

Evolutionary Learning of Business Process Models from Legacy Systems using Incremental Process Mining

André Cristiano Kalsing, Cirano Iochpe, Lucinéia Heloisa Thom and Gleison Samuel do Nascimento
Department of Informatics, Federal University of Rio Grande do Sul, UFRGS, Porto Alegre, Brazil

Keywords: Evolutionary Learning, Process Mining, Incremental Process Mining, Legacy Systems.

Abstract: Incremental Process Mining is a recent research area that brings flexibility and agility to discover process models from legacy systems. Some algorithms have been proposed to perform incremental mining of process models. However, these algorithms do not provide all aspects of evolutionary learning, such as update and exclusion of elements from a process model. This happens when updates in the process definition occur, forcing a model already discovered to be refreshed. This paper presents new techniques to perform incremental mining of execution logs. It enables the discovery of changes in the process instances, keeping the discovered process model synchronized with the process being executed. Discovery results can be used in various ways by business analysts and software architects, e.g. documentation of legacy systems or for re-engineering purposes.

1 INTRODUCTION

Extraction of business processes from legacy systems can become a very complex task when the continuous evolution of business processes is required. Business processes usually translate the current business needs of a company and they are made up of business rules, activities, control flows, roles and systems. When these business needs change, it is likely that the structure of the process will also change. Consequently, the process models discovered during the mining task, either partial or complete, should also reflect these changes. So, in this case the extraction of business processes cannot be done in one step, but several incremental ones. Moreover, this scenario could be even more complex when the re-engineering process must handle large legacy system with constant maintenance. In this case incremental discovery helps to keep the system maintenance lives while it is modernized.

To overcome these limitations, incremental process mining techniques (Kalsing, 2010a), (Kalsing, 2010b), (Ma, 2011), (Sun, 2007) are designed to allow continuous evolution of the discovered process models. Evolution means that new events executed and recorded in the log can generate new dependencies among activities or remove old ones. Thus, using these techniques we

can significantly improve the discovery process from legacy allowing an iterative approach and also more accurate results.

1.1 Problem Statement

Although some techniques for incremental process mining have already been proposed, there are still aspects to be improved. The main aspect is the better support for update operations during incremental mining of logs, such as removing elements from previously discovered process model when they are considered obsolete elements (e.g. tasks and transitions removed from process definition).

After the discovery of a partial or complete process model from legacy system, any changes contained in modified process instances (e.g. modified business rules in source code) must be merged in those models. Those kinds of changes can be seen in the example of the execution logs shown in Fig. 1 and described next. The logs were generated by the execution of a legacy information system which the source code is partially represented in Fig. 2 and 3. These logs are divided into three logical parts: *Initial Traces*, *New Traces* and *Updated Traces* (see Fig. 1). We can check that the generation of the *Initial Traces* and the *New Traces* log occur when users (i.e. Mary, Paul and Mark) execute specific use cases scenarios within an

information system (e.g. cases A, B and C) presented in Fig. 2. The logs are composed by business rules instances executed and recorded as presented in lines 2, 5, 8, 11, 14, 18 and 20.

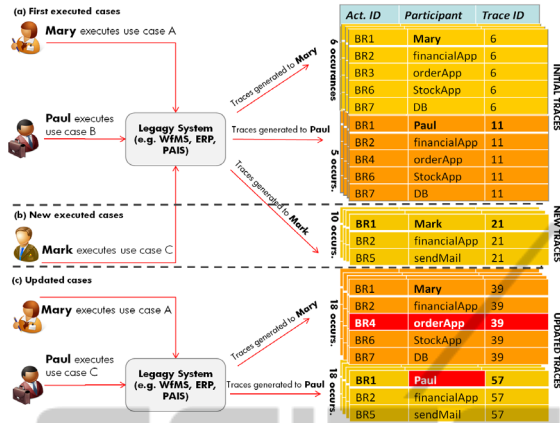


Figure 1: Information System Logs Generation.

The final log (i.e. *Updated Traces*) represents new executions of previously scenarios, but this time generating modified process instances (i.e. see changes highlighted in Fig. 1-c). We can see in this log modified traces, where task BR3 was replaced by task BR4. These changes were generated by the modified version of the source code, presented in Fig. 3. It shows that the business rule presented in lines 7-9 of Fig. 2 was removed from the system. In addition, case C, which was originally executed by user Mark, is now performed by Paul. This specific kind of change is usually not present in source code and can be represented by an organizational change.

```

1  /* MORE CODE HERE */
2  WriteLog(currentUser, "BR1", traceId);
3  printf("PRODUCT/QTY/CUSTOMER: "); //execs business rule 1
4  scanf("%s", product); scanf("%s", qty); scanf("%s", customer);
5  WriteLog("finApp", "BR2", traceId);
6  approved = fin.creditAnalysis(customer); //business rule 2
7  if (approved && price > 1000) {
8      WriteLog("orderapp", "BR3", traceId);
9      newPrice = price - (price * 0.10); // execs business rule 3
10 } else if (approved) {
11     WriteLog("orderapp", "BR4", traceId);
12     newPrice = price - (price * 0.05); // execs business rule 4
13 } else {
14     WriteLog("sendMail", "BR5", traceId);
15     sendMail.Send(customer, "credit refused!"); // exec rule 5
16     return;
17 }
18 WriteLog("stockApp", "BR6", traceId)
19 stock.Update(product, qty); // execs business rule 6
20 WriteLog("Db", "BR7", traceId);
21 query.execute("INSERT INTO orders VALUES (customer,
22 product, qty)"); // executes business rule 7
23 /* MORE CODE HERE */
24
    
```

Figure 2: Example of instrumented source code.

```

.  /* ORIGINAL SOURCE CODE HERE */
7  if (approved) {
8      WriteLog("orderapp", "BR4", traceId);
9      newPrice = price - (price * 0.05); //execs business rule 4
10 } else {
11     WriteLog("sendMail", "BR5", traceId);
12     sendMail.Send(cust, "credit refused!"); //exec bus. rule 5
13     return;
14 }
.  /* ORIGINAL SOURCE CODE HERE */
    
```

Figure 3: Modified version of legacy source code of Fig. 2.

1.2 Contribution

In our previous work (Kalsing, 2010b) we propose an incremental algorithm which is able to perform the incremental mining of process logs (i.e. only the discovery of new process elements) generated from legacy systems or any other kind of system. In this article, we are proposing new algorithms and improving the original ones to detect new and also obsolete elements on discovered process models. We use new heuristics to analyze new and modified process instances recorded in the log in order to incrementally discover process models and mainly update previous discovered process models. This distinguishes our approach from all previous works. Thus, the main contribution of this work is the creation of algorithms that introduce new mining techniques, capable to i) add new discovered activities to an existing model and ii) remove obsolete activities and transitions from the discovered model. As a complementary contribution we introduce a statistical method to measure the quality of discovered models on incremental mining.

