# Process Discovery
## *Automated Approach for Block Discovery*

Souhail Boushaba, Mohammed Issam Kabbaj and Zohra Bakkoury
*Department of Computer Science, AMIPS Research Group, Ecole Mohammadia d'Ingénieurs, Mohammed V$^{th}$ University - Agdal, Av Ibn Sina, Rabat, Morocco*

Keywords: Process Mining, Business Process Management, Process Discovery, Block Discovery.

Abstract: Process mining is a set of techniques helping enterprises to avoid process modeling which is a time-consuming and error prone task. Process mining includes three topics: process discovery, conformance checking, and enhancement (IEEE Task Force on Process Mining: Process Mining Manifesto, 2012). The principle of process discovery is to extract information from event logs to capture the business process as it is being executed. Several techniques in literature (α algorithm, α+ algorithm and others) can be applied to discover a process model from a workflow log. However, as the amount of information grows exponentially, the log files (input of a process discovery algorithm) get bigger. In fact, classical techniques, which inspect relation between each couple of tasks will have problem dealing with big data. To this end, we introduced in (Boushaba et al., 2013) a new approach aiming to extract a block of tasks from event logs. In this paper, we present a new algorithm, based on a matrix representation, to detect a block of tasks. In addition, we develop an application to automate our technique.

## 1 INTRODUCTION

In the current circumstances of competition, every company needs to accelerate its business processes. Therefore, reviewing the process execution becomes more and more essential in every business. So instead of starting by process modeling which is a time consuming and error prone task, process discovery techniques generate process models from workflow logs available in current information systems (Figure 1 illustrates the principle of process discovery).

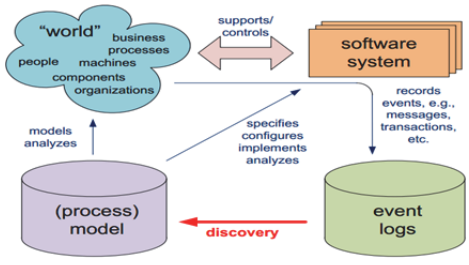


Figure 1: Process discovery (IEEE Task Force on Process Mining: Process Mining Manifesto, 2012).

As mentioned in figure1, process discovery techniques scan information system's log files to determine the process model that has been used.

Many techniques in literature are used to discover process models from event logs. However, as the data continue growing (event logs containing millions of events), classical approaches are definite but they will not keep up. In fact, more sophisticated approaches such as genetic mining (De Medeiros et al., 2004) are faster than the direct ones but are extremely inefficient as they depend on randomization to find new alternatives.Therefore we present a new block discovery oriented method (a block is a set of tasks having the same behavior with all other tasks in the process model) in (Boushaba et al., 2013). Our method for block discovery starts by representing the workflow log by a matrix and then applying a set of filters to extract the set of blocks composing the final petri net.

The idea of our algorithm is to detect two or more blocks of activities composing a partition of our log. Each block can be examined independently. Therefore, we can be able to reduce the time processing and the complexity of the extraction process. The remainder of this paper is organized as follows: In section 2 we present related works. Section 3 introduces preliminaries. Section 4 shows how to detect blocks using filters; in Section 5 we


Boushaba S., Issam Kabbaj M. and Bakkoury Z..
Process Discovery - Automated Approach for Block Discovery.
DOI: 10.5220/0004896402040211
In *Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2014), pages 204-211
ISBN: 978-989-758-030-7

present our algorithm for block detection. A case study is given in Section 6. Finally, a conclusion is drawn in Section 7.

## 2 RELATED WORKS

In (Van Der Aalst et al., 2004) and (De Medeiros et al. 2003), authors present an algorithm named the α-algorithm which discovers a large set of business process models called structured workflow nets. This basic algorithm have been extended in (Weijters and van der Aalst, 2003), to deal with noise problem by using some metrics expressed in literature, and in (Wen et al., 2007). the problem of non-free choice constraint was also dealt with.

