Deducing Case IDs for Unlabeled Event Logs

Dina Bayomie(✉), Iman M.A. Helal(✉),
Ahmed Awad, Ehab Ezat, and Ali ElBastawissi

Faculty of Computers and Information, Information Systems Department,
Cairo University, Giza, Egypt
{dina.sayed,i.helal,a.gaafar,e.ezat,alibasta}@fci-cu.edu.eg

Abstract. Event logs are invaluable sources of knowledge about the *actual* execution of processes. A large number of techniques to mine, check conformance and analyze performance have been developed based on logs. All these techniques require at least case ID, activity ID and the timestamp to be in the log. If one of those is missing, these techniques cannot be applied. Real life logs are rarely originating from a centrally orchestrated process execution. Thus, *case ID* might be missing, known as *unlabeled* log. This requires a manual preprocessing of the log to assign case ID to events in the log.

In this paper, we propose a new approach to *deduce* case ID for the unlabeled event log depending on the knowledge about the process model. We provide a set of labeled logs instead of a single labeled log with different rankings. We evaluate our prototypical implementation against similar approaches.

Keywords: Unlabeled event log · Missing data · Event correlation · Decision trees · Process mining · Unmanaged business process

1 Introduction

Most of information systems produce event logs as an evidence of the activities that have been executed. An *event log* consists of a set of events. Each event represents an executed *activity* in a business process. Events have specific *timestamps* and might be associated with other *context* data such as the human resources who participated to the completion of the activity, input and output data etc.

Postmortem analysis techniques of an event log, e.g., process discovery [2], conformance checking or process performance analysis assume the existence of *case identifier* associated with each event. A case identifier is important to correlate the different events recorded in a log. However, case identifiers only exist in execution logs of centrally orchestrated process instance, so called *labeled* event log.

Logs with automatically assigned case identifiers are classified as (★★★★) or higher level of maturity of event logs [2], also classified as level-5 of logging information as in [6]. On the other hand, when the process is executed in an unmanaged environment, logs extracted from the different information systems do not have case identifier, so called *unlabeled* event logs. There are many reasons why business processes may produce event logs with missing information and errors [6,8], such as: some events are collected and recorded by humans, as well as the lack of central systems that are aware of the process model. The latter case is the most common case in real life and represents the middle level of the event logs categories [2], as well as level-4 or lower of logging information as in [6]. This calls for a preprocessing step of a fresh *unlabeled* log to assign a case identifier for the different events before any of the log analysis techniques can be applied.

The problem of *labeling* unlabeled logs has received little attention in the community of business process management [1]. The work in [6,7,18] has addressed the issue in the form of directly mining process models from unlabeled event logs. The approach presented in [7] turns an *unlabeled* log into a *labeled* log. However, there might be uncertainty in deducing case ID for an unlabeled event, which means that there are several possible ways to label such a log.

In this paper, we address one of process mining challenges which is *"Finding, Merging, and Cleaning Event Data"* [2]. This challenge is concerned with extracting and preparing event logs for analysis. We are concerned with the subproblem of the preprocessing needed to prepare the unlabeled event logs for any further usages. We propose an approach to *automate* this preprocessing step by deducing the case identifiers (DCI) for the unlabeled event logs. In addition to the execution log, DCI requires as input the executed process model and heuristic information about the execution time of the different activities within the process. The output is a set of labeled event logs, each with a ranking score indicating the degree of trust in the labeling of events within each log.

The remainder of this paper is organized as follows: an overview of the approach along with foundational concepts and techniques are discussed in Sect. 2. In Sect. 3, we present the details of DCI. Implementation details and comparison with related approaches are discussed in Sect. 4. Related work is discussed in Sect. 5. Finally, we conclude the paper in Sect. 6 with a critical discussion and an outlook on future work.

2 Approach Overview

The DCI approach overview is described in Fig. 1. It has three main inputs: the unlabeled event log, the heuristic data, and the process model. Also, it has an optional input: the ranking-score threshold, to display the results based on the user-specified value (by default: display all results). DCI produces a set of labeled event logs due to the inherent uncertainty, as a single unlabeled event might be assigned to more than one case with different probabilities.

