DE GRUYTER OPEN

Journal of ELECTRICAL ENGINEERING, VOL 67 (2016), NO2, 111–123

DISCOVERING BLOCK–STRUCTURED PARALLEL PROCESS MODELS FROM CAUSALLY COMPLETE EVENT LOGS

Julijana Lekić [*] **— Dragan Milićev** [**]

α-algorithm is suitable to discover a large class of workflow (WF) nets based on the behaviour recorded in event logs, with the main limiting assumption that the event log is complete. Our research has been aimed at finding ways of discovering business process models based on examples of traces, *ie*, logs of workflow actions that do not meet the requirement of completeness. In this aim, we have modified the existing and introduced a new relation between activities recorded in the event log, which has led to a partial correction of the process models discovering technique, including the α-algorithm. We have also introduced the notion of causally complete logs, from which our modified algorithm can produce the same result as the α-algorithm from complete logs. The effect of these modifications on the efficiency of the process model discovering is mostly evident for business processes in which many activities can be performed in parallel. The application of the modified method for discovering block-structured models of parallel business processes is presented in this paper.

K e y w o r d s: process mining, business process model discovery, block-structured parallel process models, complete log, α-algorithm

1 INTRODUCTION

Understanding the behaviour of complex information systems is of essential importance for the ability to perform their modification, maintenance and improvement. The specification of the desired system behaviour is often not fully in accordance with its real behaviour, but provides only an idealized or desirable view of the business processes that are performed in the system. Fortunately, many information systems have a possibility to record their execution, and, in this way, to generate a trace about events describing the real system behaviour. Process mining (PM) techniques are based on the assumption that there is a strong relationship between process models and the "reality" recorded in the event log in the form of traces [1]. If the analysis of these traces can detect a behavioural model of the system, then the model can be reliably used for monitoring, maintenance and modification of the system.

In process mining, an event log is used for the implementation of three techniques: discovering [1–3], conformance and enhancement of the model [1]. The technique of interest in our research is model discovering [1–3]. This technique is used to construct models of business processes based only on examples of behaviour exhibited by the processes, *ie*, traces recorded in the event log. One of the basic and best known algorithms for process model discovering, based on records in a log, is the α-algorithm [1–3]. From records in the event log, the α-algorithm automatically generates a process model that belongs to a subclass of Petri nets [4], known as workflow (WF) nets [5]. Besides a number of other limitations, one of the basic limiting assumptions of this algorithm is that the event log needs to be complete [2, 3]. The assumption of completeness of the event log requires that in the traces recorded in it, there are all direct dependency relations that may exist between the activities of the observed business process.

The property of completeness of the log often implies the necessity of having a large number of traces in the log on which the "representative" model for the behaviour seen in the log has to be constructed. Therefore, our challenge was to find logs with potentially much lower number of traces, which may not be complete, but are sufficiently valid so that, using the appropriate algorithm based on the evidences recorded in such logs, a "representative" model can be obtained.

To achieve this, we have partially modified the technique of process model discovering, and also the α-algorithm itself by introducing the relation of indirect precedence as another basic relation between the activities recorded in the event log. Our aim was to determine how the existence of the relation of indirect precedence affects detection of concurrency of actions, and how it affects the efficiency of discovering the business process model. The results have shown that, by a generalization of the concurrency relation (with considering the relations of indirect precedence between the activities recorded in the event log), the concurrency relation, and therefore business process models, can be obtained from smaller event logs, which do not satisfy the condition of completeness as specified in [1–3]. Event logs, from which process models can be produced, often can be quite poor considering the number of traces that is necessary to obtain a valid

[*] Faculty of Technical Sciences, Kosovska Mitrovica, Kneza Milosa 7, Kosovska Mitrovica, 38220, Serbia, julijana.lekic@pr.ac.rs
[**] University of Belgrade, Faculty of Electrical Engineering, Bulevar kralja Aleksandra 73, Belgrade, 11000, Serbia

DOI: 10.1515/jee-2016-0016, Print (till 2015) ISSN 1335-3632, On-line ISSN 1339-309X © 2016 FEI STU

model, but must have certain properties in order to fulfill requirements of causal completeness which we have defined and presented in the paper.

This shows particularly good results in processes with a lot of activities that can be performed in a mutually independent order, *ie*, concurrently. In order to illustrate the effects of using the proposed modified technique of discovering process models and make it more noticeable, the results of applying the modified PM technique for discovering block-structured parallel business processes model will be shown on sample processes.

It should be noted at the very beginning that our assumptions and preconditions for process models (that have to be block-structured parallel models), to which our algorithm is applicable, may look as a serious restriction. It really is for real-world process models in general. However, our solution still covers a respectably wide subclass of process models and represents a first step in a more ambitious attempt to solve the very serious problem of log completeness. Although having a strong limitation, we still deem that even a partial solution to such a serious problem can be beneficial for future research and is worth reporting. In our future research, we will try to expand our work to other categories of processes.

In addition, the primary goal and vision of our research has a slightly different and wider context: it examines the possibility of interactive construction of block-structured parallel business processes models by demonstration. The idea is that the user performs possible scenarios of executed activities of a business process and according to this performance, the system can create the process model (candidate model), which will be in accordance to all demonstrated scenarios. The modified PM technique and the algorithm presented in this paper helped us to accomplish these goals and ideas with parallel processes, while the results and ways of realization of this wider context will be presented in our future publications. Nevertheless, at this stage, we would emphasize that the candidate model can be created interactively based on any event log, which does not have to be complete, but the final model can be created from the event log with causal completeness properties, as defined in this paper.

The next section provides an overview of the existing literature from the area of interest for our work and this paper. The third section gives some preliminaries about WF-nets, α-algorithm, parallel processes and block-structured parallel process models. The fourth section describes the proposed modification of the PM technique for business processes model discovering, considering the introduction of the relation of indirect precedence/subsequence between activities. In addition, a partial modification of the α-algorithm (the $\alpha^{||}$-algorithm) is proposed. In the fifth section, the notion of causally complete logs is defined and the application of the proposed modified PM technique for discovering block-structured models of parallel business processes based on these logs is presented. In the sixth section, the results of the performed experimental analysis are presented and discussed. The seventh section contains some conclusions and guidelines for the future work and research.

2 RELATED WORK

The process mining idea is not a new one. One of the earliest examples of usage of process mining in the context of workflow management, based on workflow graphs, was presented in [8]. Cook and Wolf have investigated similar issues in the context of software engineering, looking in particular sequential [9] and parallel behaviour of the process [10]. Dealing with sequential processes, in [11] they describe three methods for detection, one of which uses a neural network, while the other one is entirely based on the algorithmic approach, and the third one uses a Markovian approach. In [10], the same authors extend their work to concurrent processes where they propose specific metrics to discover models out of event streams, but they do not provide an approach to generate explicit process models. Herbst also dealt with the issue of process mining in the context of workflow management [11–12] using an inductive approach, observing sequential [12] and parallel models [11] separately. A notable difference between his work and other approaches is that the same task can appear multiple times in the workflow model, *ie*, the approach allows for duplicate tasks. Schimm [13] has developed a mining tool suitable for discovering hierarchically structured workflow processes.

Although many researchers have dealt with the ideas of process mining, the most comprehensive study is presented in the works of W.M.P. van der Aalst and his collaborators [1–6, 16, 18]. In [2, 3] a detailed description and formalization of techniques for discovering processes from workflow logs is presented, the α-algorithm for extracting process models from such logs is defined, and a representation of the model obtained in the form of a *sound* WF-net is shown. An introduction to process mining and an overview of existing techniques for discovering processes, and the problems which have been encountered in the application of the α-algorithm have been most fully presented in [1]. An overview of best practices and challenges is presented in [14], which is the work of a group of experts that was created in order to promote research, development and understanding of process mining, as well as its implementation and evolution.

To discover process models from traces recorded in workflow logs, many techniques have been proposed [2, 3, 8, 9, 15, 16]. Many of these techniques use Petri nets in the process of discovering and presenting the discovered process model. However, other very different approaches are also used for the same purpose.

Although the original α-algorithm is able to discover a large class of WF-nets, there are problems it cannot cope with: incompleteness of the event log, rare behaviours, complex routing constructions and others [1]. As a consequence, there is a large number of algorithms that overcome lacks of the basic α-algorithm [17]. Some of them are variants of the original α-algorithm, such as, for example, the α^+ algorithm [6] and α^{++} algorithm [18], while others use a completely different approach, such as: heuristic mining [9], genetic mining [16], fuzzy mining [19], process mining from a basis of regions [15] or flexible heuristics miner [7].

The α^+ algorithm [6] is an improved variant of the basic α-algorithm that can deal with problems related to short loops *ie*, loops of length one and two.

The α^{++} algorithm [18] is a variant of α-algorithm which can discover *non-free-choice* Petri nets, and to detect an *implicit dependency* among the tasks. An implicit dependency reflects indirect causal relationships between tasks. α^{++} algorithm is able to discover the original process model if the event log is complete and if there is no noise.