The next sections are organized as follow: Section 2 introduces the operations supported by the incremental algorithm. Section 3 introduces details of the incremental mining algorithm and how it performs the creation and update of process models. Experiments using a real legacy system and also simulated logs are presented in section 4. Section 5 introduces the related work and how techniques presented here include considerable improvements over them. In the last section, we present the conclusions and future work.

2 BASIC CONCEPTS

This section introduces the basic concepts related to process mining, which are used in our approach.

2.1 Basic Relations

The techniques presented here mine the control-flow perspective of a process model from logs. In order to find a process model on the basis of an event log, it must be analyzed for causal dependencies, e.g., if an activity is always followed by another, a dependency relation between both activities is likely to exist.

On our previous work (Kalsing, 2010a) we define the basic relations used by incremental mining algorithm. Here we introduce a new relation to represent obsolete relations in a log, where:

1) $[a > wc]_{a \mapsto wb}$ if and only if there is a log of events $W = \sigma_1 \sigma_2 \sigma_3 \dots \sigma_n$ and $i < j$ and $i, j \in \{1, \dots, n-1\}$ and a trace $\sigma = t_1 t_2 t_3 \dots t_n$, where $t_i = a$ and $t_{i+1} = (b \text{ or } c)$ and $\sigma_i \subset a > wc$ and $\sigma_j \subset a \mapsto wb$. The relation $\mapsto w$ represents the sibling relation of $> w$ derived from log of events W , where the relation $> w$ represents a relation older than (i.e. that occurs chronologically before in the log) the sibling relation $\mapsto w$.

2.2 Update Operations



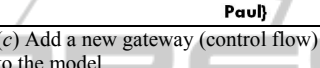

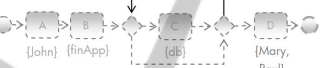
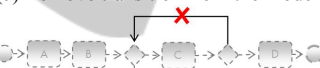
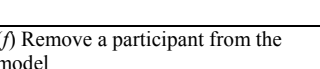
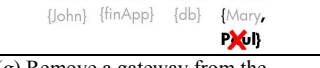
In this section we introduce the update operations that may occur during incremental mining. The first operations (i.e. rows *a-d* of Table 1) represent the insertion of structures in a process model, such as the inclusion of tasks, gateways and participants. These operations were already supported by our previous algorithm and will be complemented here with new removal operations (i.e. rows *e-h* of Table 1).

3 INCREMENTAL MINING

We consider two main phases to extract business process models from legacy: i) identification and annotation of business rules in the source code and ii) incremental mining process. The steps are shown in Fig. 4. First, the source code of the system is analyzed by static methods (Sneed, 1996), (Wang, 2008) to identify business rules in the source. After identified, the business rule is annotated to enable the output of its behavior to log (i.e. as show in fig. 2 and 3). So, the main objective of the identification of business rules is to generate log information about its behavior as input for the next step. Thus, all business process models extracted from legacy systems will be composed by business rules (i.e. tasks) extracted from the source code. The last main step in the process is the incremental process mining

approach. It is used to extract dynamic behavior from the system based on execution of business rules recorded in log data. This phase output is a set of business process structures and task participants, similar to the structures in Fig. 5.

Table 1: Update Operations of Incremental Mining.

| Operation | When it occurs |
|---|--|
| (a) Add a new task to the model  | It occurs when $a \rightarrow wb$ introduces a new transition into the graph, where b (task D) is a new task. |
| (b) Add a new participant to the model  | It occurs when a participant (human or system) starts executing a new task. |
| (c) Add a new gateway (control flow) to the model  | It occurs when $a \rightarrow wb$ introduces a new transition into the graph, where a (task B) has two causal relations (e.g. $B \rightarrow C$ and $B \rightarrow D$). |
| (d) Add a new transition to the model  | It occurs when $a \rightarrow wb$ introduces a new transition into the graph, where a and b were already in the graph. |
| (e) Remove transition from the model  | It occurs when $a \rightarrow wb$ represents a transition that does not belong more to the graph anymore, but a (C) e b (C) must be kept in the graph; |
| (f) Remove a participant from the model  | It occurs when a participant (human or system) does not execute some task anymore. |
| (g) Remove a gateway from the model  | It occurs when $a \rightarrow wb$ represents a transition that does not belong to the graph anymore, where a (B) has now just one causal relation (task C). |
| (h) Remove a task from the model  | It occurs when $a \rightarrow wb$ represents a transition that does not belong more to the graph anymore, where b (D) must be removed from the graph. |

The next sections describe the details of the incremental mining algorithm (i.e. step 5 of Fig. 4) and how it handles incremental knowledge discovery from legacy systems. More details about the whole process described by Fig. 4 can be found in (Kalsing, 2010b).

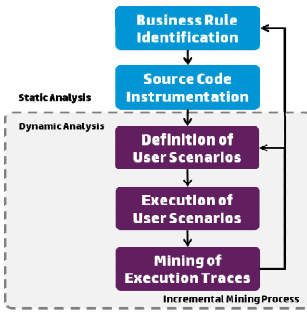


Figure 4: Knowledge Extraction Process.

3.1 Mining of Execution Traces

Our mining algorithm uses a heuristic approach to extract the dependency relations from logs. These heuristics use a frequency based metric to indicate how certain we are that there is a truly dependency relation between two events. The main algorithm is presented in Algorithm 1 (i.e. see Fig. 8) and can be divided in three main steps: 1) mining of dependency graph (i.e. mining of relations \succ_w , \parallel_w and \gg_w) and control-flow semantics; 2) discovery of obsolete relations and 3) mining of participants. The first step discovers all dependency relations from events log, such as sequences, loops, parallelism and control-flows (i.e. see item 2.a.iv of algorithm 1). It is performed applying the heuristics defined in this work and it is covered by the algorithm 2 (i.e. *ProcessRelation*). Then, in the next step, the algorithm discovers obsolete dependency relations from log. These relations represent elements removed from process definition and must be excluded from previous discovered model. In the last main step, we mine the task participants. It is a simple step to discovery and to manage the set of all participants that perform one task. This behavior is covered by algorithm 7 (i.e. *ProcessParticipant*).

