal{E} \rightarrow \mathbb{N}$ maps edges to natural numbers, where $\omega(a, b) = n$ if and only if there are n observations of activity a being directly followed by activity b in some case in the log. We require that $e \in \mathcal{E}$ if and only if $\omega(e) > 0$.

For a node $v \in \mathcal{V}$, the in-degree, denoted $deg^{in}(v)$, is $\sum_{(w,v)\in\mathcal{E}} \omega(w, v)$. The out-degree, denoted $deg^{out}(v)$, is $\sum_{(v,w)\in\mathcal{E}} \omega(v, w)$. When $deg^{in}(v) = 0$, we call the node a source node. When $deg^{out}(v) = 0$, we call the node a sink node.

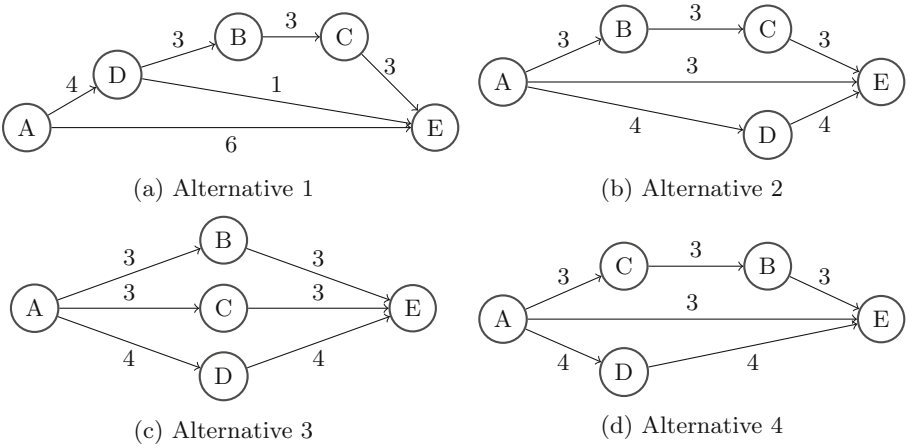
It is important to note that, for any node $(a, n) \in \mathcal{V}$ that is not a sink node, it must hold that $deg^{out}(a) = n$, because the number of cases that pass through the node must also continue to another node. Similarly, for any node $(a, n) \in \mathcal{V}$ that is not a source node, it must hold that $deg^{in}(a) = n$. We call these constraints the *orchestration graph rule*.

3 Correlation Mining Technique

The basic idea of the correlation miner is based on the orchestration graph rule, defined in the previous paragraph, which states that the number of cases that pass through an activity must be equal to the number of incoming and outgoing cases for that activity. We can count the number of cases that pass through an activity a by counting the number of occurrences $|E_a|$ of that activity in the log. Subsequently, we can draw the edges between the activities, in such a way that orchestration graph rule is met. This is a constraint programming problem that can be solved using integer linear programming. At this point we assume that the source and sink nodes are set manually.

As an example, Fig. 2a shows an orchestration model that can be constructed from the log in Table 2. In this log A occurs 10 times, while D occurs 4 times. We manually select A as a source node. Consequently, it has an in-degree of 0. The number of occurrences of A corresponds to the out-degree of A, which is 4 (to D) plus 6 (to E) equals 10. Similarly, the number of occurrences of D corresponds to the in-degree and the out-degree of D. In this way, creating an orchestration model that meets the orchestration graph rule for all activities, produces Fig. 2a. However, Fig. 2b, c, and d are also models that meet this rule. While Fig. 2c is the model that matches the original log from Table 1 best, the orchestration graph rule alone is not enough to determine that.

As the example shows, it is often possible to create more than one model that meets the criteria. Therefore, additional measurements from the event log are required in order to select the best model. To this end, the correlation miner creates two matrices, the *Precede/Succeed matrix* and the *Duration matrix*. Each score in Precede/Succeed matrix, $P/S_{i,j}$, indicates the fraction of events referring to activity i that have occurred before events referring to activity j. If this score is high, it is more likely that there is an edge from i to j. Each score in Duration matrix, $D_{i,j}$, indicates the average time or standard deviation (we will experiment with both alternatives) of the time difference between events referring to activity i and events referring to activity j. If this score is low, it is more likely that there is an edge from i to j. In the remainder of this section,



(a) Alternative 1 (b) Alternative 2

(c) Alternative 3 (d) Alternative 4

Fig. 2. Feasible graphs of example log

we explain how these matrices are computed, how they are used to construct the linear programming problem, and ultimately how the orchestration model is constructed.