The heuristic miner (HM) [9] is an algorithm that can deal with noise and can be used to express the basic behavior (*ie*, without all details and without exceptions), which is recorded in the event log. The goal of the HM algorithm is to extract a *Causal net* (C-net) in the event log.

The flexible heuristics miner (FHM) [7] is an enhanced version of the HM algorithm that represents a heuristic-based algorithm for detecting relatively complete and understandable workflow models, even though there is some noise in the event log. Process models in the FHM approach are C-nets and *augmented C-nets*.

The genetic algorithm [16] of discovering process models is a flexible and robust and it can deal with noise and incompleteness of event logs, but it is not particularly efficient for large models and event logs, where the process of discovering can last pretty long. Causal matrices, which are used for internal representation by the genetic mining, support complex routing structures such as non-free-choice constructs and invisible tasks.

The fuzzy mining [19] represents a new approach to discovery of process models that overcomes the problem of models overcrowded with details by using the concept of a roadmap as a metaphor to visualize the resulting models.

State-based regions [15] can be used to construct a Petri net from a transition system, while language-based regions can be used to construct a Petri net from a prefix-closed language. Synthesis approaches using language-based regions can be applied directly to event logs, but to apply state-based regions, one first needs to create a transition system [1]. Techniques based on regions have problems with noise and incompleteness of event logs. Therefore, they are practically suitable when combined with heuristic techniques.

The inductive miner (IM) [21], aims to discover block-structured process models that are sound and fitting to the behavior represented in the event log. IM is an example of an algorithm that discovers process trees and for which rediscoverability has been proven. The algorithm works by dividing the activities of the log over a number of branches, such that the log can be split according to this division. IM partitions the activities, selects the most important process construct, splits the log and recurses until a base case is encountered. There are different IM variants, but generally they are divided into IM for incomplete event logs and IM for infrequent behavior. IM for incomplete event logs uses divide-and-conquer approach, but replaces the activity partition step by an optimisation problem. This IM variant introduces relations between activities, estimates probabilities of these relations and searches for a partition of activities that is optimal with respect to these probabilities.

The above mentioned algorithms and techniques have overcome some of problems, but the problem of incompleteness of logs, to the best of our knowledge, has not been overcome. This fact still remains a challenge for future research.

The subject of our research presented in this paper is the problem of completeness of event logs [1]. Part of this research was preliminarily announced in our previous paper [24], where we also dealt with logs that do not meet the requirement of completeness. In that paper, we briefly introduced our modified PM technique and the algorithm and we have also defined two types of logs, so called causally complete and *weakly complete logs*. This paper is largely devoted to causally complete logs. A more detailed presentation of our modified PM technique and algorithm is given in this paper, with the formal support of definitions, theorems and proofs. Moreover, the comparison of results obtained by applying our algorithm with the results of other process mining algorithms on causally complete logs was done. Finally, another big difference from the paper [35] is that this paper brings an extensive experimental analysis whose results are presented in this paper. In this analysis, values of the minimal sizes of complete logs and causally complete logs are compared for 100 real examples of parallel business processes. In this paper, plug-ins are also presented in the existing ProM framework designed for the needs of the experimental analysis.

3 PRELIMINARIES

In this section, we give some definitions of concepts used throughout this paper. First, we introduce the process modelling language (WF-nets) and its relevant concepts. Then we give a very brief introduction to the classical α-algorithm. Finally, we introduce the notions of *parallel process* and *block-structured parallel process models*. It is assumed that there is a known set of business process activities denoted with $\mathcal{A}$. The aim of the modelling process is to determine which activities should be performed and in which possible (partial) order. Activities can be performed sequentially, concurrently or optionally, and it is possible to repeat the execution of some activities (loops).

The α-algorithm is able to discover a business process model, represented as a Petri net [4] for a subclass of Petri

Table 1. Tabular presentation of the defined relationships between activities in a log

	$\rceil a > b$	$a > b$
$\rceil b > a$	$a\#b$	$a \rightarrow b$
$b > a$	$b \rightarrow a$	$a\|b$

nets known as *workflow* nets (WF-nets) [5]. WF-net is a Petri net with exactly one initial place, exactly one ending place, and all nodes accessible from the initial place (*ie*, when only the initial place is marked with a single token). WF-net is a natural subclass of Petri nets, making it suitable for modelling and analysing of operational processes. The α-algorithm is a mining algorithm based on the fact that for most of WF nets it holds that two activities are connected with a transition if and only if their causality can be detected from the event log [1].

3.1 The classical α-algorithm

The input for the α-algorithm is a simple event log L, as a multiset (a "bag") of traces over a set of actions $\mathcal{A}$, *ie*, $L \in \mathcal{B}(\mathcal{A}^*)$, where $\mathcal{B}(X)$ represents the set of all multisets of elements of X, and $\mathcal{A}^*$ is the set of all finite (but unlimited) sequences (traces) of the elements of $\mathcal{A}$ [2, 3]. Each trace corresponds to a single process case. The elements of the set $\mathcal{A}$ are the activities that correspond to transitions in the resulting Petri net, and are denoted by lowercase letters (*ie*, $a, b, c, \dots \in \mathcal{A}$), while sets of activities are denoted by uppercase letters (*ie*, $A, B, C, \dots \subseteq \mathcal{A}$). Four log-based ordering relations that aim to capture the relevant patterns in the log are defined as follows [1–3].

Definition 1 (Log-based ordering relations). Let L be an event log over $\mathcal{A}$, *ie*, $L \in \mathcal{B}(\mathcal{A}^*)$. Let $a, b \in \mathcal{A}$. Then

- $a >_L b$ if and only if there is a trace $\sigma = \langle t_1, t_2, t_3, \dots, t_n\rangle$ and $i \in \{1, \dots, n-1\}$ such that $\sigma \in L$ and $t_i = a$ and $t_{i+1} = b$,
- $a \rightarrow_L b$ if and only if $a >_L b$ and it is not $b >_L a$,
- $a\#_L b$ if and only if it is not $a >_L b$ and it is not $b >_L a$,
- $a\|_L b$ if and only if $a >_L b$ and $b >_L a$.

For a particular log L, these relations can be represented with a matrix with columns and rows for all actions in $\mathcal{A}$, and with each cell for (a, b) having one and only one of the symbols $>$, $\rightarrow$, $\leftarrow$, $\#$ or $\|$. (As it can be easily shown [1–3], these relations are disjoint and covering, *ie*, partition the set $\mathcal{A} \times \mathcal{A}$). Such a matrix is called the *footprint* of the event log, whereby the relations between any two actions $a, b \in T$ are defined in a manner as presented in Table 1.

Based on these relations, the α-algorithm is described by the following [1–3]

Definition 2 (α-algorithm). Let L be an event log over $T \subseteq \mathcal{A}$. $\alpha(L)$ is defined as follows.

(1) $T_L = \{t \in T \mid (\exists \sigma \in L)\, t \in \sigma\}$,

(2) $T_I = \{t \in T \mid (\exists \sigma \in L)\, t = \text{first}(\sigma)\}$,

(3) $T_O = \{t \in T \mid (\exists \sigma \in L)\, t = \text{last}(\sigma)\}$,

(4) $X_L = \{(A, B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge (\forall a \in A)(\forall b \in B)(a \rightarrow_L b) \wedge (\forall a_1, a_2 \in A)(a_1 \#_L a_2) \wedge (\forall b_1, b_2 \in B)(b_1 \#_L b_2)\}$,

(5) $Y_L = \{(A, B) \in X_L \mid (\forall (A', B') \in X_L)(A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B'))\}$,

(6) $P_L = \{p_{(A,B)} \mid (A, B) \in Y_L\} \cup \{i_L, o_L\}$,

(7) $F_L = \{(a, p_{(A,B)}) \mid (A, B) \in Y_L \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_L \wedge b \in B\} \cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$,

(8) $\alpha(L) = (P_L, T_L, F_L)$.

In Definition 2, $\text{first}(\sigma) = t_1$ and $\text{last}(\sigma) = t_n$, where $\sigma = \langle t_1, t_2, t_3, \dots, t_n\rangle$.

3.2 Parallel Processes and Block-Structured Parallel Process Models

Concurrency of business processes assumes a potential parallel execution of business activities within the execution of one process instance, or at least their (independent) execution in an arbitrary order. Inferring relations of parallelism from traces recorded in the event log is one of the key elements of the process model discovering, especially in the process with a large number of activities that can be performed concurrently. For example, if there are m independent activities that can be performed in parallel in a business process, the number of different traces in the event log in which they can appear is $m!$. In order to infer all relations of parallelism by using the α-algorithm, it is necessary that there are at least $m(m-1)$ different traces in the log [1]. Although the number of necessary traces in the event log, compared to the number of possible traces, is significantly decreased by using the α-algorithm, it is still rather big. For the purpose of illustration of the effect of our modified method on the capability and efficiency of the process model discovering, we shall define the particular class of business processes to which our method is restricted at this stage, namely *parallel processes*.