There is a preprocessing step to produce an relation matrix between activities of the process model, so called the behavioral profile [21] of the process model. The generated behavioral profile is an *adapted* version of the original in [21], we elaborate more on this shortly.

The case ID *deducing* process starts with the "Build Case Decision Tree" step. It uses the unlabeled event log to construct a decision tree. It benefits from the behavioral profile and the heuristics data to filter for permissible labelings while building the tree. The "Build Event Logs" step generates the different compatible combinations of cases and writes each combination into a different labeled event log along with its ranking

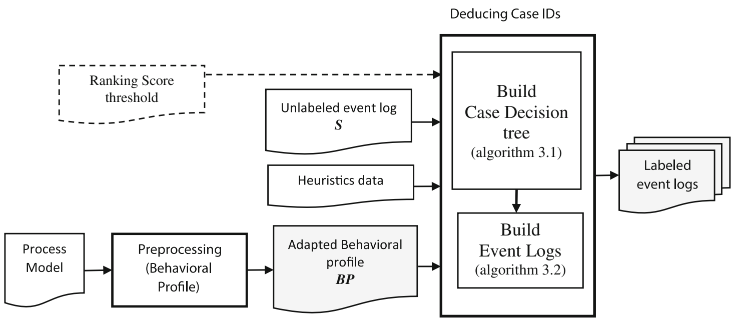

Deducing Case IDs
Ranking Score threshold
Unlabeled event log S
Heuristics data
Build Case Decision tree (algorithm 3.1)
Build Event Logs (algorithm 3.2)
Labeled event logs
Process Model
Preprocessing (Behavioral Profile)
Adapted Behavioral profile BP

Fig. 1. Approach overview

score. The resulted logs provides variety of choices to enhance quality for post analysis and mining techniques. Details about how DCI works are presented in Sect. 3. The rest of this section provides the background concepts needed to help understand how DCI works in addition to discussing the running example.

2.1 Decision Tree

In general, a decision tree represents the different decisions and their possible course of actions. Each node has a set of properties that describe its conditional probability to its parent and how it contributes to decisions in the tree. In the context of this paper, a decision tree is used to represent the different *possible* labelings of each input *unlabeled* event. Each unlabeled event may be represented by more than one node in the tree.

Definition 1 (Case Decision Tree).
$CTree = \langle Node, F, root, Leaves \rangle$

– *Node is the set of nodes within a tree. Each node is further attributed by the caseId, timestamp, activity, and a probability,*
– $F \subseteq Node \times Node$ *is the relationship between nodes,*
– $root \in Node$ *is the root node of the tree, defined with* $caseId = 0$
– $Leaves \subset Node$ *is the set of leaf nodes in the tree.*

A branch(n_i) σ in the tree is the sequence of nodes visited when traversing the tree from the node n_i to the root. $\sigma = n_i, n_{i-1}, ...n_1, root | (root, n_1) \in F \wedge \forall_{i=2}^{j} (n_{i-1}, n_i) \in F.$

Definition 1 describes the structure of the decision tree used in deducing case IDs. Each child of the *root* is a start of a new case, i.e. increments the case ID. Each node carries its conditional probability w.r.t its parent node. This will help calculate the ranking score for the generated labeled log. We elaborate more on that in Sect. 2.3.

2.2 Behavioral Profile

A Behavioral profile (BP) [20,21] describes a business process model (BPM) in terms of abstract relations between activities of the model.

Definition 2 (Behavioral Profile). *Let A be the set of activities within a process model BPM. The behavioral profile is a function* $BP : A \times A \rightarrow \{\perp, \rightsquigarrow, +, ||\}$ *that for any pair of activities defines the behavioral relation as none* $\perp$*, sequence* $\rightsquigarrow$*, exclusive* $+$ *or parallel* $||$.

A behavioral profile returns one of the relationships $\rightsquigarrow, +, ||, or \leftsquigarrow$ for any pair of activities (a, b) that belong to the process model under investigation [21]. However, as per Definition 2, we have restricted the relationships to be defined among adjacent activities only. This is needed for the calculations in the deduction algorithms.

