Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2018) 26: 1314 – 1328

doi:10.3906/elk-1708-112

Interactive process miner: a new approach for process mining

İsmail YÜREK[1,*], **Derya BİRANT**[2], **Kökten Ulaş BİRANT**[2]

[1]Graduate School of Natural and Applied Sciences, Dokuz Eylül University, İzmir, Turkey

[2]Department of Computer Engineering, Faculty of Engineering, Dokuz Eylül University, İzmir, Turkey

Received: 10.08.2017 • **Accepted/Published Online:** 03.04.2018 • **Final Version:** 30.05.2018

Abstract: Process mining is a technique for extracting knowledge from event logs recorded by an information system. In the process discovery phase of process mining, a process model is constructed to represent the business processes systematically and to give a general opinion about the progressive of processes in the event log. The constructed process model can be very complex as a result of structured and unstructured processes recorded in real life. Previous studies proposed different approaches to filter or eliminate some processes from the model to simplify it by implementing some statistical or mathematical formulas rather than user interactions. The main objective of this study is to develop an algorithm that is capable of working on large volume of event logs and handling the execution records of running process instances to analyze the execution of processes. The other significant principle is to provide an interactive method to ensure the decisions that will be taken to improve the execution of processes by verifying in a simulation environment before being put into practice. This study proposes a novel algorithm, named interactive process miner, to create a process model based on event logs and a new approach that contains three different features, including activity deletion, aggregation, and addition operations on the existing process model. The experimental results show a fundamental improvement in performance compared to the existing algorithms. As a result of this study, users will have an opportunity to analyze a large volume of event logs in a short time with low memory usage and to modify the created process model to observe the impact of improvement changes in a simulation environment before applying any changes to a system in real life.

Key words: Process mining, process model, pattern discovery

1. Introduction

Process mining is a technical way to acquire information in order to analyze, discover, improve, and manage the processes from event logs that contain elaborative materials about the history of business operations. Process mining provides an important opportunity to detect process bottlenecks of a system. It is possible to control, manage, and fix all issues after detecting the weak points of the system by applying process mining techniques. A process model represents the dependencies between activities of the process and all the information about them without categorizing which ones are important or not. This situation makes the process model hard to understand and interpret for people.

As time progresses, event logs grow rapidly and create a large volume of data. This large data increase scales up the discovery time and causes performance problems. In general, current process mining techniques analyze historical data. Incorporation of ongoing or newly completed process records is of major importance in terms of keeping the process model constantly up-to-date. Updating the process model instantly will provide the ability to see the problems in the progressive process records immediately.

*Correspondence: ismail.yurek@gmail.com

Our goal is to develop a process mining algorithm that is able to work on a large volume of event logs and incorporate the execution records of ongoing processes into the discovered process model instantly. The proposed algorithm supports different operations such as adding, deleting, and aggregating activities on the process model to provide an interactive environment that reveals the impacts of improvement changes before applying the decisions in real life.

The main contributions of this paper can be summarized as three-fold: firstly, a novel algorithm, named interactive process miner (IPM), is proposed to create a process model based on a large volume of event logs; secondly, it also proposes three new features for process mining, namely process addition, deletion, and aggregation; lastly, it demonstrates the IPM algorithm on both real-life and experimental datasets to show that it performs better than existing algorithms in terms of running time and precision rate.

2. Related works

The concept of process mining started to come up at the end of the 90s. In 1998, Agrawal et al. [1] proposed a new approach that deals with noise and parallel structure to extend the utility of actual workflow systems. Their approach allows the user to use existing event logs to model a given business process as a graph. After that, Cook and Wolf [2] described different methods for process discovery and to produce formal models based on the actual process executions.

In 2004, van der Aalst et al. [3] introduced the α-algorithm, which is able to discover a large and relevant class of workflow processes. At first the α-algorithm analyzes the event log, and then constructs various dependency relations between tasks. The aim is to analyze different kinds of workflow logs in the presence of noise and without any knowledge of the underlying process. In the same year, Cook and Wolf [4] worked on discovering concurrent models of system behavior from event traces by using probabilistic techniques.

In 2004, Herbst and Karagiannis [5] dealt with duplicate tasks and they proposed an algorithm based on an inductive approach in two steps: (i) induction and (ii) stochastic task graph (SAG) generation. The SAG is then transformed into a blocked structured model using a definition language. They developed a tool, called InWoLvE, which takes many parameters; however, it is necessary to give proper parameters to improve mining efficiency and quality.

In 2004, Schimm [6] proposed an approach to extract an accurate model from event logs and to deal with hierarchically structured workflow models that include the splits and joins. He demonstrated his method by an example and also developed a tool for process mining.

In 2005, Dongen and van der Aalst [7] defined a standard for storing event logs. They introduced a data model and an XML format called MXML (Mining eXtensible Markup Language). In the same year, Wainer et al. [8] proposed an approach based on process algebra to derive a final model from the existing models. The main idea is to rewrite the existing models with new instances. However, it is complex in implementation and it has both high time and large space complexities.

In 2006, Weijters et al. [9] proposed the HeuristicsMiner algorithm that discovers main behavior registered in a noisy event log. The algorithm includes different threshold parameters in order to overcome two problems: noise and low frequency behavior.

In 2007, Günther and van der Aalst [10] emphasized existing problems in the traditional process mining techniques when the processes are large and less-structured. To handle the problems, they developed a flexible approach based on their previous works [11,12]: fuzzy mining. Their approach adaptively analyzes, simplifies, and visualizes mined process models based on two metrics: significance and correlation of graph elements.

Medeiros et al. [13,14] used a genetic algorithm to mine process models in the ProM (process mining) framework and performed experiments on the simulated data. Their results showed that the genetic algorithm found all possible business process models that could parse all the traces in the event log. However, time and space complexity is the main disadvantage of the genetic approach. For this reason, in 2010, Bratosin et al. [15] proposed a distributed genetic approach to overcome the high computational problem of the genetic approach.