3.2 Starting the Mining of Logs

The *IncrementalMiner* algorithm uses three basics information from logs to perform the mining: Task Id, Participant and Trace Id (i.e. log in Fig. 1). To exemplify how the mining of logs is performed, consider the partial log $W = \{BR1BR2BR3BR6BR7^0, BR1BR2BR4BR6BR7^2\}$ (i.e. column Act. ID values of the log shown in Fig. 1-a). *IncrementalMiner* starts iterating on each process instance in the log. Each instance is composed by a set of events (i.e. an execution trace). For each pair of events (i.e. e_1 and e_2) in this trace, we apply a set of heuristics to

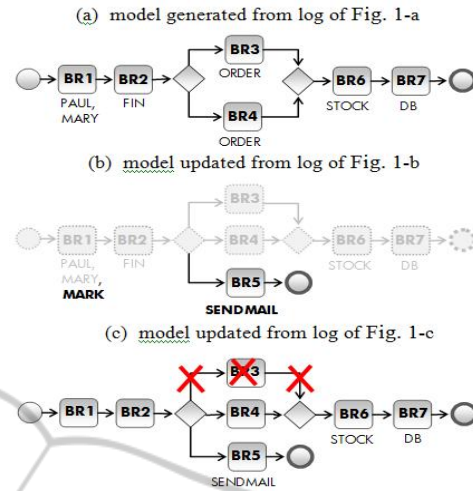


Figure 5: Process model generated from the log in Fig. 1.

discover likely relations (i.e. *ProcessRelation* algorithm). Considering the pair of two first events (e.g. BR1 and BR2) in the log, the first heuristic to be applied is heuristic (1) named *Dependency Relation heuristic*. It verifies if a dependency between the two elements in the trace exists. Let W be an event log on T , and $a, b \in T$. Then $|a \succ wb|$ is the partial number of times $a \succ wb$ occurs in W , and:

$$a \Rightarrow wb = \left(\frac{|a \succ wb| - |b \succ wa|}{|a \succ wb| + |b \succ wa| + 1} \right) \quad (1)$$

The *Dependency Relation heuristic* calculates the *Partial Confidence* (i.e. the *Confidence* value at some point of the algorithm) of this relation, using the support of $a \succ wb$ (e.g. partial number of times that $BR1$ comes before $BR2$) and the support of $b \succ wa$ (e.g. partial number of times that $BR2$ comes before $BR1$). In this example, $a \Rightarrow wb = (11-0) / (11+0+1) = 0.916$ is the partial confidence for the relation $BR1 \rightarrow BR2$. After calculated, this value and all the associated data of dependency relation are inserted in the *Dependency Tree*. *Dependency tree* is an AVL tree that keeps the candidate relations (i.e. relations that could be considered in the final process graph) together with their confidence and support values, respectively. Fig. 6 shows the *Dependency Tree* for the heuristic $a \Rightarrow wb$ calculated above (i.e. see node $BR2$, in the dependency tree $BR1$) and all further dependency relations. *IncrementalMiner* uses AVL trees because of satisfactory time for searching, insertion and removal operations, which is required by the incremental approach.

The result of this and all other heuristics of IncrementalMiner has values between -1 and 1, where values near to 1 are considered good confidences. The next two heuristics below (i.e. heuristics 2 and 3) verify the occurrence of short loops in the trace. That is, it checks the existence of iterations in the trace. Heuristic (2) calculates the confidence of short loops relations with size one (i.e. only one task, e.g. BR1BR1) and heuristic (3) considers loops of size two (i.e. two tasks, e.g. BR1BR2BR1). Let W be an event log over T , and $a, b \in T$. Then $|a >_W a|$ is the number of times $a >_W a$ occurs in W , and $|a >>_W b|$ is the number of times $a >>_W b$ occurs in W :

$$a \Rightarrow wa = \left(\frac{|a > wa|}{|a > wa| + 1} \right) \quad (2)$$

$$a \Rightarrow 2wb = \left(\frac{|a >> wb| - |b >> wa|}{|a >> wb| + |b >> wa| + 1} \right) \quad (3)$$

The heuristic (4) is used to verify the occurrence of non-observable activities in the log (i.e. AND/XOR-split/join semantic elements). Let W be an event log over T , and $a, b, c \in T$, and b and c are in dependency relation with a . Then:

$$a \Rightarrow wb^{\wedge}c = \left(\frac{|b > wc| + |c > wb|}{|a > wb| + |a > wc| + 1} \right) \quad (4)$$

$|a > wb| + |a > wc|$ represents the partial number of positive observations and $|b > wc| + |c > wb|$ represents the partial number of times that b and c appear directly after each other. Considering the partial event log $W = \{\dots, BR1BR2BR3\dots, BR1BR2BR4\dots\}$ extracted from Fig. 1-a, the value of $a \Rightarrow wb^{\wedge}c = (0 + 0) / (6 + 5 + 1) = 0.0$ indicates that $BR3$ and $BR4$ are in a XOR-relation after $BR2$. High values to $a \Rightarrow wb^{\wedge}c$ usually indicate a possible parallel AND-relation and low values a XOR-relation.

The above defined heuristics are used to calculate the candidate relations that can be added to the final dependency graph. In order to select the best relations, we used the best relations trees (see Fig. 7). These trees keep the best dependency and the best causal relations (i.e. relations with the highest confidence value, as presented in Fig. 6) of each dependency tree. Like the dependency tree, the best relations trees are updated after the evaluation of each dependency relation, calculated by heuristics (1), (2) and (3).

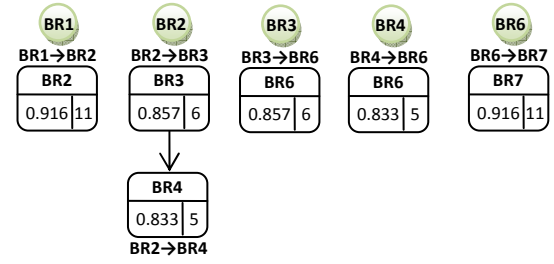


Figure 6: Dependency trees.

The first step to update the best relation tree is to update the best dependency tree for the current relation. Considering the current relation of the example (i.e. $BR1 \rightarrow BR2$), we first check if the confidence value of this relation (i.e. 0.916) is lower than the confidence value of the relations in the best dependency tree of $BR1$ (i.e. see tree $BR1$ in Fig. 7-a). If it is the case, we remove it from the final dependency graph (i.e. see algorithm UpdateBestRelation). Besides, we check if its confidence value is greater than the one of the current best relations in the dependency tree of $BR1$ and remove all the older relations from it (i.e. the older relations have confidence values smaller than the one of the current relation). Finally, we add the current relation to the end of the best dependency tree of $BR1$. The same process is carried out for the second element (i.e. $BR2$). Instead we use the best causal tree of $BR2$ (see Fig. 7-b). All the elements of the dependency tree are considered best dependencies in this simple example. So, the two trees (Fig. 7-a and Fig. 6) are very similar to each other. At the end, we discovered the partial process model (i.e. dependency graph), as shown in Fig. 5-a.