Definition 3 (Parallel process). *Parallel process* $N^{\|} = (P, T, F)$ is a sound[1] WF-net *ie*, $N^{\|} \in \mathcal{W}$, where in each possible execution of the process that ends with marking[2] the exit place, each activity from the set $t_i \in T$ executes exactly once (once and only once).

The requirement from Definition 3 that every activity from $t_i \in T$ has to be performed, imposes that each activity of a parallel business process has to appear in each trace $\sigma \in L$. As a consequence of this, there are no alternatives (optional branches) between activities of parallel process, *ie*, the relation $a_1 \#_L a_2$ (whereby $a_1 \neq a_2$) does not exist. Due to this, in a parallel process,

[1]Soundness corresponds to liveness and safeness of the corresponding short-circuited net [2, 3]. The set of all sound WF-nets is denoted with $\mathcal{W}$.

[2]The concept of marking is well known for Petri nets and precise definitions can be found in [2, 3].

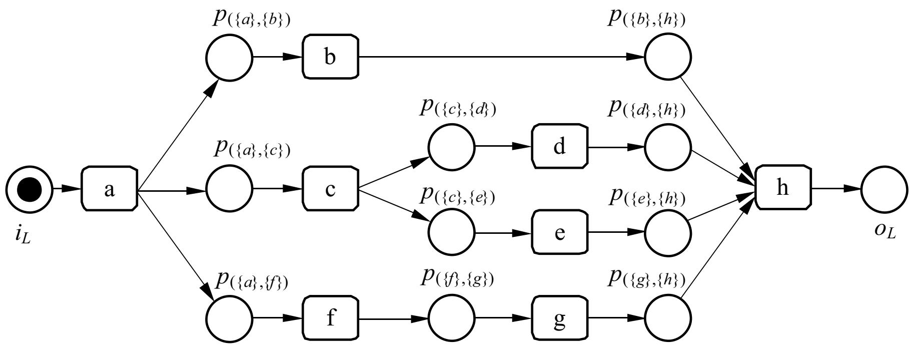

Fig. 1. Example of a block-structured parallel process model corresponding to the process tree: $\rightarrow(a, \wedge(b, \rightarrow(f, g), \rightarrow(c, \wedge(d, e))), h)$

activities perform either sequentially or in parallel, but not optionally.

The restriction from Definition 3 that each activity from $t_i \in T$ can be performed at most once, eliminates the possibility of the appearance of iterations (loops, repetitions) in the model of a parallel process.

A *block-structured* workflow net is a hierarchical workflow net that can be constructed from parts having a single entry and a single exit point by the recursive application of operators on the parts analogous to the constructs of structured programming, such as sequential, optional, iterative, and parallel execution. A process tree is a compact abstract representation of a block-structured workflow net: a rooted tree in which leaves are labelled with activities and all other nodes are labelled with operators [21]. A process model is block-structured if splits and joins are always paired into Single-Entry-Single-Exit (SESE) fragments [25].

According to the definition of block-structured WF-nets, block-structured process models and characteristics of parallel processes given by the Definition 3, process models we have dealt with make a subclass of block-structured process models that we named *block-structured parallel process models*. These are process models constructed from parts by the recursive application of only two operators: the first, $\rightarrow$, describing sequential composition, and the second, $\wedge$, describing parallel composition[3].

Figure 1 shows an example of a block-structured model of a parallel business process, presented in a form of a Petri net, which represents our running example in this paper.

4 MODIFIED TECHNIQUE FOR DISCOVERING PROCESS MODELS

In the basic α-algorithm, it is assumed that an event log L is complete in terms of the relation $>_L$. The assumption about the completeness of the log L requires that if a process model allows an activity b to be executed immediately after an activity a, then the log must contain at least one trace where b is really executed immediately after a, *ie* $a >_L b$ must hold. In this case, b is said to be *directly* (or *immediately*) following a in L [1–3].

Considering the fact that the existence of the relation of direct subsequence in the event log affects the definition of the relation of parallelism ($a \|_L b$ if $a >_L b$ and $b >_L a$) [1–3], our aim was to examine how the existence of the relation of *indirect subsequence* between activities affects the inference of the relation of parallelism, and how it reflects on the efficiency of discovering process models.

Each reduction in the number of traces in the event log, which are necessary for inferring the relation of parallelism, is resulting in more efficient model discovering. The efficiency in this context means that the method is capable of discovering the original model from generally much smaller event logs. Although this is not directly related with time efficiency of the algorithm itself, it indirectly certainly is, as the algorithm is based on the construction of relations, while this construction is proportional to the size of log, as it iterates through all traces in the log. In this way, the restrictive assumption of the α-algorithm about the event log completeness [2, 3] is relaxed. This is exactly what is achieved by our modification of the PM technique for process models discovering, as shown in the remaining parts of this paper.

4.1 Log-Based Ordering Relations in the Modified PM Technique

Our modification of the PM technique for discovering process models introduces a new basic relation $a \gg_L b$, which indicates that b *indirectly follows* a. This leads to a significantly faster discovering of activities that can be executed in parallel. The basic idea is simple: in order to conclude that two activities a and b are independent, *ie* concurrent, it is sufficient that the event log contains a certain trace in which a directly or indirectly precedes b, as well as some other trace in which b directly or indirectly precedes a.

[3]The operator $\rightarrow$ used here for constructing structured nets should not be confused with the symbol for causality relation used elsewhere.

[4]$\mathcal{P}(\mathcal{A}^*)$ is the powerset of $\mathcal{A}^*$

Table 2. The definition of the relations between activities in the modified PM technique for discovering process models

	$\rceil a > b$, $\rceil b \gg a$	$\rceil a > b$, $b \gg a$	$a > b$, $\rceil b \gg a$	$a > b$, $b \gg a$
$\rceil b > a$, $\rceil a \gg b$	$a\#b$	$b \Rightarrow a$	$a \rightarrow b$	$a \parallel b$
$\rceil b > a$, $a \gg b$	$a \Rightarrow b$	$a \parallel b$	N/A[5]	N/A
$b > a$, $\rceil a \gg b$	$b \rightarrow a$	N/A	$b \parallel a$	N/A
$b > a$, $a \gg b$	$b \parallel a$	N/A	N/A	N/A

N/A is "not applicable"

The log-based relations that are used to indicate the relevant patterns in the log in our modified technique of detecting process models are defined by Definition 4.

Definition 4 (Log-based ordering relations, in the modified PM technique of discovering process models). Let L be an event log over $\mathcal{A}$, *ie*, $L \in \mathcal{P}(\mathcal{A}^*)$[4]. Let $a, b \in \mathcal{A}$ be two activities. Then, by definition

- $a >_L b$ if and only if there is a trace $\sigma = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ and $i \in \{1, \ldots, n-1\}$ such that $\sigma \in L$ and $t_i = a$ and $t_{i+1} = b$,
- $a \gg_L b$ if and only if there is a trace $\sigma = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ and there are $i, j \in \{1, \ldots, n\}$ such that $i + 2 \leq j$, where $\sigma \in L$ and $t_i = a$, $t_j = b$, and it is not that $a >_L b$
- $a \rightarrow_L b$ if and only if $a >_L b$, and it is not $b >_L a$, and it is not $b \gg_L a$,
- $a \Rightarrow_L b$ if and only if $a \gg_L b$, and it is not $b >_L a$, and it is not $b \gg_L a$,
- $a \#_L b$ if and only if it is not $a >_L b$, and it is not $b >_L a$, and it is not $a \gg_L b$, and it is not $b \gg_L a$,
- $a \|_L b$ if and only if $a >_L b$ and $b >_L a$, or $a >_L b$ and $b \gg_L a$, or $a \gg_L b$ and $b >_L a$, or $a \gg_L b$ and $b \gg_L a$.

In a wide range of algorithms for process model discovering, there are algorithms which use a certain kind of a relation of indirect precedence [7, 18], but they are different by meaning and usage from the relation of indirect precedence introduced here. The difference between our relation of indirect precedence and the relation $x \gg y$ given in the Definition 3 in the paper [18] is that our relation of indirect precedence is released from the condition $\neg(t_k \lhd_L a$ or $t_k \rhd_L a)$. In addition, within the α^{++} algorithm [18], the indirect dependency (so called *implicit dependency*) between two activities imposes their connection by a place in the Petri net (Definition 2 in [18]), while in our conception, activities related with the indirect precedence are not connected by a place in the Petri net. The difference between the relation of indirect precedence $a \gg_L b$ given in Definition 4 of this paper and $a \gg_W b$ given in Definition 3 in [7] is in the fact that $a \gg_W b$ is related by to loops of length two, while in our modified PM techniques we do not deal with loops. In Definition 3 in [7], the relation $a \ggg_W b$ is also defined, and it implies a direct or indirect dependency. Unlike $a \ggg_W b$, our relation $a \gg_L b$ is only indirect dependency and it does not exist if there is a direct dependency between a and b.