In 2008, Song et al. [16] used simulated annealing in business process mining. Song et al. [17] have also proposed a novel approach, trace clustering, in which the event log is split into homogeneous subsets and for each subset a process model is created.

In 2011, Bose et al. [18] proposed features and statistical techniques to detect changes and to identify changed regions from a control-flow perspective. Luengo and Sepulveda [19] extended the work [18] by adding time feature and the clusters that formed by sharing both a structural similarity and a temporal proximity. Van der Aalst et al. [20] provided a configurable approach to predict the completion time of the process by constructing a process model with time information. They used an annotated transition system to predict the remaining flow time of all or some of the process instances.

In 2012, van der Aalst [21] emphasized process mining as a hot topic in business process management. Three basic types of process mining (discovery, conformance checking, and enhancement) were presented using a small example and some larger examples were given to illustrate the applicability in real-life settings. Our study described in this paper focuses on the discovery type of process mining. Van der Aalst and his team developed three workflow/process mining tools: Little Thumb, EMiT (enhanced mining tool), and ProM (process mining). Little Thumb can extract workflow nets from noisy and incomplete logs [12]. EMiT can convey workflow models with Petri nets [22]. ProM is a generic open-source framework for implementing process mining projects that includes many packages with many plug-ins [23].

In 2013, Fahland and van der Aalst [24] presented a postprocessing approach to control the balance between overfitting and underfitting by simplifying discovered process models. They expressed the discovered process model in a Petri net, and their approach can be combined with any process discovery method that generates a Petri net.

In 2015, Aleem et al. [25] presented the comparison of different process mining approaches in detail. The important point of their paper is that it collects and shows all efficient and qualitative results of business process mining for researchers. Their article groups the process mining approaches into five sections: deterministic, heuristic, inductive, genetic, and clustering-based mining approaches. Cheng and Kumar [26] proposed a technique to remove noisy traces from event logs by building a classifier and applying classifier rules on event logs. They showed that generated mined models from such preprocessed logs are superior on several evaluation metrics. Fahland and van der Aalst [27] investigated the problem of repairing discovered process models to align them to reality. They decomposed the event log into several sublogs of nonfitting traces to make conformance checking. Rovani et al. [28] presented a methodology in order to analyze medical treatment processes by showing how to apply process mining techniques based on declarative models.

In 2016, de Leoni at al. [29] proposed a framework to unify a number of approaches for correlation analysis. They tried to correlate different process characteristics related to different perspectives. Mannhardt et al. [30] proposed a process mining algorithm to check process conformance with respect to control flow, data dependencies, resource assignments, and time constraints. Pika et al. [31] presented an approach and a supporting tool to evaluate the overall risk of the process and to predict process outcomes. The approach is

based on the analysis of information about process executions recorded in event logs. Tax et al. [32] suggested an algorithm named local process model to discover frequent behavioral patterns in event logs. The algorithm focuses on local structures to enable process mining of noisy event logs and extends sequential pattern mining techniques. Bolt et al. [33] came up with a framework to make process mining repeatable and automated for event logs that may need to be decomposed and distributed for analysis. They stated the main motivation of their study is the inability to model and execute process mining workflows. Their study establishes the basic building blocks required for process mining and also describes various analysis scenarios to show the feasibility of their approach.

In 2017, Suriadi et al. [34] described a set of data quality issues and presented a patterns-based approach to clean noisy event logs. Mitsyuk et al. [35] suggested a tool to generate event logs from Business Process Model and Notation (BPMN) and they implemented script-based gateways and choice preferences to manage control flow. Bolt et al. [36] proposed an approach to address the problem of comparing different variants of the same process and to detect differences in behavior and business rules. They used transition systems that were annotated with measurements to model behavior and to underline differences.

In 2018, Alizadeh et al. [37] recommended an approach to enable the identification of deviations by reconciling the data and process perspectives. They linked data and control flow for conformance checking. In their study it is stated that the proposed approach is capable of identifying deviations in both data usage and control flow, while providing the purpose and context of the identified deviations.

Differently from the previous studies, this paper proposes a novel algorithm to create a process model based on a large volume of event logs and to handle the execution records of running process instances in a short time with low memory usage, and also provides three new features (addition, deletion, and aggregation) to support an interactive environment for process mining.

3. Interactive process miner

This paper proposes an algorithm named interactive process miner (IPM) for process mining. Process mining is a method to discover information from event logs. In process discovery, a process model is constructed to represent the processes based on observed events. Before constructing a process model, some preliminary preparations are performed such as tree construction and dependency matrix creation. The first one is creating a tree that includes all traces in the event log. Figure 1a shows an example tree structure constructed from the event log that contains 14 traces listed in Table 1. The constructed tree is used to create the dependency graph of the process model as shown in Figure 1b. The line in the event log is read and a node is added for each nonexistent activity in the current trace. The edge count, which represents the relation between two activities, is increased each time.

The constructed tree is used as input to create a dependency matrix. To form a dependency matrix, the algorithm starts from the root activity and continues traversing the tree. While traversing the tree, the goal is to reach the leaf activity. When all leaf activities are reached, it means that the constructed tree represents the complete event log, because it shows all the traces that are visited so far. While traversing all the activities, the dependency matrix is created one by one for each activity. Figure 2a shows the dependency matrix that is created by reading the event log. The other versions of the dependency matrix, which are modified by activity deletion, aggregation, and addition operations, are shown in Figure 2b, Figure 2c, and Figure 2d, respectively.