3.1 Precede/Succeed Matrix

The first step in the correlation miner is to calculate the Precede/Succeed matrix. This matrix is a square matrix of order n, with $n = |A|$. Let i represent a row in the matrix as well as an activity from A and let j represent a column and an activity. A value $P/S_{i,j}$ in the matrix represent the fraction of events from i that occurred before events from j and is computed as follows. $\Omega_{i,j} = \{(e, f) | e \in E_i \wedge f \in E_j\}$ is the set of all pairs of occurrences of i and j. Furthermore, given events e and f, let $b(e, f)$ be a function that is 1 if $t_e < t_f$ and 0 otherwise.

$$P/S_{i,j} = \frac{\sum_{(e,f)\in\Omega_{i,j}} b(e,f)}{|\Omega_{i,j}|}$$

Returning to our example (Table 2), $P/S_{C,E}$ can be computed as follows. $\Omega_{C,E}$ contains 30 elements since $|E_C| = 3$ and $|E_E| = 10$. $\sum_{(e,f)\in\Omega_{C,E}} b(e, f) = 17$, because in 17 cases C occurred before E. Therefore, $P/S_{C,E} \approx 0.57$.

$$P/S = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0.00 & 0.47 & 0.97 & 0.73 & 0.88 \\ 0.53 & 0.00 & 1.00 & 0.67 & 0.90 \\ 0.03 & 0.00 & 0.00 & 0.33 & 0.57 \\ 0.26 & 0.33 & 0.67 & 0.00 & 0.70 \\ 0.12 & 0.10 & 0.43 & 0.30 & 0.00 \end{pmatrix} \end{array}$$

Fig. 3. P/S matrix for example log

Similarly, we can calculate the value for each pair of activities in a given event log. Figure 3 illustrates the matrix that is calculated based on the example log.

For the performance reasons, we can remove some pairs to not include them for consideration during the next two steps of the algorithm. This can be done via a threshold filter. The goal of the threshold filter is to remove pairs of activities that have a very low probability of forming an edge. For example, in Fig. 3, we have $P/S_{A,C} = 0.97$ and $P/S_{C,A} = 0.03$. These values indicate that it is likely that there is an edge from A to C, but no edge from C to A. The challenge is to select a threshold that removes as many pairs as possible in order to increase performance, but to prevent false positives, because pairs that are removed at this stage will never become edges in the orchestration model.

3.2 Duration Matrix

The second step in the correlation miner is to generate the Duration matrix. This matrix is a square matrix of order n, with $n = |A|$. Let i represent a row in the matrix as well as an activity from A and let j represent a column and an activity. A value $D_{i,j}$ indicates the average time difference or the standard deviation of the time difference between events referring to activity i and events referring to activity j.

In order to calculate the value of $D_{i,j}$ we need to have a *mapping* between elements of E_i and elements of E_j. The idea behind this mapping is that it maps events that belong to the same case. We compute the mapping, by relating the events in such a way that the variance in the time difference is as low as possible, because we argue that the time difference between events from the same case have a more constant probability distribution than the time difference between events from different cases. This principle is not likely to yield a perfect mapping, in the sense that events are mapped if and only if they belong to the same case. However, a perfect mapping is not necessary. We just need a mapping that has a time distribution that is close enough to the perfect mapping.

Formally, let $\Omega_{i,j}$ be defined as in Sect. 3.1. Then we compute the mapping $M_{i,j} \subseteq \Omega i, j$ that satisfies the following constraints.

$$\begin{aligned}
&\forall (e, f) \in M_{i,j} : \exists (e, f') \in M_{i,j} \Rightarrow f = f' \\
&\forall (e, f) \in M_{i,j} : \exists (e', f) \in M_{i,j} \Rightarrow e = e' \\
&\forall (e, f) \in M_{i,j} : t_e < t_f
\end{aligned}$$

These constraints state that each event can be mapped at most once and that each event from E_i must be mapped to an event from E_j that occurs at a later point in time. In addition, the mapping must maximize its size (i.e. as many events must be mapped as possible), while minimizing the standard deviation of the time difference between mapped events. With this mapping, we can compute $D_{i,j}$ as either the mean $\overline{\Delta M_{i,j}}$ or the standard deviation $\sigma_{\Delta Mi,j}$.