2.3 Additional Information

In Fig. 1, one of the required inputs is *heuristic* data about the execution time of the individual activities within the process model.

Activity Heuristics. Each activity in a business process model has some properties. Such properties could be (timestamp, case ID, data, resource, ...). However, there is some other information related to the expected execution duration of each activity. The execution duration could be in the range $[avg - SD, avg + SD]$, where (avg) is the average execution time, and (SD) is an user-defined standard deviation for execution time. This information is very useful in filtering the case decision tree.

Definition 3 (Execution Heuristics). *Let A be the set of all activities within a process model. Execution Heuristics is a function defined using* $h_{avg} : A \longrightarrow \mathbb{R}$*; is the average execution time of an activity.* $h_{SD} : A \longrightarrow \mathbb{R}$*; is user defined standard deviation execution time of an activity, aka acceptable error.* $h_{range} : [h_{avg} - h_{SD}, h_{avg} + h_{SD}]$*; is Heuristic range.*

Node Probability. As we will be *guessing* about the likelihood of membership of an *event*, in the log, within a specific case, we employ probabilities to assign events as nodes in the decision tree. In general the probability of an activity within an event log is fairly distributed and calculated as:

$$p(activity) = \frac{k}{n} \tag{1}$$

where k is the number of occurrences of an activity in a sample space of size n. In our case the sample space is the unlabeled event log.

Based on Definition 1, each node has a conditional probability that describes the existence of *node* given its *parent*. Equation 2 describes how a node satisfying h_{avg} will be prioritized than other nodes. Equation 3 describes how to calculate the probability of a node in heuristic range.

$$p(h_{avg}) = \frac{m+1}{m^2}; \forall h_{avg} \in H_{avg} \tag{2}$$

$$p(h_{range}) = \frac{m - \frac{|H_{avg}|}{|H_{range}|}}{m^2}; \forall h_{range} \in H_{range} \quad (3)$$

In the above formulas, m is the number of possible parent nodes for the event that will be classified, H_{avg} is the set of nodes satisfying average heuristics, H_{range} is the set of nodes satisfying other parents in heuristic range.

Ranking Score Function. Deducing case IDs for an unlabeled event log will generate different possible labeled event logs. Each of these labeled logs should have a score that reflects the degree to which DCI trusts that events should be correlated that way. Scoring uses the *Rule of Elimination* [19] to describe the probability of an event log, i.e. selected branches for each case, w.r.t the included nodes. The resulting value is divided by the number of the extracted cases from the *unlabeled* event log. Equation 4 shows the scoring function we use.

$$RS(W) = \frac{\sum_{i=1}^{k} p(node|parentNode_i)p(activity)}{number\ of\ cases\ per\ log} \quad (4)$$

where W represents a labeled event log, k is the number of total nodes, i.e. represent the events in the selected branch of the case, in W, *p(activity)* is calculated based on Eq. 1, and $p(node|parentNode_i)$ represents the conditional probability of *node* w.r.t its $parentNode_i$ calculated based on Eqs. 2 and 3.

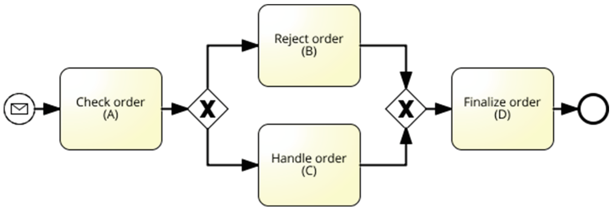


Fig. 2. Simple order handling process

2.4 Running Example

Considering the order business process model in Fig. 2, our approach as described in Fig. 1 needs the following inputs:

(1) The behavioral profile. This is represented as the matrix M for the process model in Fig. 2. The matrix is shown in Fig. 3a.
M is presenting the adapted behavioral profile matrix based on Definition 2. For example, in the model presented in Fig. 2, $BP(A, C)$ is $\rightsquigarrow$, while $BP(A, D)$ is $\perp$, as there is no direct relation between them.