Let us start traversing the tree given in our example to construct a dependency matrix. In our example, there are 4 leaf activities: L_0, L_1, L_2, and L_3. The algorithm starts from the root activity A, and traverses all the paths to reach the leaf activities. For instance, when the algorithm reaches the first leaf L_0, the followed

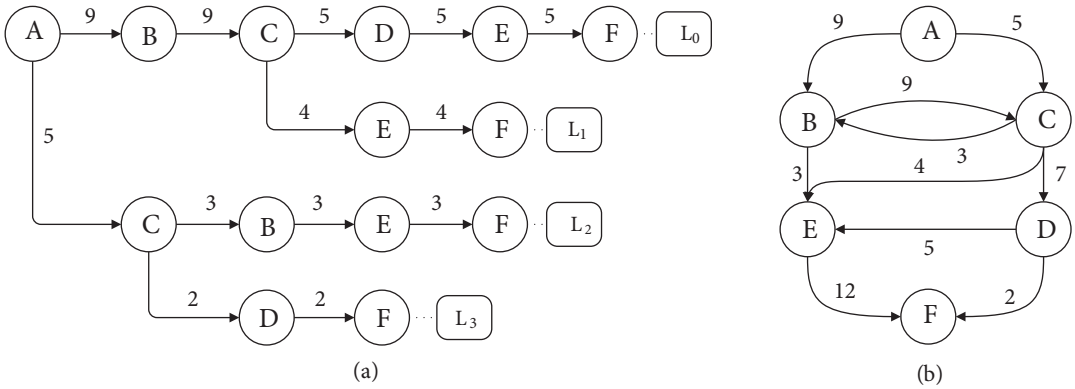

Figure 1. (a) Data model of all traces in the event log. (b) Dependency graph representation of process model.

Table 1. Sample event log.

Trace ID	Events
Trace 1	ABCDEF
Trace 2	ABCEF
Trace 3	ACBEF
Trace 4	ACDF
Trace 5	ABCDEF
Trace 6	ABCDEF
Trace 7	ACBEF
Trace 8	ABCEF
Trace 9	ACBEF
Trace 10	ABCEF
Trace 11	ABCDEF
Trace 12	ACDF
Trace 13	ABCEF
Trace 14	ABCDEF

activities are $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F$. The current question is how will we know how many times this trace exists in the event log? The answer is the last edge count of the trace. In other words, the total number of the trace in the event log is equal to the sum of last edge count of the leaf activities. After that, the number of dependencies is written to the related cell of the dependency matrix.

After visiting the first leaf L_0, again the algorithm starts from the root activity A. This time it follows a different path to reach the second leaf activity L_1. The same logic is followed until the last leaf activity. The final version of dependency matrix keeps all the relations between all activities in the complete event log.

According to the dependency matrix, each dependency between activities is read from left to right. For instance, for the 1st row and 2nd column of the matrix, it stores the count of dependency between A and B such as $A \rightarrow B = 9$. Finally, we can get all the dependency counts from the matrix as follows: $A \rightarrow B = 9$, $A \rightarrow C = 5$, $B \rightarrow C = 9$, $B \rightarrow E = 3$, $C \rightarrow B = 3$, $C \rightarrow D = 7$, $C \rightarrow E = 4$, $D \rightarrow E = 5$, $D \rightarrow F = 2$, $E \rightarrow F = 12$. By this way, it is possible to know all the dependencies and the count of them between activities.

	A	B	C	D	E	F
A	0	9	5	0	0	0
B	0	0	9	0	3	0
C	0	3	0	7	4	0
D	0	0	0	0	5	2
E	0	0	0	0	0	12
F	0	0	0	0	0	0

(a)

	A	B	C	E	F
A	0	4	3	0	0
B	0	0	4	3	0
C	0	3	0	4	0
E	0	0	0	0	7
F	0	0	0	0	0

(b)

	A	C	E	F	X
A	0	5	0	0	9
C	0	0	4	0	10
E	0	0	0	12	0
F	0	0	0	0	0
X	0	9	8	2	0

(c)

	A	B	C	D	E	F	X
A	0	9	5	0	0	0	0
B	0	0	9	0	3	0	0
C	0	0	0	7	0	0	7
D	0	0	0	0	5	2	0
E	0	0	0	0	0	12	0
F	0	0	0	0	0	0	0
X	0	3	0	0	4	0	0

(d)

Figure 2. (a) Dependency matrix. (b) Dependency matrix after activity deletion. (c) Dependency matrix after activity aggregation. (d) Dependency matrix after activity addition.

The constructed process model contains all the activities in the event log. Showing all traces in the event log in a process model makes the model complex and hard to understand. It is possible to consider the frequency of traces in the log to reduce the complexity of the process model to make the model easy to understand and interpret. Traces are eliminated from the process model based on a defined threshold. This threshold is a user-defined value. The user can set this threshold to any value between 0 and 1 in the interactive environment. Figure 3 shows how the IPM algorithm works step-by-step.

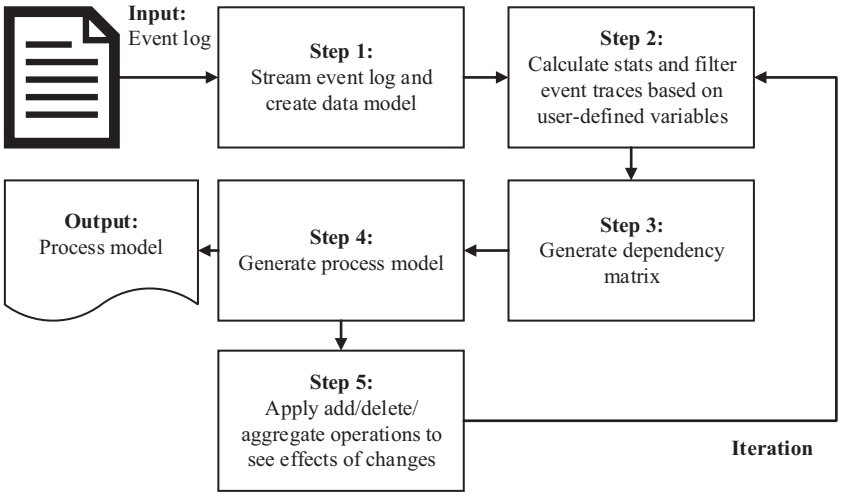

Figure 3. Block diagram of IPM algorithm.