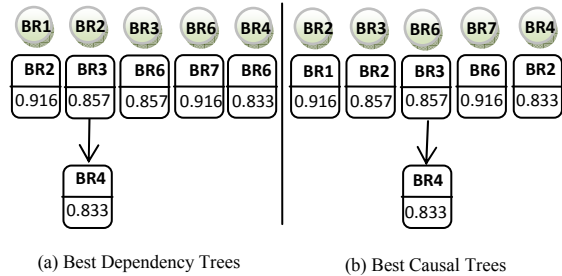


Figure 7: The best relations trees extracted from dependency tree of Fig. 6.

3.3 Adding New Elements to the Model

In an incremental discovery of process models, adding new entries to the log (e.g. new process instances) may reveal additional behavior that should be incorporated into the process model already

discovered. Thus, after process the log of initial traces as described in the previous section (i.e. log of Fig. 1-a), we still need to process the complementary log traces of Fig. 1-b, represented by the log set $W = \{BR1BR2BR5^{10}\}$. After running IncrementalMiner over this log, we will obtain the final dependency graph of Fig. 5b. The dependency graph shows additional structures extracted from the new traces (i.e. BR5 activity and participants Mark and sendMail).

Algorithm 1. IncrementalMiner

Input w : Log, g_1 : DependencyGraph **Out** g_2 : DependencyGraph

1. $g_2 \leftarrow g_1$
 2. **For each** new instance $\sigma \in W$
 - (a) **For each** event $e_i \in \sigma$
 - (i) $e_1 \leftarrow e_i$
 - (ii) $e_2 \leftarrow e_{i+1}$
 - (iii) $e_3 \leftarrow e_{i+2}$
 - (iv) **ProcessRelation**(e_1, e_2, e_3)
 - (v) **CheckOldSiblingRelations**(e_1)
 - (vi) **ProcessParticipant**(e_1)
-

Algorithm 2. ProcessRelation

Input e_1, e_2, e_3 : Event

1. $r_{12} \leftarrow \text{CreateRelation}(e_1, e_2)$.
 2. $\text{confidence} \leftarrow a \Rightarrow wb$, where $a = e_1$ e $b = e_2$.
 3. **IF** $e_1 = e_2$ then
 - (a) $\text{loop1} \leftarrow a \Rightarrow wa$, where $a = e_1$ e $a = e_2$
 - (b) **IF** $\text{loop1} > \text{LOOP1_THSLD}$ then
 - (i) **AddRelationToGraph**(e_1, e_2).
 4. **Else IF** $e_1 = e_3$ then
 - (a) $\text{loop2} \leftarrow a \Rightarrow 2wb$, where $a = e_1$ e $b = e_3$.
 - (b) **IF** $\text{loop2} > \text{LOOP2_THSLD}$ then
 - (i) **AddRelationToGraph**(e_3, e_1).
 5. $n_1 \leftarrow \text{UpdateDependencyTree}(r_{12}, \text{confidence})$
 6. $\text{andSemantic} \leftarrow a \Rightarrow wb \wedge c$, where $a = e_1, b = e_2$ e $c = e_3$
 7. $\text{amd} \leftarrow$ get best relations tree of e_1
 8. $\text{amc} \leftarrow$ get best causes tree of e_2
 9. **UpdateBestRelation**(n_1, amd)
 10. **UpdateBestCause**(n_1, amc)
 11. $n_3 \leftarrow$ get the root of amd tree
 12. $n_1 \leftarrow \text{amd.getNode}(e_2)$
 13. $n_2 \leftarrow \text{amc.getNode}(e_1)$
 14. **IF** ($\neg(n_1 = \emptyset) \vee \neg(n_2 = \emptyset) \vee \text{confidence} > \text{DEPENDENCY_THSLD} \wedge \text{support}(r_{12}) > \text{POSITIVEOBSERVATION_THSLD} \wedge \text{confidence} - n_3.\text{confidence} \leq \text{RELATIVETOBEST_THSLD}$) then
 - (a) **AddRelationToGraph**(e_1, e_2).
 15. **ComputeOldSiblingRelation**(e_1, e_2).
-

Algorithm 3. UpdateBestRelation

Input n_1 : Node, a_1 : Tree

1. $\text{value} \leftarrow \text{CheckConfidence}(n_1.\text{confidence}, a_1)$
2. **IF** $\text{value} = -1$ then //smaller value

(a) **RemoveRelationFromGraph**($n_1.\text{relation}$).

3. **Else IF** $\text{value} = 1$ then //higher value

(a) **For each** node $n_2 \in a_1$

(i) Remove n_2 from a_1

(ii) **RemoveRelationFromGraph**(e_1, e_2)

(b) Add n_1 to a_1

5. **Else IF** $\text{value} = 0$ then //equal value

(a) Add n_1 to a_1

Algorithm 4. CheckOldSiblingRelations

Input e_1 : Event

1. $\text{bestRelations} \leftarrow \emptyset$ //create a temporary set
2. $\text{worstRelations} \leftarrow \emptyset$ // create a temporary set
3. $v_{e_1} \leftarrow$ get vertex related to e_1 from graph g_2
4. $\text{out}_{v_{e_1}} \leftarrow$ get output vertices set related to v_{e_1}
5. **For each** vertex $v_i \in \text{out}_{v_{e_1}}$
 - (a) $r_{12} \leftarrow \text{GetRelation}(e_1, v_i, \text{event})$.
 - (b) $\text{sib}_{v_i} \leftarrow$ get sibling relations set of v_i
 - (c) **For each** sibling relation $ri_1 \in \text{sib}_{v_i}$
 - (i) **IF** $ri_1.\text{confidence} \geq \text{OBSOL_THSLD}$ then
 - (x) Add ri_1 to bestRelations
 - (ii) **Else**
 - (x) Add ri_1 to worstRelations
 - (d) **IF** $\text{sib}_{v_i} \subseteq \text{bestRelations}$ then
 - (i) **RemoveRelationFromGraph**(r_{12})
 - (e) **Else**
 - (i) **For each** relation $r_1 \in \text{worstRelations}$
 - (x) **For each** sibling relation $ri_1 \in r_1$
 - a. **IF** $ri_1.\text{confidence} > \text{OBSOL_THSLD}$ then
 - i. $\text{exclude} \leftarrow ri_1 \in \text{bestRelations} \wedge \text{exclude} = \text{true}$
 - (ii) **IF** $\neg(\text{bestRelations} = \emptyset) \wedge \text{exclude} = \text{true}$ then
 - (x) **For each** relation $r_1 \in \text{worstRelations}$
 - a. **RemoveRelationFromGraph**(r_1)
 - (y) **RemoveRelationFromGraph**(r_{12})