(2) The unlabeled event log S with activity and timestamp pairs, where case ID is unknown. A sample unlabeled log is shown in Fig. 3b.
(3) The activities heuristics h, cf. Definition 3, data about the execution of each activity, i.e. avg, SD, which will affect the filtering process on the case decision tree from the unlabeled log S. Example values of these heuristics for activities of the process in Fig. 2 are shown in Fig. 3c.
(4) The threshold ranking-score (optional) will eliminate some of the generated labeled event log.

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & + & \rightsquigarrow & \rightsquigarrow & \perp \\ B & \perp & + & + & \rightsquigarrow \\ C & \perp & + & + & \rightsquigarrow \\ D & \perp & \perp & \perp & + \end{array}$$

(a) BP Matrix

Case ID	Activity	Timestamp
-	A	2015-01-01 01:00:00
-	A	2015-01-01 02:00:00
-	B	2015-01-01 08:00:00
-	C	2015-01-01 09:00:00
-	D	2015-01-01 13:00:00
-	D	2015-01-01 17:00:00

(b) Unlabeled Event Log (S)

Activity	**Avg**	**SD**
A	5	3
B	5	5
C	7	2
D	7	2

(c) Heuristic Data

Fig. 3. Required input for example in Fig. 2

There are some *assumptions* that are considered while deducing case IDs for events in the unlabeled event log S. First, each event in S has a timestamp that represents the *completion* time of an activity and the start time of the next activity. Second, the process model is an acyclic model. Third, the process model has a single start so we can identify the new case.

The result of DCI is a set of labeled logs that are categorized into either complete or noisy event logs. Complete logs include all events recorded in S. Whereas noisy logs contain inconsistent events with the model, its behavioral profile, or the heuristic data.

3 Deducing Case IDs

In this section, we explain in details how DCI works (cf. Fig. 1). Section 3.1 shows the steps to build the *CTree* from the unlabeled log S. It describes the *filtering* process to avoid incorrect combinations based on the input model and the heuristics data. Section 3.2 illustrates the process of generating the set of *labeled* event logs from the *CTree* with their ranking scores.

3.1 Building Case Decision Tree

The first step in generating labeled event logs is deducing case identifier ($caseId$) for each event in the unlabeled log (S) while building Case Decision Tree ($CTree$).

Algorithm 3.1 builds the $CTree$, cf. Definition 1. It uses the unlabeled event log S, the behavioral profile BP and the heuristic data $Heur$. By processing unlabeled events with their appearance order in S, based on the time stamp, the $CTree$ is built by finding

Algorithm 3.1. Building Case Decision Tree

```
Input: $S$                      //the unlabeled event log (Fig. 3b)
Input: $BP$                     //the behavioral profile (Fig. 3a)
Input: $Heur$                   //the heuristics about activity executions (Fig. 3c)
Output: $Tree$                  // case decision tree $CTree$ in Definition 1
 1: $Tree$ = new $CTree()$
 2: $labelCaseId = 1$
 3: for all ($s \in S$) do
 4:     $Parents = modelBasedParentFiltering(s, BP, Tree)$            //using Definition 2
 5:     $heurDic = heuristicsBasedParentFiltering(s, Heur, BP, Tree, Parents)$   //using Definition 3
 6:     $Parents = heurDic[avg] \cup heurDic[otherRange]$   //list of possible parents
 7:     for all ($n \in Parents$) do
 8:         $caseId = n.caseId$
 9:         if ($caseId == 0$) then          //$n$ represents $root$
10:             $caseId = labelCaseId$       //defines a new case
11:             $labelCaseId += 1$
12:         end if
13:         $node = new\ Node()$              // in Definition 1
14:         $node.setProbability(heurDic)$   //calculated using (Eq: 2, 3)
15:         $node.setTimestamp(timestamp)$, $node.setActivity(activity)$, $node.setCaseID(caseId)$, $node.setParent(n)$
16:         $Tree.addNode(node)$;
17:     end for
18: end for
```