4. Provided approaches for process model modification

Up to this time, we presented all traces in the event log with a constructed process model. With the help of the process model, it is now possible to have a general opinion about the progressive of processes in the event log and we have a chance to see the differences or outliers between the process model we designed and what actually happened in real life. Still, finding answers to some questions is very hard at this stage. For example, what happens if activity X is removed from the process model? Or how does the process model look if activities X and Y are aggregated under another event? Or how is the process model affected if a new activity Z is added between the activities X and Y? If we know the answers to these questions, we will have an important opportunity to see the effects of any changes to a real-life system so as to improve it. Considering this motivation, we provide three new features for process mining: process addition, deletion, and aggregation.

Available researches put different approaches forward to show an activity under another superior activity or to aggregate different activities whose frequency value is below a certain threshold. However, these operations are done based on some statistical values rather than user interactions. The user who is looking for answers to the above questions should be able to modify the model interactively.

4.1. Activity deletion

Picking an activity and removing it from process model is an activity deletion operation. This section explains our new approach, which is applied to the same example event log listed in Table 1.

Let us proceed with the logic behind the algorithm step by step. For instance, what happens if the user wants activity D to be deleted from the model? The algorithm starts to traverse from the root activity and tries to reach the leaf activities. However, each time it checks if the coming activity is D or not. If not, it continues to traverse and adds this path to the dependency matrix; otherwise it stops and deletes the tracked path. In other words, the algorithm does not add the path that includes deleted activity to the data model.

Step 1: The algorithm tries to reach L_0, but it faces activity D in the path, stops, and removes the related path.

Step 2: Secondly, it tries to reach the leaf L_1 and get there. It means that this path will remain in the process model. Whenever the algorithm reaches a leaf activity, it adds this path to the dependency matrix.

Step 3: It tries to reach L_2 and again get there. Therefore, the third path will remain in the process model and it should be added to the dependency matrix.

Step 4: Finally, it tries to reach L_3 leaf activity, but it encounters activity D in the path, stops, and removes the related path. At the end of this step, the algorithm traversed all the leaf activities and the final version of the data model is created as shown in Figure 4a. According to the values in the dependency matrix, a dependency graph that represents the new process model after deletion operation is created as shown in Figure 4b.

4.2. Activity aggregation

In this section, we are looking for an answer to the question how does the process model look if *A1* and *A2* activities are aggregated under another event such as *A3*. At least two activities can be combined and represented as another activity. In the same example, assume that the user picks activities B and D to represent them under another activity named X. The algorithm starts to traverse from root activity and tries to find activities B and D to represent as X until reaching the leaf activity in the followed path.

Step 1: The data model starts to change while the algorithm is traversing all the activities. Firstly, the algorithm tries to reach L_0. Each time, it checks if the coming activity is B or D. If the activity is one of

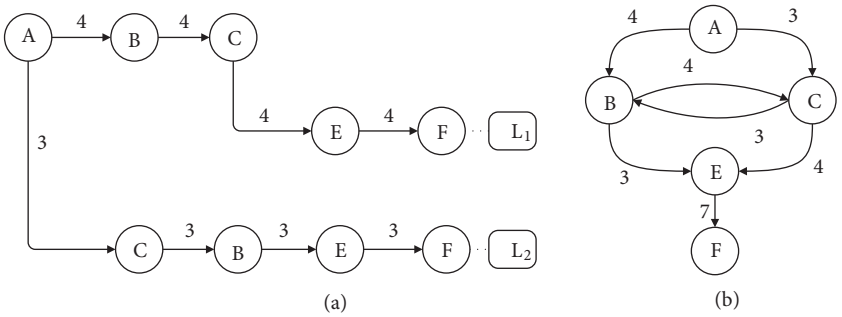

Figure 4. (a) Data model after activity deletion. (b) Dependency graph representation of new process model after activity deletion.

them, then it replaces the current activity with X such as $A \rightarrow X \rightarrow C \rightarrow X \rightarrow E \rightarrow F$ for leaf L_0. After finding the first leaf, it is added to the dependency matrix. In this example, we try to represent B and D as X, and so there is no need to add the activities B and D to the dependency matrix. Instead of B and D, activity X should be added.

Step 2: In the second step, it tries to reach L_1. It controls if the coming activity is B or D. If the activity is one of them, then it replaces the current activity with X such as $A \rightarrow X \rightarrow C \rightarrow E \rightarrow F$ for leaf L_1.

Step 3: While the algorithm tries to reach leaf L_2, it checks the activities one by one, whether they are B or D. If the answer is true, then it replaces the current activity with X such as $A \rightarrow C \rightarrow X \rightarrow E \rightarrow F$ for leaf L_2.

Step 4: Lastly, the algorithm tries to reach L_3. If it faces one of the aggregated activities, then it replaces the current activity with new one such as $A \rightarrow C \rightarrow X \rightarrow F$ for leaf L_3.

Figure 5a shows the final version of the data model. From the dependency matrix, it is clearly seen that A is the root activity, because all the values of the cells are equal to zero vertically. This means that there is no activity input to A. In addition, it is seen that F is leaf activity, because all the values of the cells are equal to zero horizontally. This means that there is no activity that outputs from F. According to the dependency matrix, the dependency graph of the new process model can be constructed as shown in Figure 5b. In this graph, activity B and D do not exist, because they are aggregated under the new activity X, and so they are represented as activity X.

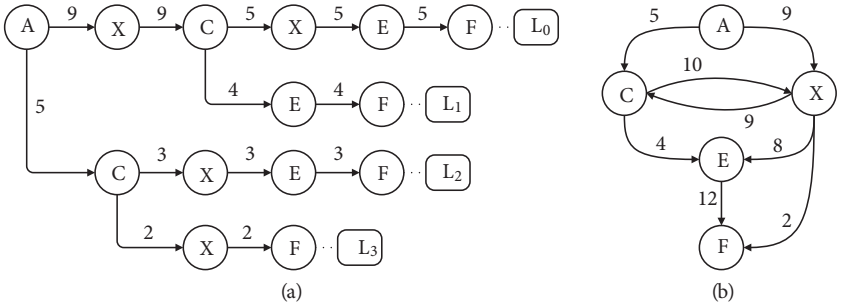

Figure 5. (a) Data model after activity aggregation. (b) Dependency graph representation of new process model after activity aggregation.