According to Definition 4 it holds

Property 1. *Let L be an event log over $T \subseteq \mathcal{A}$. For any $a, b \in T$: $a \rightarrow_L b$, or $b \rightarrow_L a$, or $a \Rightarrow_L b$, or $b \Rightarrow_L a$, or $a \#_L b$, or $a \|_L b$. In other words, the relations $\rightarrow_L$, ${\rightarrow_L}^{-1}$, $\Rightarrow_L$, ${\Rightarrow_L}^{-1}$, $\#_L$ and $\|_L$ are mutually exclusive and partition $T \times T$.*

In order to verify this property, firstly the inverse of relations will be defined: ${>_L}^{-1}$, ${\gg_L}^{-1}$ and ${\Rightarrow_L}^{-1}$, as given for ${\rightarrow_L}^{-1}$ in [2, 3]:

${\rightarrow_L}^{-1}$ is the inverse of relation $\rightarrow_L$, *ie*, ${\rightarrow_L}^{-1} = \{(y, x) \in T \times T \mid x \rightarrow_L y\}$,

${>_L}^{-1}$ is the inverse of relation $>_L$, *ie*, ${>_L}^{-1} = \{(y, x) \in T \times T \mid x >_L y\}$,

${\gg_L}^{-1}$ is the inverse of relation $\gg_L$, *ie*, ${\gg_L}^{-1} = \{(y, x) \in T \times T \mid x \gg_L y\}$,

${\Rightarrow_L}^{-1}$ is the inverse of relation $\Rightarrow_L$, *ie*, ${\Rightarrow_L}^{-1} = \{(y, x) \in T \times T \mid x \Rightarrow_L y\}$.

Accordingly, the defined relations can be presented in the following forms:

$\rightarrow_L = \left(>_L \setminus ({>_L}^{-1} \cup {\gg_L}^{-1})\right)$,

${\rightarrow_L}^{-1} = \left({>_L}^{-1} \setminus (>_L \cup \gg_L)\right)$,

$\Rightarrow_L = \left(\gg_L \setminus ({>_L}^{-1} \cup {\gg_L}^{-1})\right)$,

${\Rightarrow_L}^{-1} = \left({\gg_L}^{-1} \setminus (>_L \cup \gg_L)\right)$,

$\#_L = (T \times T) \setminus (>_L \cup {>_L}^{-1} \cup \gg_L \cup {\gg_L}^{-1})$,

$\|_L = \left((>_L \cap {>_L}^{-1}) \cup (>_L \cap {\gg_L}^{-1}) \cup ({>_L}^{-1} \cap \gg_L) \cup (\gg_L \cap {\gg_L}^{-1})\right)$, due to that the result is $T \times T = \rightarrow_L \cup {\rightarrow_L}^{-1} \cup \Rightarrow_L \cup {\Rightarrow_L}^{-1} \cup \#_L \cup \|_L$.

The defined relations can be represented with a matrix, which represents a footprint of the event log, where the relations between any two activities $a, b \in T$, in the modified PM technique for discovering process models, are defined as it is presented in Table 2. Due to mutually exclusive relations, as given in Property 1, there is only one relation defined in Table 2 in any field of the footprint, as it will be shown later for the running example.

4.2 $\alpha^{\|}$-algorithm

Since the α-algorithm is based on the relations $\#_L$ and $\rightarrow_L$ [1–3], the loss of the relation $\#_L$ between different activities will lead to changes in the α-algorithm and its application to the discovering of a parallel process model. These changes are introduced by the following lemmas.

Lemma 1. *In parallel processes there are no relations $a_1 \#_L a_2$ or $b_1 \#_L b_2$ which are defined in the step (4) of the α-algorithm, where $a_1 \neq a_2$ and $b_1 \neq b_2$.*

P r o o f . Condition of Definition 3 that each activity from the set $t_i \in T$ executes exactly once requires that in each trace of a log there are both a and b for each $a, b \in T$, due to which there has to be either $a >_L b$ or $a \gg_L b$ or $b >_L a$ or $b \gg_L a$, which according to Definition 4 requires that $a \#_L b$ does not hold.

Lemma 2. *For parallel processes* $X_L = \{(A,B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge (\forall\, a \in A)(\forall\, b \in B)(a \rightarrow_L b)\}$.

Proof. In the α-algorithm [1–3] the step (4) is: $X_L = \{(A,B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge (\forall\, a \in A)(\forall\, b \in B)(a \rightarrow_L b) \wedge (\forall\, a_1, a_2 \in A)(a_1 \#_L a_2) \wedge (\forall\, b_1, b_2 \in B)(b_1 \#_L b_2)\}$. Since, regarding to Lemma 1, there are no $a_1 \#_L a_2$ or $b_1 \#_L b_2$, the elements defined on the basis of these relations are lost in the expression for X_L.

Lemma 3. *For parallel processes* $Y_L = X_L$.

Proof. In the α-algorithm the step (5) is: $Y_L = \{(A,B) \in X_L \mid (\forall\, (A',B') \in X_L)(A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B'))\}$, where $(A',B') \in X_L$ can be a non-maximal pair (obtained on the basis of relation $a \rightarrow_L b$), or a maximal pair (obtained on the basis of relations $a_1 \#_L a_2$ and/or $b_1 \#_L b_2$). $(A,B) \in Y_L$ are maximal pairs extracted from X_L. Since in parallel processes there are not $a_1 \#_L a_2$ or $b_1 \#_L b_2$, only pairs (A',B') remain, which are obtained on the basis of the relation $(a \rightarrow_L b)$, and which are, that way, maximal pairs, *ie*, (A,B). However, since $A = A'$ and $B = B'$, then $((A,B) \in Y_L) = ((A',B') \in X_L)$ too, *ie*, then $Y_L = X_L$.

Since in parallel processes (according to Lemma 3) $Y_L = X_L$, the algorithm for these processes has one step less than the α-algorithm. This modified α-algorithm will be denoted with $\alpha^{\parallel}(L)$, where "$\parallel$" reflects the fact that the algorithm targets parallel business processes. Regarding to the previously introduced and presented modifications, the $\alpha^{\parallel}$-algorithm can be described with the following definition.

Definition 5 ($\alpha^{\parallel}$-algorithm). Let L be an event log over $T \subseteq \mathcal{A}$. $\alpha^{\parallel}(L)$ is defined as follows:

(1) $T_L = \{t \in T \mid (\exists\, \sigma \in L) t \in \sigma\}$,

(2) $T_I = \{t \in T \mid (\exists\, \sigma \in L) t = \mathrm{first}(\sigma)\}$,

(3) $T_O = \{t \in T \mid (\exists\, \sigma \in L) t = \mathrm{last}(\sigma)\}$,

(4) $X_L = \{(A,B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge (\forall\, a \in A)(\forall\, b \in B)(a \rightarrow_L b)\}$,

(5) $P_L = \{p_{(A,B)} \mid (A,B) \in X_L\} \cup \{i_L, o_L\}$,

(6) $F_L = \{(a, p_{(A,B)}) \mid (A,B) \in X_L \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A,B) \in X_L \wedge b \in B\} \cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$,

(7) $\alpha^{\parallel}(L) = (P_L, T_L, F_L)$.

In Definition 5, $\mathrm{first}(\sigma) = t_1$ and $\mathrm{last}(\sigma) = t_n$, where $\sigma = \{t_1, t_2, t_3, \ldots, t_n\}$.

The application of the $\alpha^{\parallel}(L)$ algorithm for discovering block-structured models of parallel processes is described by the following definition.

Definition 6 (Ability to rediscover a parallel process model). Let $N^{\parallel} = (P,T,F)$ be a parallel process, and let $\alpha^{\parallel}(L)$ be the mining algorithm which maps logs of $N^{\parallel}$ onto sound WF-net, *ie*, $\alpha^{\parallel} \colon \mathcal{P}(T^*) \rightarrow \mathcal{W}$. If for any complete log L of $N^{\parallel}$, the $\alpha^{\parallel}$-algorithm returns $N^{\parallel}$ (modulo renaming of places), then the $\alpha^{\parallel}$-algorithm is able to rediscover $N^{\parallel}$.

5 APPLYING THE MODIFIED TECHNIQUE FOR DISCOVERING BLOCK–STRUCTURED PARALLEL PROCESS MODELS FROM CAUSALLY COMPLETE EVENT LOGS

A necessary condition for discovering the original network by the α-algorithm is that the log on which the algorithm is applied needs to be complete, where the condition of completeness is based on the relation $>_L$ [1–3]. The condition of completeness in our modified PM technique for model discovering is related to the causality relation $\rightarrow_L$, and accordingly a new type of completeness, *causal completeness*, is defined in this section.