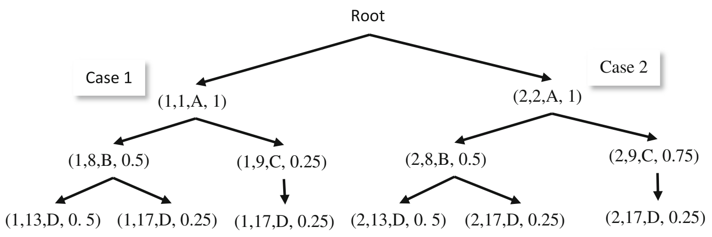


Fig. 4. Case decision tree

the different possible parents of event $s \in S$. The candidate parents are identified based on model and heuristic data (cf. Definitions 2 and 3). For candidate parents that pass the filtering steps, a new node, representing the labeled version of s, is added as a child of the candidate parent respectively with a probability computed based on (Eqs. 2 and 3). Due to space limitations, we excluded the details of model- and heuristic-based filtering, these details can be found in [3].

Figure 4 presents the decision tree generated by Algorithm 3.1 for the inputs in Fig. 3. The tuple (id, ts, a, p) with each node defines the deduced case ID, the time stamp, the activity name and node *probability* respectively. Timestamps are abstracted to hours from the original timestamps in Fig. 3b. In Fig. 4, event $(9; C)$ is represented in $Tree$ by two nodes, case 1 includes one node with probability 0.25 for this event and the same for case 2 but with probability 0.75 based on Eqs. 2 and 3. In order to assign node $(9; C)$, it is checked w.r.t. its heuristics avg = 7,SD = 2 for its set of possible parents, i.e. $(1, 1, A, 1), (2, 2, A, 1)$. Hence, children nodes for node $(2, 2, A, 1)$ are calculated using Eq. 2, while children nodes for node $(1, 1, A, 1)$ are calculated using Eq. 3. Also note that the event $(17; D)$ is represented by four nodes, in case 1 it has two nodes with different parents and the same for case 2.

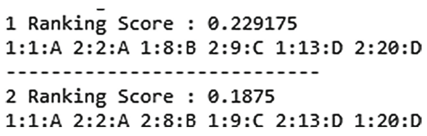
3.2 Generating Labeled Event Logs

Algorithm 3.2 uses the *CTree* built by Algorithm 3.1 to generate a set of *labeled* event logs associated with their ranking score. The generation process avoids any unnecessary event logs, to prevent both *redundant cases* and *duplicated events* in the same labeled event log.

Algorithm 3.2. Generate Labeled Event Log(s)

```
Input: $Tree$                                   // $CTree$ built in algorithm 3.1
Input: $rsThresold$                             // user-defined Ranking Score threshold
Output: $Ws = \{completeEls, NoisyEls\}$ // set of labeled event logs contains both complete and noisy event logs sets
$ELDic = newDict()$                  // {ID:Branches} where branch represents one of possible execution of the case
$EventLogId = 1$
$numOfCases = count(Tree[\text{'root'}].children)$
$completeELs = \{\ \}$   //set of generated complete event log(s)
$noisyELs = \{\ \}$      // set of generated noisy event log(s)
for all ($b \in Tree.Branches$) do
    $tempIDs = getELsIDs(ELDic, caseId-1)$ //set of ids for each log in $ELDic$ includes the previous case($caseId-1$)
    for all ($el_{id} \in tempIDs$) do
        $conflictSet = \{branch\ for\ branch\ in\ ELDic[el_{id}]\ where\ b.caseId\ =\ branch.caseId\}$
        $conflictSet = conflictSet \bigcup \{branch\ for\ branch\ in\ ELDic[el_{id}]\ where\ b.events \bigcap branch.events\}$
        if ($conflictSet\ ==\ \phi$) then
            $ELDic[el_{id}] = ELDic[el_{id}] \bigcup \{b\}$
        else
            $ELDic[EventLogId] = ELDic[el_{id}] - conflictSet$
            $ELDic[EventLogId] = ELDic[el_{id}] \bigcup \{b\}$
            $EventLogId+ = 1$
        end if
    end for
end for
$validLogs = getELsValues(ELDic, numOfCases)$     //set of event logs from $ELDic$ which contain last case
for all ($el \in validLogs$) do
    $rs = RS(el)$   //based on Eq. 4
    if ($length(el)\ ==\ length(S)\ \&\ rs \geq rsThresold$) then
        $completeELs = completeELs \bigcup \{el\}$
    else
        $noisyELs = noisyELss \bigcup \{el\}$
    end if
end for
$Ws = \{completeELs, noisyELs\}$
```