4.3. Activity addition

In this section, we are looking for an answer to the question of how the process model is affected if a new activity named *A3* is added between the activities *A1* and *A2*. There are many activity addition combinations. The user may want to add a new activity for specific conditions such as if and only if A comes after D, or if activity D is between E and F. The algorithm is able to handle any of these combinations. In the same example, assume that the user wants to add a new activity X between the activities C and B. At the same time, the user wants to add the same new activity X between the activities C and E.

The algorithm starts to traverse from root activity and tries to find dependencies of $C \rightarrow B$ and $C \rightarrow E$ to add new activity X between them.

Step 1: Firstly, the algorithm tries to reach L_0. Each time, it checks current and next activity. If the current activity is C and the coming activity is B or E, then it inserts the activity X between them such as $C \rightarrow X \rightarrow B$ or $C \rightarrow X \rightarrow E$. In the first path, there is no sequence that supplies this condition. Therefore, the algorithm does not apply any changes to the path of the leaf L_0.

Step 2: Secondly, it tries to reach L_1. In the second path, there is a sequence that supplies the desired condition. Thus, the path of L_1 changes into the following path: $A \rightarrow B \rightarrow C \rightarrow X \rightarrow E \rightarrow F$. After finding the second leaf, it is added to the dependency matrix. In this case, the dependency matrix has a new row and column for activity X.

Step 3: Thirdly, the algorithm tries to reach L_2. It looks for the condition in the tracked path. In the third path, there is a sequence that supplies the desired condition ($C \rightarrow B$). Hence, the path of L_2 changes into the following path: $A \rightarrow C \rightarrow X \rightarrow B \rightarrow E \rightarrow F$. After finding the third leaf, it is also added to the dependency matrix.

Step 4: Lastly, it tries to reach L_3. In the last path, there is no sequence that supplies the desired condition. Therefore, the algorithm does not apply any changes to the path of the leaf L_3. The final version of the data model and dependency graph are shown in Figures 6a and 6b, respectively.

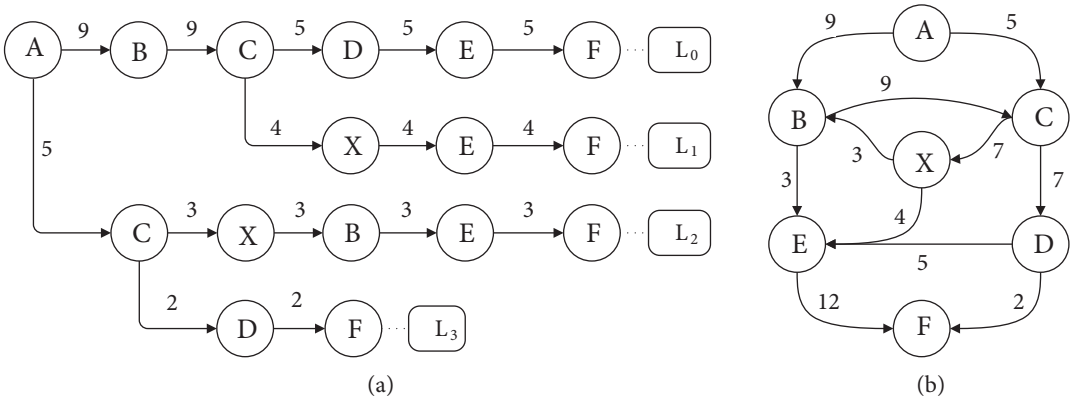

Figure 6. (a) Data model after activity addition. (b) Dependency graph representation of new process model after activity addition.

5. Example results of process model modification

In the repair dataset, different activities can be seen such as "Register" (accepting the faulty telephone), "Analyze Defect" (analyzing the telephone fault), "Repair (Simple)" (simple repair for fault of telephone),

"Repair (Complex)" (complex repair for fault of telephone), "Test Repair" (testing the telephone after repairing), "Restart Repair" (starting repairing again if the test fails), "Inform User" (informing the user after repair), and "Archive Repair" (archiving and closing fault record). After performing analysis on the event log of telephone repair process, the created process model can be viewed in Figure 7a.