5.1 Causally Complete Event Logs

For a particular process model to be discovered, there may be a large (in general, an unlimited) number of different complete logs. However, all these complete logs have the same footprint, *ie*, the same causality relation. We call this relation the *basic causality relation*. On the other hand, there may exist other logs for the same process model that are not complete, but which have the same footprint, *ie*, the same causality relation obtained from those logs. We are focused on investigating such logs, which we refer to as *causally complete logs*. Obviously, the idea is to find causally complete logs that may be, in general, much smaller than fully complete logs (in the terminology of the original α-algorithm).

Definition 7 (The basic causality relation). Let $N = (P,T,F)$ be a sound WF-net, *ie*, $N \in \mathcal{W}$, and let L be a complete workflow log of N. $\rightarrow^{B}{}_{N}$ is the *basic causality relation* of network N iff $\rightarrow^{B}{}_{N} = \rightarrow_L$.

Considering the so defined basic causality relation, a causally complete log is defined as follows.

Definition 8 (The causally complete log). Let $N = (P,T,F)$ be a sound WF-net, *ie*, $N \in \mathcal{W}$, and let $\rightarrow^{B}{}_{N}$ be the basic causality relation of N. L_c is a *causally complete workflow log* of N iff

1) $\rightarrow_{Lc} = \rightarrow^{B}{}_{N}$, and

2) for any $t \in T$ there is $\sigma \in L_c$ such that $t \in \sigma$.

Based on such a defined log L_c, one can obtain the original network of a parallel business process, so the following definition holds.

Definition 9 (Ability to rediscover a parallel process model from L_c). Let $N^{\parallel} = (P,T,F)$ be a parallel process, and let $\alpha^{\parallel}(L)$ be a mining algorithm which maps logs of $N^{\parallel}$ onto a sound WF-net, *ie*, $\alpha^{\parallel} \colon \mathcal{P}(T^*) \rightarrow \mathcal{W}$. If for any causally complete log L_c of $N^{\parallel}$, $\alpha^{\parallel}$-algorithm returns $N^{\parallel}$ (modulo renaming of places), then $\alpha^{\parallel}$-algorithm is able to rediscover $N^{\parallel}$.

Table 3. Footprint of the event $\log L$

	a	b	c	d	e	f	g	h
a	$\#$	$\rightarrow$	$\rightarrow$	$\Rightarrow$	$\Rightarrow$	$\rightarrow$	$\Rightarrow$	$\Rightarrow$
b	$\leftarrow$	$\#$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\rightarrow$
c	$\leftarrow$	$\parallel$	$\#$	$\rightarrow$	$\rightarrow$	$\parallel$	$\parallel$	$\Rightarrow$
d	$\Leftarrow$	$\parallel$	$\leftarrow$	$\#$	$\parallel$	$\parallel$	$\parallel$	$\rightarrow$
e	$\Leftarrow$	$\parallel$	$\leftarrow$	$\parallel$	$\#$	$\parallel$	$\parallel$	$\rightarrow$
f	$\leftarrow$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\#$	$\rightarrow$	$\Rightarrow$
g	$\Leftarrow$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\leftarrow$	$\#$	$\rightarrow$
h	$\Leftarrow$	$\leftarrow$	$\Leftarrow$	$\leftarrow$	$\leftarrow$	$\Leftarrow$	$\leftarrow$	$\#$

5.2 An example of the $\alpha^{\|}$-algorithm application

Having the same causal relationship as its basis for operation, the modified PM technique and the $\alpha^{\|}$-algorithm can rediscover the network from any log in which $\rightarrow_L = \rightarrow^B{}_N$, *ie*, from any causally complete log, as the example that follows will illustrate.

Let us observe a parallel process model shown in Fig. 1, and a $\log L$ obtained after several executions of the process. Only different traces, without stating the number of their appearances, are presented in the event log. This is due to the fact that the frequency of trace appearances is irrelevant for our consideration.

$$L = \big[\langle a,b,c,d,e,f,g,h\rangle, \langle a,f,g,b,c,e,d,h\rangle, \langle a,c,d,e,f,g,b,h\rangle, \langle a,b,f,g,c,d,e,h\rangle\big].$$

The basic causality relation for this example is

$$\rightarrow^B{}_N = \{(a,b),(a,c),(a,f),(b,h),(c,d),(c,e),(d,h),(e,h),(f,g),(g,h)\}\,.$$

The log-based ordering relations for this example are:

$$>_L = \{(a,b),(a,c),(a,f),(b,c),(b,f),(b,h),(c,d),(c,e),(d,e),(d,h),(e,d),(e,f),(e,h),(f,g),(g,b),(g,c),(g,h)\}\,,$$

$$\gg_L = \{(a,d),(a,e),(a,g),(a,h),(b,d),(b,e),(b,g),(c,b),(c,f),(c,g),(c,h),(d,b),(d,f),(d,g),(e,b),(e,g),(f,b),(f,c),(f,d),(f,e),(f,h),(g,d),(g,e)\}\,,$$

$$\|_L = \{(b,c),(c,b),(b,d),(d,b),(b,e),(e,b),(b,f),(f,b),(b,g),(g,b),(c,f),(f,c),(c,g),(g,c),(d,e),(e,d),(d,f),(f,d),(d,g),(g,d),(e,f),(f,e),(e,g),(g,e)\}\,,$$

$$\rightarrow_L = \{(a,b),(a,c),(a,f),(b,h),(c,d),(c,e),(d,h),(e,h),(f,g),(g,h)\}\,,$$

$$\Rightarrow_L = \{(a,d),(a,e),(a,g),(a,h),(c,h),(f,h)\}\,,$$

$$\#_L = \{(a,a),(b,b),(c,c),(d,d),(e,e),(f,f),(g,g),(h,h)\}.$$

From the footprint shown in Table 3, it can be seen that the causality relation of L is

$$\rightarrow_L = \{(a,b),(a,c),(a,f),(b,h),(c,d),(c,e),(d,h),(e,h),(f,g),(g,h)\}\,.$$

It can be observed that the causality relation of L is equal to the basic causality relation, *ie*, $\rightarrow_L = \rightarrow^B{}_N$, which makes the $\log L$ causally complete.

By applying the $\alpha^{\|}$-algorithm to the given $\log L$, we obtain

(1) $T_L = \{a,b,c,d,e,f,g,h\}$,

(2) $T_I = \{a\}$,

(3) $T_O = \{h\}$,

(4) $X_L = \{(\{a\},\{b\}),(\{a\},\{c\}),(\{a\},\{f\}),(\{b\},\{h\}),(\{c\},\{d\}),(\{c\},\{e\}),(\{d\},\{h\}),(\{e\},\{h\}),(\{f\},\{g\}),(\{g\},\{h\})\}$,

(5) $P_L = \{p_{(\{a\},\{b\})}, p_{(\{a\},\{c\})}, p_{(\{a\},\{f\})}, p_{(\{b\},\{h\})}, p_{(\{c\},\{d\})}, p_{(\{c\},\{e\})}, p_{(\{d\},\{h\})}, p_{(\{e\},\{h\})}, p_{(\{f\},\{g\})}, p_{(\{g\},\{h\})}, i_L, o_L\}$,

(6) $F_L = \{(a, p_{(\{a\},\{b\})}), (p_{(\{a\},\{b\})}, b), (a, p_{(\{a\},\{c\})}), (p_{(\{a\},\{c\})}, c), (a, p_{(\{a\},\{f\})}), (p_{(\{a\},\{f\})}, f), (b, p_{(\{b\},\{h\})}), (p_{(\{b\},\{h\})}, h), (c, p_{(\{c\},\{d\})}), (p_{(\{c\},\{d\})}, d), (c, p_{(\{c\},\{e\})}), (p_{(\{c\},\{e\})}, e), (d, p_{(\{d\},\{h\})}), (p_{(\{d\},\{h\})}, h), (e, p_{(\{e\},\{h\})}), (p_{(\{e\},\{h\})}, h), (f, p_{(\{f\},\{g\})}), (p_{(\{f\},\{g\})}, g), (g, p_{(\{g\},\{h\})}), (p_{(\{g\},\{h\})}, h), (i_L, a), (h, o_L)\}$,

(7) $\alpha^{\|}(L) = (P_L, T_L, F_L)$.

For the need of discovering original networks of block-structured parallel business processes by the modified PM method and the $\alpha^{\|}$-algorithm based on causally complete logs, we have developed a plug-in $\alpha^{\|}$-algorithm for the existing ProM framework [20]. The program code of the plug-in is located in a separate, new package, *alpha_parallel_algorithm* and is located at address [22].

At the same address there is a program code of the $\alpha^{\|}$ *-algorithm - helper plug-in*, which is given in a separate package *alpha_parallel_algorithm_basic_causal_relation*. The mentioned plug-in is created for the purpose of extraction of basic causal relations from a complete event log.