Algorithm 3.2 considers the combination C_r^b, where b is the number of branches in *CTree* and r is the number of cases. Ordering of the branches by *caseId* is used to avoid unnecessary event logs. For example, a branch with leaf node (2:13:D), with $caseId = 2$, checks only the event logs containing branch with $caseId = 1$, cf. Fig. 4. The output categorizes the labeled event logs into *completeELs* and *noisyELs* event logs, based on the given threshold and the number of events in the generated labeled event log.



```
1 Ranking Score : 0.229175
1:1:A 2:2:A 1:8:B 2:9:C 1:13:D 2:20:D
----------------------------
2 Ranking Score : 0.1875
1:1:A 2:2:A 2:8:B 1:9:C 2:13:D 1:20:D
```

Fig. 5. All possible event logs for example in Fig. 2

Figure 5 is the output for the given inputs, cf. Fig. 3, after applying Algorithms 3.1 and 3.2 respectively.

4 Evaluation

In this section, we explain the evaluation setup of our approach. We discuss our prototype in Sect. 4.1. Section 4.2 shows the evaluation procedure and results.

4.1 DCI Implementation

We implemented a prototype[1] for the *DCI* using Python. As a preprocessing of the input, we modified the implementation of the behavioral profile [12] presented in Java as defined in Definition 2. The implementation of DCI is divided into two subprocesses, cf. Fig. 1:

– *Building Case Decision tree* (Algorithm 3.1): Its performance is affected by both the length of the unlabeled event log S and the number of branches in $CTree$. The time complexity of this part is defined as in Eq. 5.

$$O(nm) = n(km + p) \tag{5}$$

where n is the number of events in S, m is the number of $leaf$ nodes in $CTree$, k is the number of activities in the process model, p is the number of nodes in the tree.

– *Generating Labeled Event Log* (Algorithm 3.2): Its performance is affected by the number of generated combinations between $CTree$ branches, where the complexity of bCr increases exponentially with the growth of the number of branches b. Hence, avoiding incorrect combinations, while building labeled event logs, overcomes this problem. The time complexity is defined as in Eq. 6.

$$O(u(m + 1)) = km.bu + u \tag{6}$$

where u is the number of generated labeled event logs, m is the number of $leaf$ nodes in $CTree$, k is the number of activities in the process model, b is the number of branches within the event log, i.e. maximum number of cases in S.

4.2 Evaluation Procedure

Figure 6 shows the evaluation steps of DCI with both synthetic and real life logs. To generate synthetic logs, we use the ProM [16] plug-in: “Perform a simple simulation of a (stochastic) Petri net”. Then the simulated log is updated to reflect the heuristic data. For real life logs, we used the ProM [16] plug-in: “Mine Petri net with Inductive Miner” for inductive mining technique [10] to obtain the process model. Then we extract heuristic information from the real life log using a tool we built. In either case,

[1] Complete implementation in https://github.com/DinaBayomie/DeducingCaseId.

we remove *caseId* from the labeled log to produce an unlabeled log. Also we build the behavioral profile for the process model.

We evaluate DCI against the Expectation-Maximization approach (*E-Max*) [7]. E-Max is a greedy algorithm that mines the unlabeled event log. Table 1 compares between DCI and E-Max. DCI produces multiple labeled event logs. E-Max generates a single log. As a consequence, E-Max is sensitive to overlapping cases [7], which is irrelevant for DCI. Moreover, E-Max partially supports *parallelism*, while DCI fully supports. Neither DCI nor E-max support *Loops*.

Table 1. Comparison between DCI and E-Max [7] features

	Inputs	Event logs	Effect of overlapping cases	Parallelism	Loop
E-Max	Unlabeled log	1	+	+/−	−
DCI	Unlabeled log + Model + Heuristics	*m*	−	+	−

Table 2 shows that execution time of E-Max approach is usually smaller than DCI, since E-Max is a greedy algorithm that generates one labeled log. However, the real log (CoSeLoG project) has many default paths in the original model that affects the breadth of the decision tree in DCI exponentially. Regarding the number of generated cases, DCI is more accurate in determining the instances executed in the event log than E-Max, which is a consequence of considering the process model. From mining the generated event logs, DCI-based models are closer to the original model than E-Max-based model. More details about models and results could be found in https://github.com/DinaBayomie/DeducingCaseId.