$|Mi, j|$ must be maximized
$\sigma_{\Delta Mi,j}$ must be minimized, where $\Delta M_{i,j} = \{t_f - t_e | (e, f) \in M_{i,j}\}$

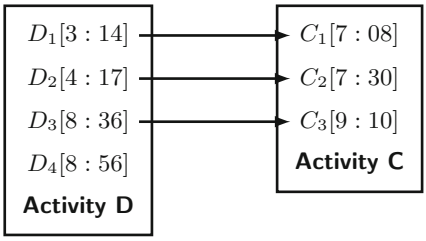


Fig. 4. Mapping of events E_D to events E_C

$$D = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 74 & 109 & 108 & 220 \\ 54 & 0 & 214 & 164 & 69 \\ 6 & 0 & 0 & 87 & 42 \\ 0 & 106 & 150 & 0 & 124 \\ 0 & 102 & 62 & 96 & 0 \end{pmatrix} \end{array}$$

Fig. 5. Duration matrix for example log

Returning to our example (Table 2), $D_{D,C}$ can be computed as follows. Figure 4 shows the mapping from events E_D to events E_C that satisfies the constraints above. The average time difference between mapped events is approximately 153 s. Similarly, we can calculate the value for each pair of activities in a given event log. Figure 5 illustrates the matrix that is calculated based on the example log. Note that the time difference in the matrix is slightly different from the time difference that we computed above. This is because we implemented an efficient (greedy) algorithm to compute the mapping, which does not always produce the mapping that satisfies all constraints. In particular, it does not always return the mapping with the lowest standard deviation, but rather the mapping with a local minimum. Although a lower score of $D_{i,j}$ indicates that it is more likely that there is an edge from i to j, a score of 0 means it is not possible to have such an edge, since we assume that events cannot happen at the same time in the event log.

3.3 Orchestration Model Construction

The final step in the correlation miner is to generate the orchestration model. We first explain how all possible orchestration models can be generated, taking the orchestration graph rule into account. Then we explain how the best orchestration model can be selected from all possible orchestration models, using the Precede/Succeed matrix (Sect. 3.1) and Duration matrix (Sect. 3.2).

In order to construct feasible models, we formulate our problem as an *Integer Linear Programming* (ILP) problem. Each possible edge (i, j) between activities becomes a variable x_{ij} of our ILP problem. The value that is assigned to each variable, x_{ij}, indicates the frequency with which an event for activity i is directly

followed by an event from activity j in a case. Therefore this value can be seen as an edge weight for the edge that connects the node that refers to activity i to the node that refers to activity j. Let $\mathcal{X}$ denote the set that contains the introduced variables. Also, let A_s be the set of activities that we consider start activities and A_e be the set of activities that we consider end activities. We can reduce the problem space, by removing all variables:

- x_{ij} that represent unlikely edges, because of a $P/S_{i,j}$ or $D_{i,j}$ thresholds;
- $x_{is}, s \in A_s$, which represent incoming edges to start activities; and
- $x_{ei}, e \in A_e$, which represent outgoing edges from end activities.

Subsequently, for the resulting set of variables $\mathcal{X}$, we can formulate the constraints of our ILP problem as follows. The lower bound for each variable is 0, while the upper bound for each variable is $min(|E_i|, |E_j|)$, because these are the minimum and the maximum number of cases that can flow on the edges. The sum of the number of cases on the incoming edges of an activity a, must be equal to the number of times an event E_a occurs for that activity. Similarly, the sum of the number of cases on the outgoing edges of an activity a, must be equal to the number of times an event E_a occurs for that activity. Consequently, the constraints become:

$$x_{ij} \geq 0 \qquad \text{for each } a \notin A_s : \sum_{x_{ia} \in \mathcal{X}} x_{ia} = |E_a|$$

$$x_{ij} \leq min(|E_i|, |E_j|) \qquad \text{for each } a \notin A_e : \sum_{x_{ai} \in \mathcal{X}} x_{ai} = |E_a|$$