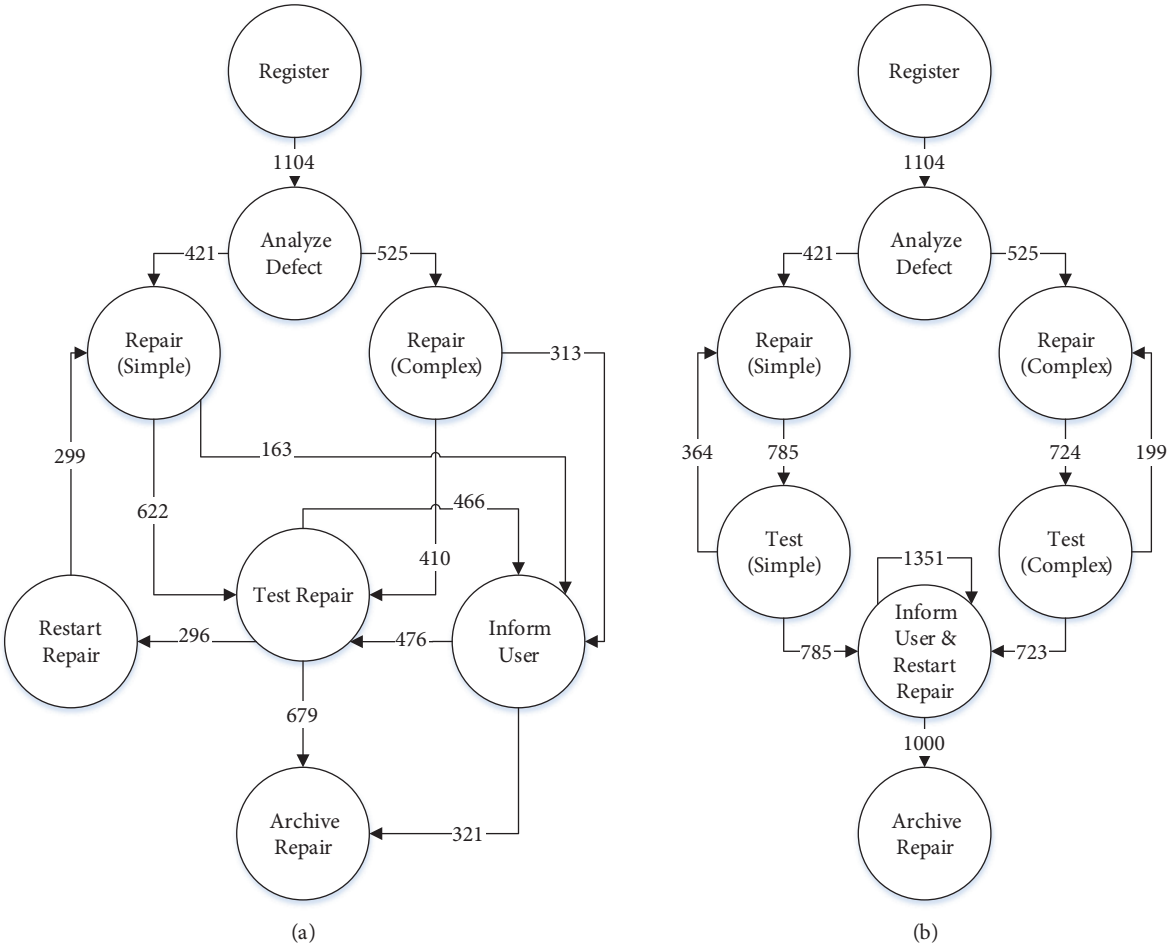

Figure 7. (a) Process model before modifications. (b) Process model after modifications.

When we look at the process model, it is observed that there are a lot of transitions between events, and as a result of this situation a complicated process flow is formed. Some improvement changes should be done to reduce this complexity and make the execution of process leaner. We want to make some changes in process flow to achieve this goal. Firstly, "Test (Simple)" and "Test (Complex)" events are added after each repair event to separate test operations. In this case, existing "Test Repair" event is removed from the process flow. Finally, "Inform User" and "Restart Test" events are aggregated under "Inform User & Restart Repair" event. After applying these changes, the process model is formed as shown in Figure 7b. As stated, we can perform different experiments in a simulation environment to improve the execution of processes. Once the most appropriate model has been identified, the improvement changes start to be implemented in real life. Thanks to the provided IPM algorithm, process improvements can be achieved quickly and truthfully.

6. Experimental results

To evaluate the developed IPM algorithm, two experiments were performed by using 2.4 GHz quad core processor, 16 GB RAM. The code was implemented in the Java platform. The experiments were performed on the ProM platform [23].

Datasets of traffic [38], hospital [39], and repair [40] processes are used to test the developed algorithm. Table 2 shows detailed information about the datasets. The hospital dataset includes real-life event logs of the clinical treatment process of an academic hospital in the Netherlands. It consists of 1143 traces and 150,291 events. The hospital dataset is composed of long and complex event logs, which are defined as Spaghetti. The traffic dataset includes event logs generated by an information system that performs road traffic control. It consists of 150,370 traces and 561,470 events. The repair dataset consists of synthetically created event logs for the telephone repair process. It consists of 1104 traces and 11,855 events.

Table 2. Dataset characteristics.

Dataset	Traces	Events	Event classes		Events per case	Event classes per case
Repair Dataset	1104	11,855	12	Min	4	4
				Mean	11	12
				Max	24	12
Hospital Dataset	1143	150,291	624	Min	1	1
				Mean	131	33
				Max	1814	113
Traffic Dataset	150,370	561,470	11	Min	2	2
				Mean	4	4
				Max	20	10

The first experiment was executed on hospital and traffic datasets to compare the running time and memory usage of four algorithms: Alpha Miner, Fuzzy Miner, HeuristicsMiner, and Interactive Process Miner (IPM - our algorithm). The results of this experiment given in Table 3 show the running time and memory usage of the IPM algorithm are better than those of the others. IPM created the process model in 2.38 s by using 463 MB RAM on the hospital dataset and 0.67 s by using 36 MB RAM on the traffic dataset. When we checked the running times and memory usage of the algorithms, we observed that IPM is the fastest algorithm and has the lowest memory consumption.

Table 3. Experimental results for performance evaluation.

	Hospital dataset		Traffic dataset	
	Running time (s)	Memory usage	Running time (s)	Memory usage
Alpha Miner	> 120.00	1503 MB	1.87	675 MB
Fuzzy Miner	39.11	1806 MB	7.52	789 MB
HeuristicsMiner	19.90	1087 MB	3.62	825 MB
Interactive Process Miner	2.38	463 MB	0.67	36 MB

The second experiment was executed on repair dataset. The Multi-Perspective Process Explore plug-in

was used to evaluate the quality of created process model [41]. This plug-in requires two input files. One of them is the XES file that contains the event log. The other one is the Petri net representation of the process model. This plug-in does not support the conversion of created model to Petri net representation for the Alpha Miner and Fuzzy Miner algorithms. For this reason, we only used HeuristicMiner algorithm to compare the results of the second experiment. In the second experiment, we focused on fitness and precision metrics to evaluate the success of the process model created by the algorithm. Fitness shows how much of the observed traces in the log are described by the process model (Eq. (1)). Precision is the ratio between the amount of traces observed in the event log and the amount of traces described by the model (Eq. (2)) [41].