Algorithm 5. ComputeOldSiblingRelation

Input e_1, e_2 : Event

1. $v_{e_1} \leftarrow$ Get vertex related to e_1 from graph g_2
 2. $r_{12} \leftarrow \text{GetRelation}(e_1, e_2)$.
 3. $\text{sib}_r \leftarrow$ get sibling relations set related to r_{12}
 4. **For each** sibling relation $ri_1 \in \text{sib}_r$
 - (a) **IF** $ri_1 = r_{12}$ then
 - (i) $\text{confidence} \leftarrow a \Rightarrow wb \mapsto c$, where $a > wb =$ support of r_{12} , $a > wc =$ support of r and $[a \rightarrow wc]_{a \mapsto wb} =$ support of ri_1
 - (ii) $ri_1.\text{confidence} \leftarrow \text{confidence}$
-

Algorithm 6. CheckConfidence

Input conf : Number, a_1 : Tree **Output**: v : Number

1. $n_1 \leftarrow$ get root node of a_1
 2. **IF** $\text{confidence} > n_1.\text{confidence}$ then
 - (a) $v \leftarrow 1$
 3. **Else IF** $n_1.\text{confidence} = \text{confidence}$ then
 - (b) $v \leftarrow 0$
 4. **Else**
 - (c) $v \leftarrow -1$
-

Algorithm 7. ProcessParticipant

Input e_1 : Event

1. $v_{e_1} \leftarrow$ Get vertex related to e_1 from graph g_2
2. **IF** $\neg e_1.\text{participant} \in v_{e_1}.\text{participants}$
 - (a). Add $e_1.\text{participant}$ to $v_{e_1}.\text{participants}$

Figure 8: Pseudocode of Incremental Mining Algorithms.

3.4 Removing Elements from Model

A process model is considered complete when it replays all events recorded in a complete log (van der Aalst, 2004). In this case, new traces added to the log will not change and neither will add new behavior to the model. However, this definition could not be true when changes occur in the process structure, recording possibly modified instances in the log.

The problem in identifying modified process instances is that this information is often not recorded in the log. What happens in this case is the empirical analysis to check the likelihood of obsolete events (events related to elements that are not longer in the process definition, and consequently they also no longer occur). Thus, the main goal of the algorithms presented in this section and also one of the main contributions of this work is the ability to identify and remove obsolete dependency relations from discovered process models during the incremental mining.

To demonstrate how these algorithms work, we shall return to the log presented in Fig. 1-c. This log contains modified versions of initial process instances and it was recorded by the execution of modified source code presented in Fig. 3. The first change can be observed in the log generated by case A (i.e. executed by the user Mary). This case generates 18 occurrences of trace BR1BR2BR4BR6BR7. However, we can see in Fig. 1-a that this case originally used to record the trace BR1BR2BR3BR6BR7. The change in this case is the replacement of task BR3 by task BR4.

Process instances changes as described above can be often imperceptible during the mining task, because they depend on several factors such as frequency of related events (i.e. events related to task BR4), incidence of noise, etc. Thus, *ComputeOld SiblingRelation* and *CheckOldSiblingRelation* enable the identification of obsolete relations based on previous discovered model, modified process instances and its frequency. An obsolete relation is basically a dependency relation $>_w$ presented in the dependency graph which is not executed anymore. The algorithm *ComputeOldSiblingRelation* is responsible for calculating the confidence of the candidate obsolete relations and it is used by the algorithm *ProcessRelation* (i.e. see item 15 of algorithm 2). To perform this, it uses the heuristic (5) defined bellow. The heuristic calculates the confidence of relation $>_w$ to be an obsolete relation against its sibling relations \mapsto_w . To demonstrate this, we can see the

relation BR2→BR3 in Fig. 9-a. The sibling relations of BR2→BR3 are BR2 \mapsto BR4 and BR2 \mapsto BR5 (i.e. dashed transitions). Thus, we need to apply the heuristic for each sibling relation of BR2→BR3. So, let W be a log of events T , and $a, b, c \in T$. So $|a > wb|$ is the partial number of times that $a > wb$ occurs in W , $|a > wc|$ is the partial number of times that $a > wc$ occurs in W , $|[a > wc]_{a \mapsto wb}|$ is the partial number of times that $[a > wc]_{a \mapsto wb}$ occurs in W and:

$$a \Rightarrow wb \mapsto c = \frac{\left(\left(1 + \frac{|a > wb|}{|a > wc|} \right) \times (|a > wb| - |[a > wc]_{a \mapsto wb}|) \right)}{\left(\left(1 + \frac{|a > wb|}{|a > wc|} \right) \times (|a > wb| - |[a > wc]_{a \mapsto wb}|) + 1 \right)} \quad (5)$$

Another important point of heuristic (5) presented above is that as more events are available in the log greater is the confidence of obsolete relations. This behavior is introduced by the partial expression $1 + (|a > wb| / |a > wc|)$ presented in (5). It is used as a factor to maximize the value calculated by $a \Rightarrow wb \mapsto c$.

Considering the log with modified instances (i.e. log generated from the execution of a modified source code) $W = \{..., BR1BR2BR4BR6BR7^{18}, BR1BR2BR5^{18}\}$, presented in Fig. 1-c, we can check that the task BR3 was replaced by BR4. All dependency relations in the dependency graph which have task BR3 as a causal relation is likely to be an obsolete relation. In the example, the relation BR2→BR3 is the only relation having task BR3 in the causal relation. To confirm if it is an obsolete relation, we need to analyze it from the sibling relations perspective (i.e. BR2 \mapsto BR4 e BR2 \mapsto BR5). Thus, we apply the heuristic (5) for each of them. The first sibling relation BR2→BR4 has the support of $a > wb$ as the partial number of times that BR2→BR4 occurs in the log (i.e. 23 times), the support of $a > wc$ as the partial number of times that BR2→BR3 occurs in the log (i.e. 6 times), and $[a > wc]_{a \mapsto wb}$ as the support of sibling relation BR2 \mapsto BR4 before the last occurrence of BR2→BR3. Fig. 9-a shows that the support of sibling relation BR2 \mapsto BR4 was zero (i.e. no occurrences) in the last time that BR2→BR3 occurs. After that, we can calculate the confidence of BR2→BR3 to be an obsolete relation against BR2 \mapsto BR4 as $a \Rightarrow wb \mapsto c = ((1 + 23 / 6) \times (23 - 0)) / (((1 + 23 / 6) \times (23 - 0)) + 1) = 0.991$. We repeated the same process to the next sibling relation

$BR2 \mapsto BR5$, where $a \Rightarrow wb \mapsto c = ((1 + 28 / 6) \times (28 - 0)) / (((1 + 28 / 6) \times (28 - 0)) + 1) = 0.993$. As result of the execution of *ComputeOldSiblingRelation* for both sibling relations $BR2 \mapsto BR4$ and $BR2 \mapsto BR5$ we have obtained values 0.991 and 0.993, respectively.