Returning to the example of the log from Table 2, we can formulate the following constraints. $x_{AE} \geq 0$ and $x_{AE} \leq 10$. Similarly, $x_{AB} \geq 0$ and $x_{AB} \leq 3$. Also, $x_{AE} + x_{BE} + x_{CE} + x_{DE} = 10$ and $x_{AB} + x_{AC} + x_{AD} + x_{AE} = 10$. In addition, $x_{CB} + x_{CD} + x_{CE} = 3$, and $x_{AC} + x_{BC} + x_{DC} = 3$. Note that, since x_{CA} represents an incoming edge to a start node A and x_{EC} represents an outgoing edge from an end node E, these two variables have been removed and consequently are ignored in these constraints.

By applying these constraints, we can construct all feasible orchestration models. Now in the second part of this section, our goal is to select the model, from all feasible ones, that best represents the event log behavior. For this purpose, we employ matrices that we computed in the first two steps of the correlation miner algorithm in order to formulate an *objective function* to select the *optimal* model out of all feasible ones, using ILP principles.

The matrices from step 1 and step 2 provide some evidence to suggest that if activity pair (i, j) contains a high value for $P/S_{i,j}$ and a low value for $D_{i,j}$, these two activities may have a higher chance to form an edge. Based on this statement, our objective is to select the model that has a higher cumulative value of P/Ss and a lower cumulative value of Ds for the edges. To achieve this objective, we define a *coefficient* for each variable x_{ij}, in our ILP in order to consider these values. Since we are interested in a high value of $P/S_{i,j}$ and a low

value of $D_{i,j}$, we define an *edge ratio* as $\frac{D_{i,j}}{P/S_{i,j}}$ for each variable x_{ij} that must be minimized. Note that, we need to keep the sum of edge ratios for each edge independent of the assigned value of the referring variables and count the edge ratio if there is *an* edge, irrespective of the weight of that edge. Thus, ideally, each x_{ij} is divided by its value and set to 0 if $x_{ij} = 0$ in order to eliminate their influence. However, this is not possible in an ILP. Therefore, in order to reduce this influence we divide each variable by its upper bound. In conclusion, we formulate the objective function for our ILP as follow:

$$\mathcal{C}_{ij} = \frac{D_{i,j}}{P/S_{i,j}} \cdot \frac{1}{min(|E_i|, |E_j|)} \quad minimize \sum_{x_{ij} \in \mathcal{X}} \mathcal{C}_{ij} \cdot x_{ij}$$

Returning to the example log (Table 2), by considering the constraints, multiple models can be constructed as shown on Fig. 2. Now, we can calculate the objective function for each of these orchestration models and select the one that has a lowest value as an output of our algorithm. The value for the orchestration model from Fig. 2c is the lowest. Therefore, we select the orchestration model of Fig. 2c as the final result of our correlation miner.

The algorithm that we described above may return a model that contains cycles. However, since the correlation miner only deals with acyclic orchestrations, these cycles should be removed. Therefore, it employs Johnson's algorithm [12] to capture all edges that are involved in cycles in the model. It then calculates the edge ratio for these edges and removes the edge with highest edge ratio. The correlation miner then reruns the third step of the algorithm and repeats this until a model without cycles is returned. As the main loop of the algorithm is ILP solving, the time complexity is polynomial.

4 Evaluation

This section presents the evaluation of the correlation miner. It first presents the setup of the evaluation in Sect. 4.1. Then it presents two evaluations, one using synthetic event logs (Sect. 4.2) and one using a modified version of a real-world event log (Sect. 4.3).

4.1 Evaluation Setup

In order to conduct our evaluation, we have implemented the correlation miner as a Java application. Since the algorithm uses the Gurobi ILP Solver [15], which is commercial software, one needs to install this software on his machine with a proper license. For the interested reader we provide on-line access to the algorithm[1], which can only be used for the academic purposes. Moreover, we are currently investigating the licensing issue in order to implement the algorithm as a ProM [6] plug-in.

[1] http://is.ieis.tue.nl/research/correlation.