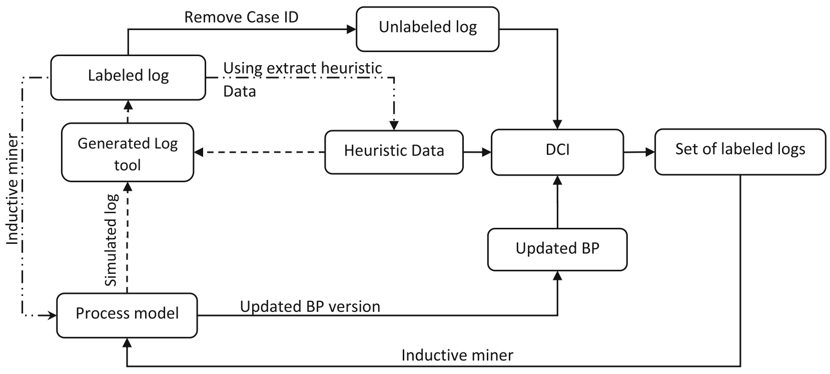


Fig. 6. Evaluation steps

Table 2. Comparison between DCI and E-Max execution

Original model	Log size	Criteria	DCI	E-Max
CoSeLoG project[2]	521 events	Execution time Number of cases	≈ 36000 s 100	≈3.58 s 100
Synthetic log 1	651 events	Execution time Number of cases	18.428 s 100	0.907 s 104
Synthetic log 2	498 events	Execution time Number of cases	10.773 s 100	2.2612 s 149

Receipt phase of an environmental permit application process (WABO), CoSeLoG project http://data.3tu.nl/repository/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6

5 Related Work

In [4,5], a business provenance graph data model was used to generate an automated auditing report. One of the main challenges was to create internal controls to deal with incomplete data. Moreover [11] has presented a method of modeling the uncertainty associated with the raw information and deducing relationships within the provenance model [9]. The main deduced item is the timestamp of an activity, and its level of confidence and accuracy. In [13,14], a stochastic process model is used to repair missing events in the log. It uses path probabilities to determine which are the most likely missing events. We can see that work in [4,5,9,11,13,14] is complementary to our work, where we deduce the missing case identifier, whereas the other work deduces or predicts the timestamp.

There are several process mining techniques that discover and conform the model from event logs. Most of these techniques need a labeled event log to proceed [1]. Also there are different performance analysis techniques that use labeled event logs to extract process performance indicators [15]. We see our work as intermediate step between low quality logs, according to [2], and those mining and analysis approaches.

In [6], authors discuss how to discover web service workflows and the difficulty of finding a rich log with specific information. The execution of web services has missing workflow and case identifiers in order to analyze workflow execution log. They also discuss the need of extra information regarding execution time heuristics.

Moreover, the work in [17,18] discusses the problem from a different point of view. Instead of generating labeled log, it provides a sequence partitioning approach to produce a set partitions that represents the minimum cover of the unlabeled log. The main limitations of the approach are handling loops and also the representation of parallelism as it will represent the execution of concurrent parts of the process into different patterns as if they are not related. We share the same limitation with respect to loops. However, our approach can handle concurrency.

6 Conclusion and Future Work

In this paper, we have introduced an approach to deduce case IDs for unlabeled event logs, DCI. We use as input, in addition to the unlabeled event log, process behavioral profiles and heuristic data about activity execution in order to generate a set of labeled event logs with ranking scores.

DCI handles *noise* in event log as defined in [1]. The *noisy* unlabeled event log might contain a different behavior other than the presented in the process model. Another type of noise is based on *inaccurate* heuristic data with the actual process model execution. Also, DCI handles *incompleteness* of event log [1], i.e. a snapshot from a process execution which violates the process model.