It should be noted that this utility of extracting the basic causality relation is *not* used by the $\alpha^{\|}$-algorithm. As it has been explained, the point is that our algorithm is guaranteed to discover the original process model if the input log is causally complete, just as the original α-algorithm is guaranteed to restore the original model if the log is fully complete; on the opposite, none of these algorithms can guarantee that the obtained model is the original one if the log is not causally or fully complete, respectively. The plug-in is just an independent, helper utility that can be used during experimentation to check whether the given log is causally complete or not, for the *given known* process model. Of course, in the process of process discovery, the model is unknown.

Figure 2 shows the $N^{\|}$ network obtained by applying the $\alpha^{\|}$-algorithm over $\log L$ and using the plug-in $\alpha^{\|}$-*algorithm.*

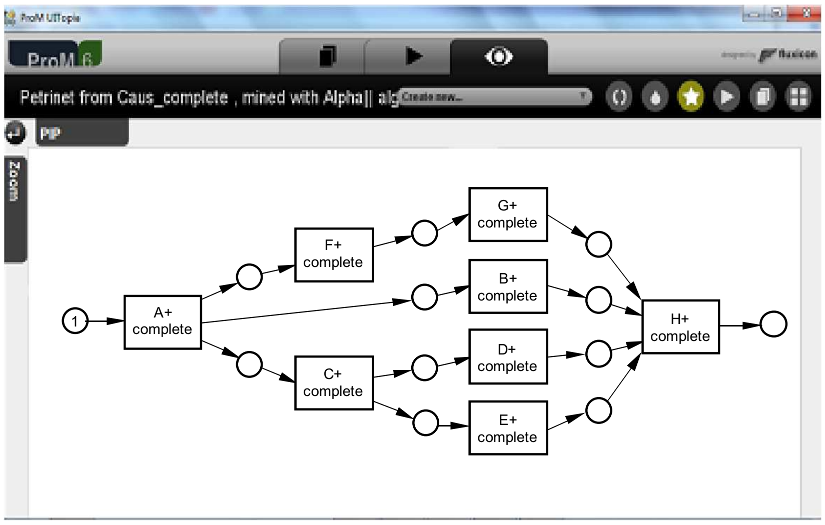

Fig. 2. WF-net $N^{||} = \alpha^{||}(L)$

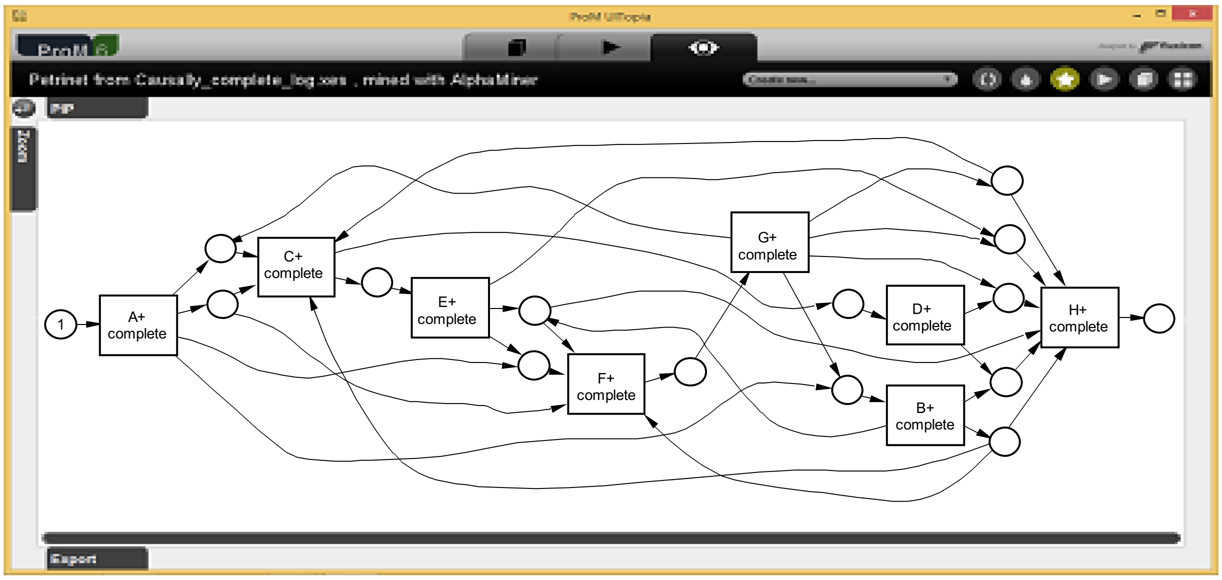

Fig. 3. The result of the application of Alpha Miner to the causally complete log L

It can be noticed that the network $N^{||} = \alpha^{||}(L)$ in Fig. 2 is equal to the original network in Figure 1, although it is obtained from a causally complete log which is not complete and which is smaller than the complete log, as it will be shown later in this paper.

5.3 Comparison of the $\alpha^{||}$-algorithm with other algorithms

It can be seen from the above said that we deal with the problem of completeness which originates from the original α-algorithm and is present in all other its modifications or other algorithms for discovering process models. As other algorithms resulted mainly in the attempt to overcome some other problems, but not the problem of completeness, and as our algorithm was created by a modification of the basic α-algorithm, we thought it would be most appropriate to compare it with the α-algorithm in the first place.

As for computational complexity, considering the fact that our modified method has one more basic relation ($\gg$) in comparison with the α-algorithm, the extraction of the needed relations could imply greater complexity. However, given that the time needed to build relations is directly proportionally to the size of the event log and that causally complete logs are smaller than complete logs, establishing of relations by our method can be reached faster. In addition, the $\alpha^{||}$-algorithm is based on a small number of relations (no relation $\#$ between different activities) and has one step less than the α-algorithm, which contributes to its simplicity.

In order to evaluate the potential and effectiveness of our algorithm, we have applied several other algorithms to the same sample log L given above and checked their ability to rediscover the original model. The sample log L is not complete, because there are missing elements in the relation $>_L$: $b > d$, $b > e$, $b > g$, $c > b$, $c > f$,

Table 4. The number of activities per example

Number of activities	Number of examples
5	3
6	16
7	21
8	13
9	14
10	10
11	4
12	3
13	6
14	3
15	4
16	2
17	1
Average number of activities per example	Total
8.97	100

Table 5. The number of branches per example

Number of branches	Number of examples
6	4
7	20
8	22
9	17
10	5
11	5
12	7
13	3
14	7
15	4
16	1
17	2
18	2
23	1
Average number of branches per example	Total
9.74	100

$c > g$, $d > b$, $d > f$, $d > g$, $e > b$, $e > g$, $f > b$, $f > c$, $f > d$, $f > e$, $g > d$ and $g > e$, which could be potentially performed on the basis of the process model given in Fig. 1, and the resulting WF-net $N^{\|}$.

When the original α-algorithm is applied on this causally complete log L, the model showed in Fig. 3 is obtained. Due to the lack of relations: $b > d$, $b > e$, $b > g$, $c > b$, $d > b$, $e > b$ and $f > b$, Alpha Miner was unable to detect that the activity b is parallel to the activities c, d, e, f and g. Due to the lack of relations: $f > c$, $f > d$, $f > e$, $c > f$ and $d > f$, Alpha Miner was unable to detect that the activity f is parallel to the activities c, d and e. Due to the lack of relations: $c > g$, $d > g$, $e > g$, $g > d$ and $g > e$, Alpha Miner was unable to detect that the activity g is parallel to the activities c, d and e. For these reasons, the model obtained by Alpha Miner is so complex and different from the original network.

The Appendix of this paper presents the results of the application of the available plug-ins for several other algorithms on the same given causally complete log L: Alpha++ Miner, Heuristics Miner, Fuzzy Miner, Genetic Miner, ILP Miner, Mine transition system and Inductive Miner, and can be found as technical report at the address [22]. As it can be seen from Appendix, neither of the selected algorithms have succeeded to rediscover the original model from the given causally complete log. On the contrary, in most cases, the rediscovered models were rather complex and very far from the original model. We also give our opinion about the reasons of the inability to rediscover the original model for these algorithms.

6 EXPERIMENTAL ANALYSIS

Our experimental analysis was performed on real examples, where the size of minimal complete and minimal causally complete logs were compared. In order to achieve this within the existing ProM framework [20], another plug-in has been developed $\alpha^{\|}$*-algorithm – minimal logs from complete log* which, from the given complete log extracts complete and causally complete logs with minimal possible number of traces, comparing their size. The program code of the plug-in is located in a separate, new package,
α_parallel_algorithm_minimal_logs_from_complete_log,
and is located at address [22].

6.1 Characteristics of analysed examples

The experimental analysis was performed on a sample of 100 real examples obtained by arbitrary manual search of the Internet and selecting publicly available models of business processes, which fulfill our conditions of block-structured models of parallel processes[5]. The considered examples with their .xes files complete and causally complete logs can be found at the address given in [22].

In Tables 4 and 5 some characteristics that reflect the network structure and size of the analysed examples are given. Table 4 shows the number of examples having a certain number of activities in the network, and the average number of activities per example. Table 5 shows the number of examples having a certain number of branches in the network, and the average number of branches per example.