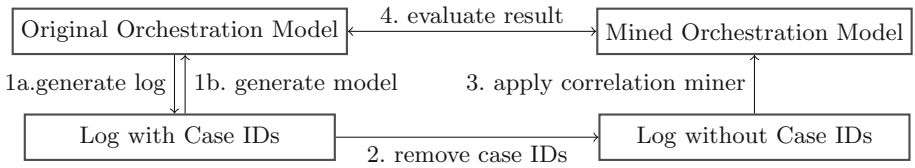


Fig. 6. Evaluation procedure

Figure 6 depicts the procedure for evaluating our algorithm. For the evaluation with synthetic logs, we first create synthetic orchestration models with particular properties, to investigate the effect that these properties have on our algorithm. This will be explained in detail in Sect. 4.2. Subsequently (step 1a), we generate synthetic logs for these models, using the BIMP simulator[2]. For the evaluation with real-world logs, we take a log from practice. Subsequently (step 1b), we generate a orchestration model for that log. This will be explained in detail in Sect. 4.3. Now that we have both a log and a model, we remove the case identifiers from the log (step 2) to generate a log that can be used for correlation mining and (step 3) we apply the correlation miner. Finally (step 4), we assess the quality of the mined model.

In order to measure the quality of the mined models, we use *precision* and *recall*. Precision measures the fraction of edges in the mined model that are correct, i.e. are also in the original orchestration model. Recall measures the fraction of correct edges that have been found. These measures are defined as follows. Let TP be the set of edges that exist in the mined model and also in the original; FN be the set of edges that do not exist in the mined model but do exist in the original model; and FP be the set of edges that exist in the mined model but do not exist in the original model. Then precision and recall are defined as:

$$precision = \frac{TP}{TP + FP} \quad recall = \frac{TP}{TP + FN}$$

Returning to Fig. 2 and assuming that Fig. 2b is the mined model and Fig. 2c the original model, we can calculate precision and recall as follows. TP for these two models is 4, including AB, AD, CE and DE; FN is 2, including AC and BE; and finally FP is again 2, including BC and AE. Consequently, precision and recall for Fig. 2b are approximately 0.67.

In related work fitness and appropriateness have been introduced [16] as measures to evaluate the quality of a mined orchestration model. However, these measures evaluate the quality of an orchestration model as it is compared to a log. We evaluate the quality of the orchestration model as it is compared to another orchestration model; the orchestration model that should have been returned. By doing so, we can get more meaningful results. However, clearly

[2] http://bimp.cs.ut.ee/.

this is only possible when such an orchestration model indeed exists. In other situations fitness and appropriateness should be used.

4.2 Synthetic Event Logs

We evaluated the correlation miner using over 250 synthetic event logs with different properties. We varied properties that we assumed would have an effect on the quality of the mined orchestration model. Specifically, we generated logs:

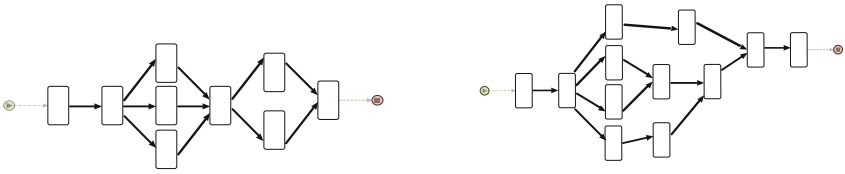
- using both structured and unstructured orchestration models [13] and using orchestration models with different numbers of branches, because these factors influence model complexity and we assume that more complex models are more difficult to mine;
- that contained different numbers of cases, because a higher number of cases provides more data to mine from and should therefore produce more accurate models;
- with different inter-arrival times and activity durations, because time properties play an important role in our algorithm and should therefore have an impact on the quality of the result.

In total we used 6 orchestration models to generate our logs. For reasons of space, we do not present all results, but rather the results for the model that produced the best results (Fig. 7a) and the model that produced the worst results (Fig. 7b). These models primarily vary with respect to whether they are structured or unstructured. The maximum number of branches does not vary much, because our evaluation showed that the number of branches did not play a substantial role in the quality of the results.