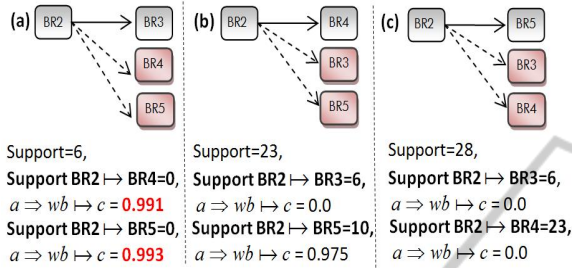


Figure 9 : Calculating the candidate obsolete relations.

To check if the relation above (i.e. $BR2 \mapsto BR3$) can really be defined as an obsolete dependency relation, we execute the algorithm 4 (i.e. see *CheckOldSiblingRelation* at item 2.a.v of *IncrementalMiner*). The algorithm checks the confidence of each sibling relation $\mapsto w$ of dependency relation $> w$ calculated by *ComputeOldSiblingRelation*. To set the minimum acceptable value calculated by the heuristic (5), the threshold *Obsolete Relation* was created and used by the following definition:

Definition 1 (Obsolete Relation). Let $> w$ be a relation with two or more sibling relations $\mapsto w$, a dependency relation is considered obsolete if and only if all sibling relations have $a \Rightarrow wb \mapsto c$ result above threshold *Obsolete Relation*.

The definition above is followed by algorithm 4, as presented at item 5.c and 5.d. Moreover, we set the threshold *Obsolete Relation* value to 0.990 to get only obsolete relation candidates with high confidence. So, if all sibling relations $\mapsto w$ of $> w$ reach heuristic results above threshold, then we need to remove $> w$ from dependency process graph, as presented in items 5.d and 5.e.ii of algorithm 6. Back to the example, we can see that relation $BR2 \mapsto BR3$ presented in Fig. 9-a has both of sibling relations with heuristic result above 0.990 (i.e. 0.991 for $BR2 \mapsto BR4$ and 0.993 for $BR2 \mapsto BR5$). Thus, it means that we need to remove the relation from dependency graph, such as presented in the final discovered model in Fig. 5-c.

4 EXPERIMENTS

The experiments in this section were implemented in Java language and divided into two groups. The first group shows the quality of models mined from logs generated by a process execution simulator (i.e. see section 4.1). The second group demonstrates the quality of models mined from logs of a real legacy system.

4.1 Experiments on Simulated Data

Obtaining practical data for incremental mining is not a trivial task. Therefore, we have used a simulation tool (Burattin, 2010) to generate data about process models definitions and their execution logs. The models are generated in a recursive way. First n parts are generated. Each part is a task (with probability 50%), a parallel structure (20%), an alternative structure (20%) or a loop (10%). We also included noise in 5% of all traces in the log. Additionally, each task has a performer that represents a process participant. For a parallel or an alternative structure the simulation randomly generates b branches. Usually there are no more than 100 tasks in a workflow model (Weijters et al, 2006), so we set $n = 4$ and $2 \leq b \leq 4$ to limit the scale. Each model has at least one loop and at most three alternative structures and at most three parallel structures. The simulation randomly generates each task's waiting time and execution time. At the choice point it enters each branch with the same probability. Each generated model has also a modified version. It was used to simulate the evolution of the process model and to perform the incremental mining with modified process instances. At the end, we generated 400 models (i.e. 200 original process models and 200 modified versions of them) and 200 log files, with an average of 47.6 tasks. In each log dataset there are 500 simulated instances made up by 300 new process instances (i.e. from original process model) and 200 modified process instances, generated from the modified version of these processes.

4.2 Quality of Non Incremental Mining

For measuring the correctness (i.e. accuracy) of our method in a non-incremental scenario, we have used the conformance checking metrics for models and logs (Rozinat, 2007). The result is evaluated from aspects of *Token Fitness* (i.e. which evaluates the extent to which the workflow traces can be associated with valid execution paths specified in the

model), *Behavioral Appropriateness* (i.e. which evaluates how much behavior is allowed by the model but is never observed in the log) and *Structural Appropriateness* (i.e. which evaluates the degree of clarity of the model). We use the conformance checker plug-in of ProM 5.2 (van Dongen, 2005). The average results from the mining of 200 datasets (i.e. considering just original process instances without changes from the logs) are shown in Table 2.

The Token Fitness and the Structural Fitness metric values suggest that our method has nearly the same precision as α -algorithm (van der Aalst, 2004) and Behavioral Appropriateness slightly lower than α -algorithm and the method proposed by Ma et al (Ma, 2011).

Table 2: Quality metrics values.

| Metric | Our Method | α -algorithm | Ma et al [5] |
|--------------------|------------|---------------------|--------------|
| Token Fitness | 0.998 | 0.882 | 0.953 |
| B. Appropriateness | 0.851 | 0.865 | 0.854 |
| Structural Fitness | 1.000 | 1.000 | 0.901 |

4.3 Quality of Incremental Mining

Because of a lack of techniques to measure the models conformance during the incremental mining with modified process instances, it was necessary to use an alternative technique. Kappa (Cohen, 1960) was used to give a quantitative measure of agreement between the input process models which generate the log records (i.e. observer 1) and the process model mined by the *IncrementalMiner* (i.e. observer 2). Here, this measure of agreement defines the level of similarity between the process structures of both models. The reason to use Kappa to check process graph similarity instead other methods like (Dijkman, 2009) is that we must also consider the organizational aspects of process such as participants of process, which is not part of the graph structure.