6.2 Analysis results

Table 6 presents results of the performed comparative analysis of the minimal size of complete and causally complete logs, needed for discovering original networks of the considered examples.

[5]The models were found by searching the Web for the keywords: block-structured parallel process, parallel business process, activity diagram, BPMN diagrams *etc.*

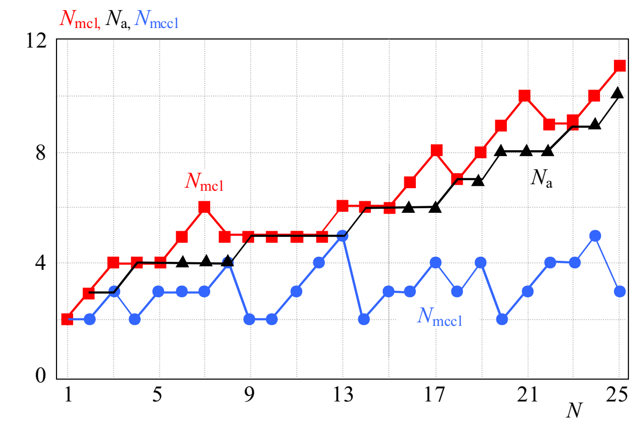

Fig. 4. The relation between Na and N_{mcl}, and Na and N_{mccl} depending on number of different networks constructed (N)

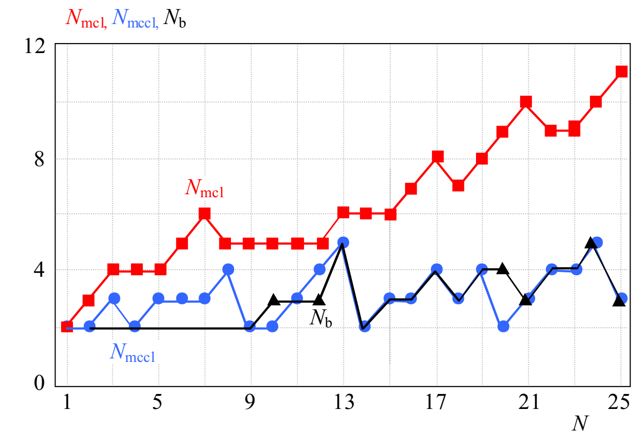

Fig. 5. The relation between N_b and N_{mcl}, and N_b and N_{mccl} depending on number of different networks constructed (N)

Table 6. Results of the comparative analysis of the minimal size of complete and causally complete logs

Number of examples	Number of traces in logs: Minimal complete log size	Number of traces in logs: Minimal causally complete log size	Minimal causally complete log less than minimal complete log by
1	2	2	0.00 %
12	3	2	33.33 %
13	4	2	50.00 %
41	4	3	25.00 %
5	5	2	60.00 %
4	5	3	40.00 %
4	5	4	20.00 %
3	6	2	66.67 %
3	6	3	50.00 %
1	6	5	16.67 %
3	7	3	57.14 %
3	8	4	50.00 %
1	9	2	77.78 %
2	9	4	55.56 %
2	10	3	70.00 %
1	10	5	50.00 %
1	11	3	72.73 %
Total			On average less by
100			37.55 %

The performed experimental analysis has shown that the size of causally complete logs from which the original networks of the observed parallel business processes can be discovered are lower, or (in the worst but rare case) equal to the size of complete logs.

From Table 6 it can be seen that in 99 examples (from the examined 100 examples), the size of the minimal causally complete logs are less than the size of minimal complete logs by an average of 37.55 %, while in only one example their values are equal.

In order to confirm that the hypothesis that the size of causally complete logs is less than the size of complete logs is statistically relevant, we have applied the Wilcoxon-Mann-Whitney rank-sum nonparametric test [34] on the results from Table 6.

If we denote: X = size of minimal causally complete logs, and Y = size of minimal complete logs, it is needed to test the null hypothesis that distributions of these two marks (labels) are equal, *ie*, $H_0\colon F_X = F_Y$, against the alternative hypothesis H_1: "The size of minimal causally complete logs X is less than the size of minimal complete logs Y". The corresponding critical area C in this case is:

H_0	H_1	C
$F_X = F_Y$	X is less than Y	$z_0 \le -z_{0.5-\alpha}$

Applying the Wilcoxon-Mann-Whitney test on the obtained experimental analysis results, the following values are obtained:

$n_1 = 99$; $n_2 = 99$; $n = n_1 + n_2 = 198$; $V = 240$; $E(V) = n_1 n_2/2 = 4900.5$; $D(V) = E(V)(n+1)/6 = 162533.25$; $z_0 = (V - E(V))/[D(V)]^{1/2} = -11.56$.

For the level of significance $\alpha = 0.05$, the critical area of this test is $C = (-\infty, -1.645]$. Since the realized value of the test statistic z_0 belongs to the critical area C, the null hypothesis H_0 is rejected in favour of alternative hypothesis H_1. In other words, for the level of significance $\alpha = 0.05$, it can be concluded that the claim that the size of minimal causally complete logs is less than the size of minimal complete logs is statistically significant.

Observing the structure of networks in the considered examples, the experimental analysis has shown that the size of the log, from which the original networks can be discovered, depends on the number of parallel branches in the network, as well as on the total number of activities in mutually parallel branches.

In Table 7 the results of the performed experimental analysis are presented, in which considered examples are grouped according to the total number of activities in parallel branches and the number of parallel branches in the network. Considering such groups of examples, sizes minimal complete logs and minimal causally complete logs are presented, as well as the difference between them, and the difference between the total number of activities in parallel branches and the number of parallel branches in the network.

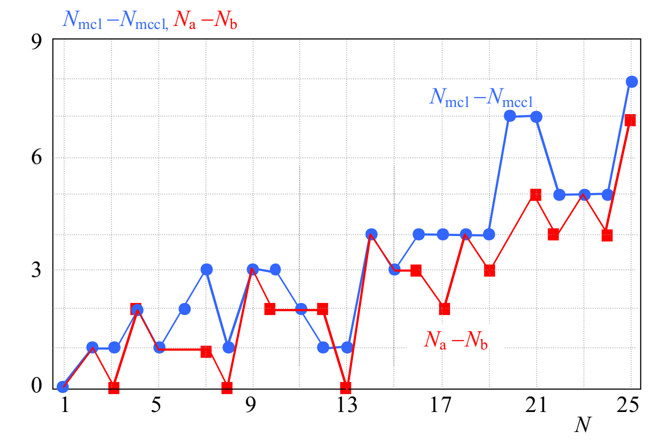

Fig. 6. The relation between two differences: $N_a - N_b$ and $N_{\text{mcl}} - N_{\text{mccl}}$ depending on number of different networks constructed (N)

Table 7. The influence of the number of parallel branches in the network and the total number of activities in each parallel branch on the log size

Number of examples	N_a	N_b	$N_a - N_b$	N_{mcl}	N_{mccl}	$N_{\text{mcl}} - N_{\text{mccl}}$
1	2	2	0	2	2	0
12	3	2	1	3	2	1
39	3	3	0	4	3	1
11	4	2	2	4	2	2
6	4	3	1	4	3	1
2	4	3	1	5	3	2
1	4	3	1	6	3	3
3	4	4	0	5	4	1
4	5	2	3	5	2	3
1	5	3	2	5	2	3
2	5	3	2	5	3	2
1	5	3	2	5	4	1
1	5	5	0	6	5	1
3	6	2	4	6	2	4
2	6	3	3	6	3	3
1	6	3	3	7	3	4
1	6	4	2	8	4	4
2	7	3	4	7	3	4
1	7	4	3	8	4	4
1	8	2	6	9	2	7
1	8	3	5	10	3	7
1	8	4	4	9	4	5
1	9	4	5	9	4	5
1	9	5	4	10	5	5
1	10	3	7	11	3	8

Here and In Figs. 4–6, the following notations are used:
N_a – the total number of activities in parallel branches
N_b – the number of parallel branches in the network
N_{mcl} – the number of traces in minimal complete logs
N_{mccl} – the number of traces in minimal causally complete logs

In Fig. 4 the influence of the total number of activities in parallel branches (N_a) on the number of traces in minimal complete logs (N_{mcl}) and on the number of traces in minimal causally complete logs (N_{mccl}) is presented. From Fig. 4 (and from Tab. 7) it can be seen that the size of minimal complete logs is proportional to the total number of activities in mutually parallel branches. It can also be seen that the number of activities in parallel branches does not affect the size of minimal causally complete logs.

In Fig. 5 the influence of the number of parallel branches in the network (N_b) on the number of traces in minimal complete logs (N_{mcl}) and on the number of traces in minimal causally complete logs (N_{mccl}) is presented. From Fig. 5 (and from Tab. 7) it can be seen that the size of minimal causally complete logs is proportional to the number of parallel branches in the network. It can also be seen that the number of parallel branches in the network does not affect the size of minimal complete logs.