Table 3 presents the results for mining logs that were generated from the model from Fig. 7a. We generated different logs from this model, in such a way that the duration of activities and the inter-arrival times, were either:

- distributed uniformly with a median that was selected randomly from the interval $(1, 100]$;
- distributed normally with a mean that was selected randomly from the interval $(1, 100]$ and a standard deviation that was selected randomly from the interval of $\frac{1}{5}$ to $\frac{1}{7}$ of the selected mean;
- distributed exponentially with a mean that was selected randomly from the interval $(1, 100]$.

For each of these three distributions, we produced an event log with 200, 2,000, and 10,000 cases, which is the maximum number of cases that can be produced by the BIMP Simulator. In two experiments, the correlation miner was not able to generate an orchestration model, because after removing some variables (e.g. by removing cycles, or P/S threshold filter), it was unable to solve the model with ILP, as the model became infeasible with regard to its constraints. The results in this table show that the correlation miner can mine a simple structured model perfectly, provided that there are sufficiently many cases in the log.



(a) Structured Model with 3 Branches (b) Unstructured Model with 4 Branches

Fig. 7. Evaluation orchestration models

Table 3. Results of using the correlation miner for Fig. 7a

Distribution	Uniform			Exponential			Normal		
# of cases	200	2000	10000	200	2000	10000	200	2000	10000
Recall	1.00	1.00	1.00	n/a	1.00	1.00	n/a	1.00	1.00
Precision	1.00	1.00	1.00	n/a	1.00	1.00	n/a	1.00	1.00

Table 4 presents an overview of the results for mining logs that were generated from the model from Fig. 7b. Experiments on these logs have been conducted with activity duration distributions that were selected in the same manner as for the model from Fig. 7a, but with inter-arrival times that were either selected in the same manner or from an interval with a longer duration ($(200, 300]$). We also experimented with the ranges $(1, 10]$ and $[10, 10]$. However, these did not lead to substantially different results. Therefore, due to space restrictions, we do not publish those results here. For each of these combinations of probability distributions, we produced an event log with 100, 1,000, and 10,000 cases.

Under most conditions, the results for this more complex unstructured model are worse than for the structured model. If a structured model contains a task with n outgoing flows, there is for sure another task with n incoming flows; however, if an unstructured model contains a task with n outgoing flows, it may not have any task with n incoming flows. Therefore, solving the structured model is easier for our ILP. The other interesting finding to emerge from Table 4 is that if the inter-arrival time between cases is higher than the service time of activities, this leads to substantially better results. Comparing the first 6 column of results in this table with the second 6, shows that in the experiments with higher inter-arrival time the results for recall and precision noticeably increased on average by 27 % and 42 % respectively. This result can be explained, because if the inter-arrival time is the same as the service time, cases are more 'intertwined' in the log, such that the correlation miner has more difficulties telling them apart based on their timing properties (which other mining algorithms can do based on case identifiers).

Also, as expected, and in line with the findings from Table 3, a higher number of cases in the log leads to better models in most cases. However, in experiments in which activities were distributed exponentially, our algorithm did not find a significant difference between the event logs that originally contained 100 and 1000 cases.

Table 4. Results of using the correlation miner for Fig. 7b

Inter-arrival time	[1–100]						[200–300]					
Distribution	Uniform			Exponential			Uniform			Exponential		
# of cases	100	1000	10000	100	1000	10000	100	1000	10000	100	1000	10000
Recall	0.57	0.79	0.64	0.57	0.57	0.71	0.75	0.93	1.00	0.93	0.71	1.00
Precision	0.42	0.65	0.47	0.42	0.42	0.56	0.86	0.87	1.00	0.81	0.59	1.00

Hence, based on these results, we can conclude that the correlation miner is applicable to mine logs without case identifiers, especially for structured models and when the inter-arrival time between cases is higher than the service time of activities and the number of cases in the log is high enough.

4.3 Real-World Event Log

We also evaluated the algorithm based on a real-world event log. To this end, we used a modified version of the log from the BPI Challenge of 2012 [5]. First, we mined an orchestration model from this event log using Disco [11]. We then removed all loops in order to make it acyclic, because that our algorithm only deals with acyclic models at this stage. For the same reason, we then removed the cases from the log in which loops appeared. The resulting version of this real-world event log contained 13 activities, 70,425 events, and 11,647 cases.