$$fitness\,(L,N)=\frac{1}{2}\left(1-\frac{\sum_{\sigma\in L}L\,(\sigma)\times m_{N,\sigma}}{\sum_{\sigma\in L}L\,(\sigma)\times c_{N,\sigma}}\right)+\frac{1}{2}\left(1-\frac{\sum_{\sigma\in L}L\,(\sigma)\times r_{N,\sigma}}{\sum_{\sigma\in L}L\,(\sigma)\times p_{N,\sigma}}\right) \tag{1}$$

where L is event log, N is process model, σ is trace, $L(\sigma)$ is the frequency of trace σ, p is produced tokens, c is consumed tokens, m is missing tokens, and r is remaining tokens.

$$precision\,(P,\varepsilon)=\frac{\sum_{e\in\varepsilon}|obs_p\,(e)\,|}{\sum_{e\in\varepsilon}|pos_p\,(e)\,|} \tag{2}$$

where P is process model, ε is event log, e is event, $obs_p(e)$ is the observed behavior as seen in the event log, and $pos_p(e)$ is the possible behavior when event e occurs as allowed by a model.

The framework calculated the fitness as 84.8% and the precision as 73.9% for the created model by IPM as given in Table 4. Although fitness values are very close to each other (IPM: 84.8%, HeuristicMiner: 88.7%), there is a significant difference between the precision values of the two algorithms (IPM: 73.9%, HeuristicMiner: 57.5%). It is necessary to evaluate these two metrics together. When we look at fitness value, the process model created by IPM contains 84.8% of the event log. On the other side, the process model created by HeuristicMiner contains 88.7% of the event log. When we look at the precision value, 73.9% of the execution variants that we can build by looking at the process model created by IPM are observed in the event log. Conversely, only 57.5% of the execution variants that we can build by looking at the process model created by HeuristicMiner are observed in the event log. Clearly, HeuristicMiner has created a very general process model to conform it to the event log and a big part of execution variants expressed by the process model cannot be observed in the event log. From this point of view, we can say that the model created by IPM is closer to reality and more successful than HeuristicMiner.

Table 4. Experimental results to evaluate the success of process model.

Algorithms	Precision (%)	Fitness (%)
HeuristicsMiner	57.50	88.70
Interactive Process Miner	73.90	84.80

In summary, this study proposes a new process mining algorithm that runs on large datasets and handles execution records of running instances in a short time with low memory usage. The proposed algorithm provides an interactive method that allows users to modify the constructed model by adding, deleting, and aggregating the activities to see the impacts of process improvement changes in a simulation environment before applying decisions in real life.

7. Conclusions and future work

This paper proposes IPM, a new process mining algorithm, that has the capabilities of working on large volume of event logs and handling the execution records of running process instances to create a process model in a short time with high precision and fitness values and contains three different features, including activity deletion, aggregation, and addition operations on the process model to support a simulation environment for users.

The contribution of the proposed algorithm can be summarized as follows:

- Even though current algorithms analyze historical data, IPM is able to analyze historical event logs as well as to incorporate the execution records of ongoing processes into the process model instantly.
- IPM is able to analyze large volume of event logs. Experimental studies have proved that IPM is the fastest algorithm that consumes the least amount of memory.
- IPM enables modification of the discovered process model. Thus, IPM enables observation of the effects of possible decisions to be taken in a simulation environment. It has an important feature in order to make the right decision and to observe possible problems in advance.

Total running time and memory usage metrics were used to compare algorithms in performance perspective. Performance evaluation results have proved that the IPM algorithm outperforms the existing studies. Findings indicate that the running time and memory usage of the IPM algorithm are better than those of Alpha Miner, HeuristicsMiner, and Fuzzy Miner. In addition, the proposed algorithm was compared with the existing algorithms in terms of the success of the process model. In the experimental studies, fitness and precision metrics were used to evaluate the success of the process model created by the proposed algorithm. We need to evaluate this result by considering fitness and precision values together. From this point of view, we can say that IPM is more successful than the others.

In the future, it is possible to enhance IPM implementation by introducing new process mining perspectives such as organizational, resource, and time perspective in a user interactive environment.

References

[1] Agrawal R, Gunopulos D, and Leymann F. Mining process models from workflow logs. In: 6th International Conference on Extending Database Technology; 23–27 March 1998; Valencia, Spain. Heidelberg, Germany: Springer. pp. 467-483.

[2] Cook JE, Wolf AL. Discovering models of software processes from event-based data. ACM T Softw Eng Meth 1998; 7: 215-249.

[3] van der Aalst WMP, Weijters T, Maruster L. Workflow mining: discovering process models from event logs. IEEE T Knowl Data En 2004; 16: 1128-1142.

[4] Cook JE, Du Z, Liu C, Wolf AL. Discovering models of behavior for concurrent workflows. Comput Ind 2004; 53: 297-319.

[5] Herbst J, Karagiannis D. Workflow mining with InWoLvE. Comput Ind 2004; 53: 245-264.

[6] Schimm G. Mining exact models of concurrent workflows. Comput Ind 2004; 53: 265-281.

[7] van Dongen BF, van der Aalst WMP. A Meta model for process mining data. In: 17th Conference on Advanced Information Systems Engineering (EMOI-INTEROP Workshop); 13–17 June 2005; Porto, Portugal: FEUP edições. pp. 309-320.

[8] Wainer J, Kim K, Ellis CA. A workflow mining method through model rewriting. In: 11th International Workshop on Groupware; 25–29 September; Porto de Galinhas, Brazil. Heidelberg, Germany: Springer, 2005. pp. 184-191.

[9] Weijters AJMM, van der Aalst WMP, de Medeiros AKA. Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven Technical Report WP 2006; 166: 1-34.

[10] Günther CW, van der Aalst WMP. Fuzzy Mining: Adaptive process simplification based on multi-perspective metrics. In: 5th International Conference on Business Process Management; 24–28 September 2007; Brisbane, Australia. Heidelberg, Germany: Springer. pp. 328-343.

[11] van der Aalst WMP, Weijters T, Maruster L. Workflow mining: discovering process models from event logs. IEEE T Knowl Data E 2004; 16: 1128-1128.