In Fig. 6 the relation between difference the total number of activities in parallel branches and the number of parallel branches in the network $(N_a - N_b)$ is presented, as well as the difference the number of traces in minimal complete logs and the number of traces in minimal causally complete logs $(N_{\text{mcl}} - N_{\text{mccl}})$. From Fig. 6 (and from Tab. 7) it can be seen that the difference between the size of the minimal complete logs and the size of the minimal causally complete logs, expressed in the number of traces, is proportional to the difference between the total number of activities in parallel branches and the number of parallel branches in the network.

7 CONCLUSION

The results presented in this paper address concurrent processes without loops or optional branches. The examples show that with our modification of the discovering technique, we are able to reduce the problem of completeness of logs in parallel processes that occurs in the basic α-algorithm. That way, we can improve the efficiency of obtaining a block-structured process model, with the meaning that our algorithm can guarantee the discovery of the original model from significantly smaller logs, satisfying a restricted precondition of causal completeness (instead of full completeness).

Preliminary results presented in [24] suggest the possibility that by applying the presented modified technique, parallel process models can be discovered from logs whose size are even less than the size of causally complete logs. Our future work will be focused on investigating logs with the smallest possible number of records from which the block-structured model of parallel process can be discovered, as well as on conditions that these logs have to fulfill.

Our assumptions and preconditions for process models (that have to be block-structured parallel models), to which our algorithm is applicable, may look as a serious restriction. However, our solution still covers a respectably wide subclass of process models and represents a first step in a more ambitious attempt to solve the very serious problem of log completeness. In our future research, we will try to expand our work to other categories of processes.

References

[1] AALSTVANDER, W. M. P. : Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer-Verlag, Berlin, (2011).

[2] AALSTVANDER, W. M. P.—WEIJTERS, A. J. M. M.—MARUSTER, L. : Workflow Mining: DisProcess Models from Event Logs., IEEE Transactions on Knowledge and Data Engineering, **16** No. 9 (2004), 1128-1142.

[3] AALSTVANDER, W. M. P.—WEIJTERS, A. J. M. M.—MARUSTER, L. : Workflow Mining: Which Processes can be Rediscovered?, BETA Working Paper Series, WP 74, Eindhoven Univ. of Technology, Eindhoven, (2002).

[4] AALSTVANDER, W. M. P.—STAHL, C. : Modeling Business Processes: A Petri Net Oriented Approach. MIT press, Cambridge, MA, (2011).

[5] AALSTVANDER, W. M. P. : Verification of Workflow Nets, Application and Theory of Petri Nets, Azema, P. and Balbo, G. eds., (1997), 407-426, Berlin: Springer-Verlag.

[6] DEMEDEIROS, A. K. A.—AALSTVANDER, W. M. P.—WEIJTERS, A. J. M. M. : Workflow Mining: Current Status and Future Directions. In Meersman, R., Tari, Z. and Schmidt, D. C. editors, On the Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE,, vol. 2888 of Lecture Notes in Computer Science (2003), pages 389-406, Springer, Berlin.

[7] WEIJTERS, A. J. M. M.—RIBEIRO, J. T. S. : Flexible Heuristics Miner (FHM), BETA Working Paper Series, WP 334 (2010), Eindhoven University of Technology, Eindhoven.

[8] AGRAWAL, R.—GUNOPULOS, D.—LEYMANN, F. : Mining Process Models from Workflow Logs, Proc. Sixth Intl Conf. Extending Database Technology (1998),469-483.

[9] COOK, J. E.—WOLF, A. L. : Discovering Models of Software Processes from Event-Based Data, ACM Trans. Software Eng. and Methodology **7** No. 3 (1998) pages215-249.

[10] COOK, J. E.—WOLF, A. L. : Event-Based Detection of Concurrency, Proc. Sixth Intl Symp. the Foundations of Software Eng. (FSE-6), (1998),35-45.

[11] HERBST, J. : Dealing with Concurrency in Workflow Induction jour Proc. European Concurrent Eng. Conf., Baake, U., Zobel, R. Al-Akaidi, M. eds. (2000).

[12] HERBST, J.—KARAGIANNIS, D. : Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models jour Intl J. Intelligent Systems in Accounting, Finance, and Management,.

[13] SCHIMM, G. : Process Miner-A Tool for Mining Process Schemes from Event-Based Data, Proc. Eighth European Conf. Artificial Intelligence (JELIA), Flesca, S. and Ianni, G., eds. (2002),525-528.

[14] DANIEL, F.—BARKAOUI, K.—DUSTDAR, S. : IEEE Task Force on Process Mining: Process Mining Manifesto, In Business Process Management Workshops, ser. Lecture Notes in Business Information Processing, Eds., **99**, (2012), Springer-Verlag, Berlin, 169194.

[15] CARMONA, J.—CORTADELLA, J.—KISHINEVSKY, M. : A Region-Based Algorithm for Discovering Petri Nets from Event Logs, in Business Process Management (BPM2008), 358-373.

[16] MEDEIROS, A.—AALSTVANDER, W. M. P.—WEIJTERS, A. J. M. M. : Genetic Process Mining: An Experimental Evaluation, Data Mining and Knowledge Discovery, **14** No. 2 (2007), 245-304.

[17] [VANDONGEN, B. F.—ALVESDEMEDEIROS, A. K.—WEN, L. : Process Mining: Overview and Outlook of Petri Net Discovery Algorithms, ToPNOC 2 (2009), 225-242.

[18] WEN, L.—AALSTVANDER, W. M. P.—WANG, J.—SUN, J. : Mining Process Models with Non-Free-Choice Constructs, Data Mining and Knowledge Discovery, **15** No. 2 (2007), 145-180.

[19] GUENTHER, C. W.—AALSTVANDER, W. M. P. : Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics, in: BPM 2007, 4714 of LNCS (2007), Springer, 328-343.

[20] AALSTVANDER, W. M. P.—VANDONGEN, B. F.—GÜNTHER, C. W.—MANS, R. S.—ALVESDEMEDEIROS, A. K.—ROZINAT, A.—RUBIN, V.—SONG, M.—VERBEEK, H. M. W.—WEIJTERS, A. J. M. M. : ProM 4.0: Comprehensive Support for Real Process Analysis,, In Kleijn, J. and Yakovlev, A. editors, Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007), 4546 of Lecture Notes in Computer Science (2007),, 484-494, Springer-Verlag, Berlin.

[21] LEEMANS, S. J. J.—FAHLAND, D.—AALSTVANDER, W. M. P. : Discovering Block-structured Process Models from Incomplete Event Logs, In Ciardo, G. and Kindler, E. editors, Applications and Theory of Petri Nets 2014, l8489 of Lecture Notes in Computer Science (2014),, 91-110, Springer-Verlag, Berlin.

[22] https://drive.google.com/open?id=0B7gNCSuMP3pKNEdN V0I0SUhHUjA.

[23] POPOVIC, C. B. : Mathematical Statistics, Faculty of Sciences and Mathematics, University of Niš (2009), (in Serbian).

[24] LEKIC, J.—MILICEV, D. : Discovering Models of Parallel Workflow Processes from Incomplete Event Logs, In Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD-2015), 477-482.

[25] LING, J. M.—ZHANG, L.—FENG, Q. : An Improved Structure-based Approach to Measure Similarity of Business Process Models, In: Proc. of the 26th Intl Conf. on Software Engineering and Knowledge Engineering, 2014, 377-380.

Received 15 December 2015

Julijana Lekić received her BSc degree in Electrical Engineering from the Faculty of Electrotechnical Engineering, University in Priština, Serbia, and her MSc degree from the Faculty of Electrotechnical Engineering, University of Belgrade, Serbia. She is currently a Teaching Assistant with the Faculty of Technical Sciences, University in Priština (temporarily displaced in Kosovska Mitrovica). Her current research interests are in computer science, particularly information systems, software engineering, and business process modeling and mining. She is currently pursuing her PhD in these areas.

Dragan Milićev is Full Professor at the University of Belgrade, Faculty of Electrical Engineering, Department of Computing. He received his dipl. ing. degree in 1993, MSc in 1995, and PhD in 2001, all from the University of Belgrade. He is specialized in software engineering, model-based engineering, model-driven development, UML, software architecture and design, business process modeling, information systems, and real-time systems. He is a member of the Program Committees of several premier international conferences on model-based engineering (MODELS, ECMFA, and MODELSWARD), and a member of the Editorial Board of Springer's Software and System Modeling journal (SoSyM). He authored three books on object-oriented programming and UML, published in Serbian, and a book in English, published by Wiley/Wrox, entitled "Model-Driven Development with Executable UML" (also published in Chinese by Tsinghua University Press in Beijing, China). With almost 30 years of extensive industrial experience in building complex commercial software systems, he has been serving as the chief software architect, project manager, or consultant in a few dozen international projects. He is the founder of SOL Software (www.sol.rs), a software research and development company specialized in designing software development tools based on modeling, as well as in building custom applications and systems.