In accordance with the evaluation procedure described in Sect. 4.1, we then removed all case identifiers from the log, mined the log using the correlation miner and evaluated the quality of the mined model. The results obtained from this experiment show a precision of 85 % and a recall of 63 %. The lower result for recall can be explained by the fact that our algorithm seeks to find edges with higher frequencies, while we used the Disco model that included all possible edges that could be mined from the log, not just the ones with the higher frequency. Normally speaking, one would not be interested in the Disco model that contains all edges, because this model also includes edges that represent exceptional situations. Indeed if we remove edges with lower frequency (i.e., <50) from the original model our recall indeed increases to 68 % as precision decreases to 79 %.

Based on these results, we claim that the correlation miner is also applicable to mine real-world event logs

5 Related Work

To the best of our knowledge, the closest work to the correlation miner has been published in [8], where the authors tackled the same problem, but based their analysis on event logs neither with case id nor with timestamps. Since the correlation miner also uses timestamps, it is expected to produce more reliable results. Specifically, for the real-world event log (Sect. 4.3), [8] yields a model

with precision 61 % and recall 41 %, which are 24 % and 22 % lower than the correlation miner's results respectively. However, [8] can process cyclic orchestration models.

As suggested in [1], correlating events in an event log is a continuing challenge in the field of process mining, and more specifically, in service mining. Therefore, we discuss the rest of related works in three categories: (i) process discovery, (ii) service mining, and (iii) event correlation.

A large and growing body of literature has been published in the field of process discovery. Van Dongen et al. reviewed discovery algorithms that generate orchestration Petri Net notation [21]. Also, other surveys such as [17], shown that a significant number of studies employ different approaches to discover orchestration models from event logs, such as fuzzy algorithm [10]. A recent study by Verbeek and van der Aalst [22] introduced a generic divide-and-conquer framework to enable process discovery and conformance checking for big event logs. Their approach is similar to our correlation miner in the sense that both have been implemented using Integer Linear Programming. However, the precise optimization problem that they solve is different.

The topic of service mining has been investigated in [2], which illustrated the potential of applying process mining in the context of web services by employing the ProM as a process mining tool and IBM's WebSphere as a reference system. Furthermore, Dustdar et al. [7] introduced Web Services Interaction Mining, which concerns itself with performing process mining techniques in order to analyze service interactions. Finally, [23] presented a web service mining framework aiming at the discovery of unexpected and interesting service compositions. To the best of our knowledge, other than [8], no process or service mining algorithms currently exist that can mine a log that contains no case identifiers and no additional data elements to do the correlation on.

Several techniques have been developed to facilitate event correlation in the context of service-oriented systems. [3] identified a set of 18 correlation patterns that have been grounded in a formal model. The concept of correlation set, a query to retrieve identifiers from messages that are unique for a particular cases, has been included in many correlation techniques such as [9], which proposed an algorithm that assigns a certain identifier to each case in a multi-party supply chains. De Pauw et al. carried out a study to discover conversations in web services by using semantic correlation analysis [4], in which different services pass dedicated identifiers inside their messages. Moreover, in [14] the authors developed an interactive semi-automated tool for event correlation from web service interaction logs. In contrasts, our algorithm enables fully automated event correlation and is only based on occurrences and timestamps of events rather than other data elements that are associated with events.

6 Conclusion

In this paper, we have presented an algorithm, the correlation miner, that enables mining of orchestration models from event logs without case identifiers.

The basic idea of this algorithm is to construct an orchestration model, in such a way that the number of cases that flow into and out of an activity should be equal to the number of events that happen for that activity. However, it is often possible to generate more than one orchestration model that meets this rule. Therefore, we defined additional criteria that need to be fulfilled in order to select the best model. Since logs without case identifiers provide two elements of information: (i) how many times an event occurred for a particular activity, and (ii) at which time an event has occurred, we designed our algorithm based on these two characteristics.

Accordingly, the correlation miner has three steps. In the first two steps, it creates two matrices, the Precede/Succeed matrix and the Duration matrix. Given two activities, the corresponding element in the Precede/Succeed matrix indicates the fraction of events referring to the first activity that have occurred before events referring to the second activity. If this value is high, it is more likely that there is an edge from the first to the second activity. Similarly, given two activities, the corresponding element in the Duration matrix indicates the average time difference between events referring to the first activity and events referring to the second activity. If this value is low, it is more likely that there is an edge from the first to the second activity. Finally, in the third step, the correlation miner constructs all possible orchestration models based that meet the rule mentioned above and then it selects the best one based on the values from the Precede/Succeed matrix and the Duration matrix.