[12] Weijters AJMM, van der Aalst WMP. Rediscovering workflow models from event-based data using little thumb. Integr Comput-Aid E 2003; 10: 151-162.

[13] de Medeiros AKA, Weijters AJM, van der Aalst WPM. Genetic process mining: a basic approach and its challenges. In: BPM 2005 International Workshops; 5 September 2005; Nancy, France. Heidelberg, Germany: Springer. pp. 203-215.

[14] de Medeiros AKA, Weijters AJMM, van der Aalst WPM. Genetic process mining: an experimental evaluation. Data Min Knowl Disc 2007; 14: 245-304.

[15] Bratosin C, Sidorova N, van der Aalst WMP. Distributed genetic process mining. In: IEEE Congress on Evolutionary Computation; 18–23 July 2010; Barcelona, Spain. New York, NY, USA: IEEE. pp. 1-8.

[16] Song W, Liu S, Liu Q. Business process mining based on simulated annealing. In: 9th International Conference for Young Computer Scientists; 18–21 November 2008; Hunan, China. New York, NY, USA: IEEE. pp. 135-139.

[17] Song M, Gunther CW, van der Aalst WMP. Trace clustering in process mining. In: BPM 2008 International Workshops; 1–4 September 2008; Milan, Italy. Heidelberg, Germany: Springer. pp. 109-120.

[18] Bose RPJC, van der Aalst WMP, Žliobaitė I, Pechenizkiy M. Handling concept drift in process mining. In: 23rd International Conference on Advanced Information Systems Engineering; 20–24 June 2011; London, UK. Heidelberg, Germany: Springer. pp. 391-405.

[19] Luengo D, Sepulveda M. Applying clustering in process mining to different versions of business process that changes over time. In: BPM 2011 International Workshops; 29 August 2012; Clermont-Ferrand, France. Heidelberg, Germany: Springer. pp. 153-158.

[20] van der Aalst WMP, Schonenberg MH, Song M. Time Prediction Based on Process Mining. Inform Syst 2011; 36: 450-475.

[21] van der Aalst WMP. Process mining: overview and opportunities. ACM Transactions on Management Information Systems 2012; 3: 1-17.

[22] van Dongen BF, van der Aalst WMP. EMiT: A process mining tool. In: 25th International Conference on Application and Theory of Petri Nets; 21–25 June 2004; Bologna, Italy. Heidelberg, Germany: Springer. pp. 454-463.

[23] van Dongen BF, de Medeiros AKA, Verbeek HMW, Weijters AJMM, van der Aalst WMP. The ProM framework: a new era in process mining tool support. In: 26th International Conference on Application and Theory of Petri Nets; 20–25 June 2005; Miami, USA. Heidelberg, Germany: Springer. pp. 444-454.

[24] Fahland D, van der Aalst WMP. Simplifying discovered process models in a controlled manner. Inform Syst 2013; 38: 585-605.

[25] Aleem S, Capretz LF, Ahmed F. Business process mining approaches: a relative comparison. International Journal of Science, Technology and Management 2015; 4: 1557-1564.

[26] Cheng HJ, Kumar A. Process mining on noisy logs — Can log sanitization help to improve performance? Decis Support Syst 2015; 79: 138-149.

[27] Fahland D, van der Aalst WMP. Model repair — aligning process models to reality. Inform Syst 2015; 47: 220-243.

[28] Rovani M, Maggi FM, de Leoni M, van der Aalst WMP. Declarative process mining in healthcare. Expert Syst Appl 2015; 42: 9236-9251.

[29] de Leoni M, van der Aalst WMP, Dees M. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Inform Syst 2016; 56: 235-257.

[30] Mannhardt F, de Leoni M, Reijers HA, van der Aalst WMP. Balanced multi-perspective checking of process conformance. Computing 2016; 98: 407-437.

[31] Pika A, van der Aalst WMP, Wynn MT, Fidge CJ, ter Hofstede AHM. Evaluating and predicting overall process risk using event logs. Inform Sciences 2016; 352-353: 98-120.

[32] Taxa N, Sidorovaa, N, Haakmab R, van der Aalst WMP. Mining local process models. Journal of Innovation in Digital Ecosystems 2016; 3: 183-196.

[33] Bolt A, de Leoni M, van der Aalst WMP. Scientific workflows for process mining: building blocks, scenarios, and implementation. Int J Softw Tools Te 2016; 18: 607-628.

[34] Suriadi S, Andrewsa R, ter Hofstedea AHM, Wynna MT. Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. Inform Syst 2017; 64: 132-1564.

[35] Mitsyuk AA, Shugurov IS, Kalenkova AA, van der Aalst WMP. Generating event logs for high-level process models. Simul Model Pract Th 2017; 74: 1-16.

[36] Bolt A, de Leoni M, ter Hofstedea AHM, van der Aalst WMP. Process variant comparison: using event logs to detect differences in behavior and business rules. Inform Syst 2017; 0: 1-14.

[37] Alizadeh M, Lu X, Fahland D, Zannone N, van der Aalst WMP. Linking data and process perspectives for conformance analysis. Comput Secur 2018; 73: 172-193.

[38] de Leoni M, Mannhardt F. Road traffic fine management process. Eindhoven University of Technology Dataset 2015. doi: 10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

[39] van Dongen BF. Real-life event logs - Hospital log. Eindhoven University of Technology Dataset 2011. doi: 10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

[40] Bose RPJC, van der Aalst WMP. Trace alignment in process mining: opportunities for process diagnostics. In: 8th International Conference on Business Process Management; 13–16 September 2010; Hoboken, NJ, USA. Heidelberg, Germany: Springer. pp. 227-242.

[41] Mannhardt F, de Leoni M, Reijers HA. The multi-perspective process explorer. In: Demo Session of the 13th International Conference on Business Process Management; 31 August–3 September 2015; Innsbruck, Austria. CEUR Workshop Proceedings. pp. 130-134.