The values obtained from Kappa range from -1 (i.e. complete disagreement or low similarity) to +1 (i.e. perfect agreement or full similarity). The statistic formula is presented by equation (6), where $P(A)$ is the empirical probability of agreement between two observers for one aspect of the process, and $P(E)$ is the probability of agreement between two observers who performed the classification of that aspect randomly (i.e. with the observed empirical frequency of each mapping aspect). Thus, high Kappa values suggest high similarity between the input and output models. In this work, the

agreement measure is applied to six different aspects, as shown in Table 3.

$$K = \frac{P(A) - P(E)}{1 - P(E)} \quad (6)$$

The first aspect checks the mapping of activities in the model. It identifies whether all activities presented in the mined models belongs to the process definition and are also arranged in the process graph appropriately. The second aspect refers to the mapping of participants to the tasks of the process. It defines whether the participant associated with a task actually performs the activity in the process. The next two aspects define the mapping of incoming and outgoing transitions of an activity. These aspects define if all input and output transitions associated with an activity are correct. The last two aspects evaluate whether the semantics of input and output activities (e.g. AND/XOR-split/join) are correct. So, we can check whether control flows associated to the process are correct.

Table 3: Kappa results for real and simulated data.

| Aspect | Precision with Simulated Logs | Precision with Real Legacy System Logs |
|----------------------|-------------------------------|--|
| Activity Mapping | 1.000 | 0.920 |
| Participants Mapping | 0.710 | 1.000 |
| Input Transitions | 0.950 | 0.930 |
| Output Transitions | 0.910 | 0.900 |
| Output Semantics | 1.000 | 1.000 |
| Input Semantics | 1.000 | 1.000 |

The Kappa verification of models is divided into three main steps which are represented in Fig. 10. First, we perform the mining of initial log (i.e. new traces without changes, generated by simulator, as described in Section 4.1) through *IncrementalMiner* to generate the intermediate models. After that, we submit to *IncrementalMiner* both intermediate model (i.e. generated in first step) and the log with modified instances of processes, also generated by simulator. The result is a set of updated process models containing all new dependencies and the updated ones. As the last step, we submit both modified process definition and updated discovered process model to Kappa for similarity verification. In column *Precision with Simulated Logs* of Table 3, we can check the results. We have obtained high values for Kappa in the majority of aspects (i.e. values above 0.8 are considered good agreement (Landis, 1977)), what suggests that the discovered models and the designed models that generate the logs are similar. This means that the incremental mining of logs was conducted efficiently for the main aspects.

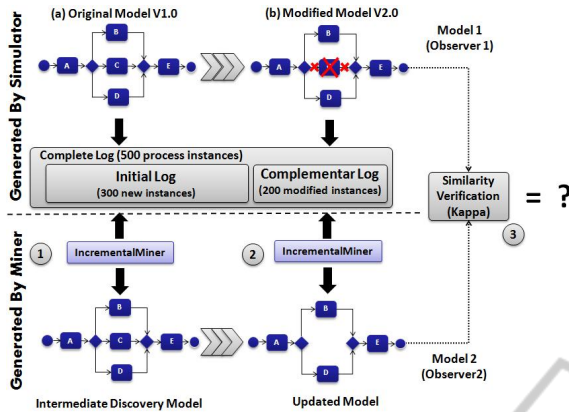


Figure 10: Incremental Mining Verification Process.

4.4 Experiments on Real Legacy System

To demonstrate how effective this approach on a real scenario is, we use a real legacy system. So, the process logs were generated from successive executions of an ERP legacy system written in COBOL language. It has more than 2,000,000 lines of source code and several modules (i.e. Financial Management, Sales Management, etc). Moreover, each module implements several implicit (i.e. no formal process models defined) and interrelated business processes, represented in source code as business rules, and illustrated in Fig. 11.

In order to start the discovery of business processes from legacy system, we followed the process defined in Fig. 4. On this experiment each module of legacy system was annotated and recompiled to generate trace events to the log. They were instrumented in such a way that each executed business rule (i.e. task) records an event into the log. Moreover, use case scenarios were defined to coordinate both the system execution and the log generation.

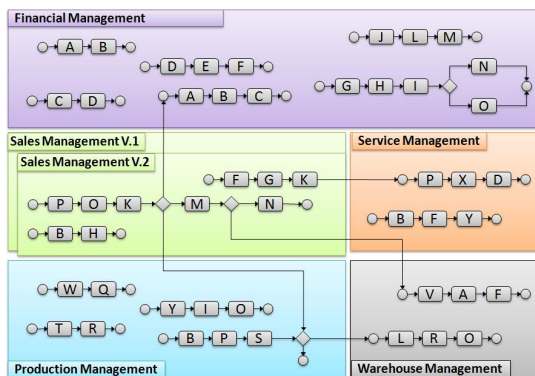


Figure 11: Legacy ERP system and related modules.

We have split and executed the user scenarios in seven groups. Each group represented all the scenarios related to one specific user of legacy system. Following the successive execution of these system scenarios, seven incremental dataset (i.e. one per user) with approximately 30,000 trace instances were generated. The datasets were named respectively as A, B, C, D, E, F and G. The datasets A, B recorded incremental logs related to execution of Financial Management module. Therefore, dataset C recorded process instances from execution of Sales Management. Dataset D recorded process instances of Service Management module. The next two datasets E and F recorded process instances of modules Production Management and Warehouse Management, respectively. The last dataset (i.e. G) perhaps introduce modified process instances, generated from execution of modified version of Sales Management module (i.e. version 2 as illustrated in Fig. 11).

Table 4: Elements Extracted from Legacy.

| Element | Log file | | | | | | Total | |
|--------------------------|----------|----|-----|-----|-----|----|---------|----|
| | A | B | C | D | E | F | | |
| Biz Processes Structures | +2 | +1 | +2 | +1 | +4 | +3 | - | 13 |
| System Participants | +3 | +1 | +1 | +1 | +1 | +1 | - | 8 |
| Human Participants | +1 | +1 | +1 | +1 | +1 | +1 | +1 | 7 |
| Tasks (Business Rules) | +20 | +8 | +15 | +11 | +24 | +1 | +2 (-4) | 77 |
| XOR-split/join | +10 | +5 | +5 | +1 | +3 | +3 | +3 (-2) | 28 |
| AND-split/join | +1 | | | | | | | 1 |

The results can be seen in Table 4. It shows the complete list of elements extracted from the incremental mining of execution logs. All elements listed are part of business process model structure. Incremental mining reveals new elements after each mined dataset of instances, gradually generating a more complete model. On the other hand, the last dataset log (i.e. G) reveal obsolete elements that were removed from process models (i.e. see negative values on rows Tasks and XOR-split/join). Thus, these results can demonstrate that we can extract process models from legacy in an incremental way even on those situations where the system have to be modified during the mining process.