To evaluate the applicability of the correlation miner, we performed experiments with both synthetic event logs and a real-world event log. The results from the evaluation show that the correlation miner produces good results under most conditions. In particular, it produced a model with 85 % precision and 63 % recall for the real-world log. The evaluations with the synthetic logs show that results are better for structured models than for unstructured models. Also, results are better when there are more cases in the log to mine from, and when the inter-arrival time of cases is higher than the duration of activities.

The current correlation miner is only able to mine acyclic orchestrations. There is, therefore, a definite further need for research to enable mining of cyclic orchestrations as well. Furthermore, additional work is needed to ensure that our algorithm can produce an accurate average time difference between activities for the Duration matrix, given that we do not know which events belong to the same case and should, therefore, be used to compute the time difference.

References

1. van der Aalst, W.M.P.: Challenges in service mining: record, check, discover. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 1–4. Springer, Heidelberg (2013)
2. van der Aalst, W.M., Verbeek, H.E.: Process mining in web services: the websphere case. IEEE Data Eng. Bull. **31**(3), 45–48 (2008)
3. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation patterns in service-oriented architectures. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 245–259. Springer, Heidelberg (2007)

4. De Pauw, W., Hoch, R., Huang, Y.: Discovering conversations in web services using semantic correlation analysis. In: ICWS, pp. 639–646. IEEE (2007)
5. van Dongen, B.: Bpi challenge 2012. dataset (2012)
6. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: a new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)
7. Dustdar, S., Gombotz, R.: Discovering web service workflows using web services interaction mining. IJBPIM **1**(4), 256–266 (2006)
8. Ferreira, D.R., Gillblad, D.: Discovering process models from unlabelled event logs. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 143–158. Springer, Heidelberg (2009)
9. Gerke, K., Mendling, J., Tarmyshov, K.: Case construction for mining supply chain processes. In: Abramowicz, W. (ed.) Business Information Systems. LNBIP, vol. 21, pp. 181–192. Springer, Heidelberg (2009)
10. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
11. Günther, C., Rozinat, A.: Disco: discover your processes. In: Proceedings of the BPM 2012 Demo Track. CEUR Workshop Proceedings, vol. 940, pp. 40–44 (2012)
12. Johnson, D.B.: Finding all the elementary circuits of a directed graph. SIAM J. Comput. **4**(1), 77–84 (1975)
13. Mendling, J., Reijers, H.A., Cardoso, J.: What makes process models understandable? In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 48–63. Springer, Heidelberg (2007)
14. Motahari-Nezhad, H.R., Saint-Paul, R., Casati, F., Benatallah, B.: Event correlation for process discovery from web service interaction logs. VLDB J. **20**(3), 417–444 (2011)
15. Optimization, G.: Inc. gurobi optimizer reference manual, version 5.0 (2012)
16. Rozinat, A., van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
17. Tiwari, A., Turner, C.J., Majeed, B.: A review of business process mining: state-of-the-art and future trends. BPM J. **1**, 5–22 (2008)
18. van der Aalst, W.: Process Mining. Springer, Heidelberg (2011)
19. van der Aalst, W.: Service mining: using process mining to discover, check, and improve service behavior. IEEE Trans. Serv. Comput. **6**(4), 525–535 (2013)
20. van der Aalst, W., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012)
21. van Dongen, B.F., Alves de Medeiros, A.K., Wen, L.: Process mining: overview and outlook of Petri net discovery algorithms. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 225–242. Springer, Heidelberg (2009)
22. Verbeek, H.M.W., van der Aalst, W.M.P.: Decomposed process mining: the ilp case. In: Fournier, F., Mendling, J. (eds.) BPM 2014 Workshops. LNBIP, vol. 202, pp. 264–276. Springer, Heidelberg (2015)
23. Zheng, G., Bouguettaya, A.: Service mining on the web. IEEE Trans. Serv. Comput. **2**(1), 65–78 (2009)