As a limitation, DCI does not support cyclic models. Cyclic models is a problem in most of process mining techniques. Also, the performance of our algorithm is affected by the number of concurrent branches within the process as the number of combinations grows exponentially. Finally, if the heuristic data are inaccurate this will also affect the the number of possible labelings of the event log.

As a future work, we intend to address labeling event logs of cyclic process models. Also, we intend to consider the availability of additional contextual data in the log.

References

1. der Aalst, W.V.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Heidelberg (2011)
2. van der Aalst, W.M.P., 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)
3. Bayomie, D., Helal, I.M.A., Awad, A., Ezat, E., ElBastawissi, A.: Deducing Case IDs for unlabeled Event Logs. Technical report, Cairo University. http://scholar.cu.edu.eg/?q=ahmedawad/files/bplabellingeventlog.pdf
4. Doganata, Y.N.: Designing internal control points in partially managed processes by using business vocabulary. In: ICDE Workshops. pp. 267–272. IEEE (2011)
5. Doganata, Y., Curbera, F.: Effect of using automated auditing tools on detecting compliance failures in unmanaged processes. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 310–326. Springer, Heidelberg (2009)
6. Dustdar, S., Gombotz, R.: Discovering web service workflows using web services interaction mining. Int. J. Bus. Process Integr. Manag. **1**(4), 256 (2006)
7. 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)
8. Herzberg, N., Kunze, M., Rogge-Solti, A.: Towards process evaluation in non-automated process execution environments. In: ZEUS. CEUR Workshop Proceedings, vol. 847, pp. 97–103 (2012). www.CEUR-WS.org
9. Idika, N.C., Varia, M., Phan, H.: The probabilistic provenance graph. In: IEEE Symposium on Security and Privacy Workshops. pp. 34–41. IEEE Computer Society (2013)
10. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M., Wohed, P. (eds.) Business Process Management Workshops. LNBIP, vol. 171, pp. 66–78. Springer, Heidelberg (2014)
11. Mukhi, N.K.: Monitoring unmanaged business processes. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6426, pp. 44–59. Springer, Heidelberg (2010)
12. Polyvyanyy, A., Weidlich, M.: Towards a compendium of process technologies - the jBPT library for process model analysis. In: CEUR Workshop Proceedings onCAiSE 2013 Forum, vol. 998, pp. 106–113 (2013). www.CEUR-WS.org
13. Rogge-Solti, A.: Probabilistic Estimation of Unobservered Process Events. University of Potsdam, Ph.D. (2014)

14. Rogge-Solti, A., Mans, R.S., van der Aalst, W.M.P., Weske, M.: Repairing event logs using timed process models. In: Demey, Y.T., Panetto, H. (eds.) OTM 2013 Workshops 2013. LNCS, vol. 8186, pp. 705–708. Springer, Heidelberg (2013)
15. Suriadi, S., Ouyang, C., van der Aalst, W.M., ter Hofstede, A.H.: Event Gap Analysis: Understanding Why Processes Take Time. Technical report QUT: ePrints (2014)
16. Van Der Aalst, W.M.P., Van Dongen, B.F., Günther, C., Rozinat, A., Verbeek, H.M.W., Weijters, A.: Prom: the process mining toolkit. In: CEUR Workshop Proceedings, vol. 489 (2009)
17. Walicki, M., Ferreira, D.R.: Mining sequences for patterns with non-repeating symbols. In: IEEE Congress on Evolutionary Computation, CEC. pp. 1–8. IEEE (2010)
18. Walicki, M., Ferreira, D.R.: Sequence partitioning for process mining with unlabeled event logs. Data Knowl. Eng. **70**(10), 821–841 (2011)
19. Walpole, E.R., Myers, R.H., Myers, S.L., Ye, K.E.: Probability and Statistics for Engineers and Scientists, 9th edn. Pearson, London (2011)
20. Weidlich, M.: Behavioral profiles - a relational approach to behaviour consistency. Ph.D. thesis, University of Potsdam (2011)
21. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Causal behavioural profiles - efficient computation, applications, and evaluation. Fundamenta Informaticae **113**, 399–435 (2011)