To measure the quality of models extracted from legacy, we have also used Kappa statistic. The same aspects considered on Section 4.3 and shown on Table 3 were used. Here, these aspects were used to demonstrate the level of business analysts agreement on the business process structures obtained from legacy. So, in experiments using two different business analysts, Kappa values between 0.90 and 1.00 were obtained, as shown in column *Precision*

with *Real Legacy System Logs* of Table 3. That means the business analysts agree with most of process models structures mined from legacy, during the incremental discovery.

5 RELATED WORK

Gunther (Gunther et al, 2008) introduced the mining of ad-hoc process changes in adaptive Process Management Systems (PMS). This technique introduces extension events in the log (e.g. insert task event, remove task event, etc) that record all changes in a process instance. So change logs must be interpreted as emerging sequences of activities which are taken from a set of change operations. It is different from conventional execution logs where the log content describes the execution of a defined process. The problem here is that sometimes legacy information systems and WfMS do not generate process change information into the log. Thus, it is very hard to discovery process changes from these systems. Bose (Bose et al, 2011) although, introduced concept drift applied to mining processes. He applied techniques for detection, location and classification of process modifications directly in the implementation log without the need for specialized log containing such modifications as proposed by Gunter. After that, Luengo (Luengo et al, 2012) proposed a new approach using clustering techniques. He uses the starting time of each process instance as an additional feature to those considered in traditional clustering approaches.

The work presented here supports the main operations of evolutionary learning (e.g. insert and exclusion operations) using an incremental mining approach. Moreover, our technique does not require extra information in the log to detect process changes. Thus, it makes possible to avoid the reprocessing of the complete set of logs, reducing its total processing time.

6 SUMMARY AND OUTLOOK

This paper proposed an incremental process mining algorithm for mining of process structures in an evolutionary way from legacy systems. This is an important step for the incremental legacy modernization because it keeps the system maintenance live while the system is modernized. The algorithm enables the discovery of new and obsolete relations from log as new or modified

traces are executed and recorded in the log. Thus, we can keep all process models discovered updated with the process definition when it changes.

In quality experiments using non-incremental simulated data and conformance metrics, the models discovered by IncrementalMiner present good accuracy. Regarding the incremental approach, IncrementalMiner also shows good precision for the models discovered from logs with modified process instances. During the discovery of process models from real legacy system and also simulated logs, the algorithm shows good results on the extracted models (i.e. Kappa values above 0.900). Thus, our approach could be an effective alternative for incremental mining of process models during the re-engineering of legacy systems.

Altogether, the main contribution of this work was the creation of a mechanism that introduces the incremental mining of logs with support to i) the discovery of new dependency relations (i.e. new tasks) and participants in order to complement a partial or complete process model, and ii) the identification and removal of obsolete dependency relations in order to update an existent process model. We also introduced an alternative way to measure the quality of models generated during the incremental process mining.

As future work we include the improvement of the identification of obsolete participants in the model (i.e. see low kappa value in simulate data of Table 3) and the integration of algorithm and the incremental approach in ProM tool.

REFERENCES

- Kalsing, A. C., Iochpe, C., Thom, L. H.: "An Incremental Process Mining Algorithm". In *ICEIS (1)*, pp. 263-268 (2010).
- Kalsing, A. C., Nascimento, G. S., Iochpe, C., Thom, L. H.: "An Incremental Process Mining Approach to Extract Knowledge from Legacy Systems". In *14th IEEE EDOC 2010*, pp. 79-88 (2010b).
- Cohen, J., "A coefficient of agreement for nominal scales, Educational and Psychological Measurement". Vol. 20 (1), pp. 37-46 (1960)
- Van Dongen, J., Medeiros, A. K., Verbeek, H., Weijters, A. J. M. M., van der AALST, W. M. P.: "The ProM framework: A new era in process mining tool support". *Applications and Theory of Petri Nets*, pp. 1105-1116, Springer (2005)
- Ma, H., Tang, Y., Wu, L.: "Incremental Mining of Processes with Loops". *International Journal on Artificial Intelligence Tools*, Vol. 20, Number 1, pp. 221-235 (2011)

- Sun, W., Li, T., Peng, W. Sun, T.: "Incremental Workflow Mining with Optional Patterns and Its Application to Production Printing Process". *International Journal of Intelligent Control and Systems*, Vol. 12, Number 1, pp. 45-55 (2007)
- 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), pp. 1128-1142 (2004)
- Gunther, C.W., Rinderle-Ma, S., Reichert, M., Van der Aalst, W.M.P.: "Using process mining to learn from process changes in evolutionary systems. *International Business Process Integration and Management*", Vol. 3, N 1, pp. 61-78, Inderscience (2008)
- Weijters, A. J. M. M., Van der Aalst, W.M.P., Medeiros, A. K.: "Process Mining with the Heuristics Miner Algorithm". Eindhoven, Tech. Rep. WP, vol. 166 (2006).
- Rozinat, A., Medeiros, A. K., Gunther, C.W., Weijters, A. J. M. M., Van der Aalst, W.M.P.: "Towards an evaluation framework for process mining algorithms". *BPM Center Report BPM-07-06*, BPMcenter. Org (2007)
- Burattin, A., Sperduti, A.: "PLG: a Framework for the Generation of Business Process Models and their Execution Logs". In *Proceeding BPI Workshop 2010*. Stevens Institute of Technology; Hoboken, New Jersey, USA; September 13 (2010)
- Dijkman, R., Dumas, M., García-Bañuelos, L.: "Graph matching algorithms for business process model similarity search". *Business Process Management Journal*, pp. 48-63, Springer (2009)
- Sneed, H. M., Erdos K., "Extracting business rules from source code," *Proceeding of the Fourth IEEE Workshop on Program Comprehension*, pp. 240-247, 1996.
- Wang C., Zhou, Y., Chen, J., "Extracting Prime Business Rules from Large Legacy System," *International Conference on Computer Science and Soft. Engineering*, vol 2, pp. 19-23, 2008.
- Bose, R. P., Van der Aalst, W. M. P., Zliobaité, I., Pechenizkiy, M.: "Handling concept drift in process mining". *Conference on Advanced Information Systems Engineering, CAISE'11*, pp 391-405. London, UK, 2011.
- Landis, J. R., Koch, G. G., "The measurement of observer agreement for categorical data," *International Biometric Society*, vol. 33. pp. 159-174, 1977.
- Luengo, D., Sepúlveda, M., "Applying clustering in process mining to find different versions of a business process that changes over time", *Business Process Management Workshops*, pp 153-158, Springer, 2012.