The idea of matrix representation was presented before; Alves De Medeiros et al. have addressed the same issue in (De Medeiros et al., 2004), by representing the process log as a matrix. However authors use the same basic operator (direct succession) inspecting the relation between each couple of tasks (i.e. non-block-oriented method). Our approach aims to transform a workflow log into a matrix (using new operators) to first detect blocks and then discover the process model. In (Chen et al., 2009) Li describes a complete method for discovering a reference process model from a set of process variants, the main difference between our approach and Li's method is that the input is not the same: we discover process model from the information registered in the log. As per Li's approach, the entry items are the process variants.

Mathematically, the matrix representation in Li's approach cannot lead to proper matrix calculus, due to its use of symbols such as L for loops, for Xor block as components of the matrix. In our approach, the matrix contains strictly figures.

All these methods and others inspect the ordering relations in a log and discover the ordering relations between every couple of tasks in a log, which extends the processing time.

The described method in this paper, reinforcing the method presented in (Boushaba et al., 2013), explores the flow relations between blocks of tasks. Besides in (Lemans et al., 2013) described a method called inductive miner "IM" extended to inductive miner-infrequent "IMi" in (Leemans et al., 2013) in order to deal with infrequent behaviour, was also proposed for the same purpose. It aims to extract the set of blocks composing the process model. However, the principal of the IM method is based on cutting the most relevant design pattern from the directly-follows graph which is not a well deterministic criterion.

## 3 PRELIMINARIES

In this paper, we present an approach for block discovery based on a matrix representation. This section introduces the concepts used in the remainder of this paper. To model our processes, we use a variant of Petri nets named Place Transitions Net for more details, the reader is referred to (Murata, 1989).

### 3.1 Indirect Succession

The first step in our work is to create an operator which defines indirect succession in a given workflow log. Our operator is denoted ⋙. The basic ordering relations are defined as follows:

#### 3.1.1 Indirect Succession Operators

Let W be a workflow log over T, i.e., W ∈ P(T*).
Let a, b ∈T:

- $a \ggg_w b$ if and only if there exists a trace $\sigma = t_1 t_2 t_3 \dots t_n$ and $i, j \in \{1, \dots, n\}$ such that $\sigma \in W$, $t_i = a$, $t_j = b$and $i < j$, (Indirect Succession);
- $a \twoheadrightarrow_w b$ if and only if $a \ggg_w b$ and $b \lll_w a$, (Causality);
- $a \not\equiv_w b$ if and only if $a \lll_w b$ and$b \lll_w a$, (No indirect or direct relation);
- $a \|\|_w b$ if and only if $a \ggg_w b$ and $b \ggg_w a$, (Parallelism).

To illustrate the principle of our basic operators on the log file L:

L= [(A,B,C,D), (A,C,B,D), (E,F)]

- $A \ggg_L C$ because there exists a trace (A,B,C,D) such that A is indirectly succeded by C,
- $E \not\equiv_L D$ because E is never succeded by D and D is never succeded by E
- $B \|\|_L C$ because there exists a trace (A,B,C,D) where $B \ggg_L C$ and there exists a trace (A,C,B,D) where $C \ggg_L B$.

#### 3.1.2 Indirect Succession Matrix

Let L be the log composed of n tasks, and $(A_i)_{i \in n}$ the set of tasks in the log. The indirect succession matrix $(M_{i,j})_{\substack{1 \le i \le n \\ 1 \le j \le n}}$ is a binary Matrix (0, 1) where:

1. $M_{i,j} = 1$ *if* $A_i \ggg_L A_j$;

2. *Else* $M_{i,j} = 0$.

To exemplify the principle,the indirect succession matrix corresponding to the event log L is:

Table 1: Indirect succession matrix for log file L.

| ⋙ | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 | 0 |
| B | 0 | 0 | 1 | 1 | 0 | 0 |
| C | 0 | 1 | 0 | 1 | 0 | 0 |
| D | 0 | 0 | 0 | 0 | 0 | 0 |
| E | 0 | 0 | 0 | 0 | 0 | 1 |
| F | 0 | 0 | 0 | 0 | 0 | 0 |

- $M_{A,C} = 1$ because $A \ggg_L C$,
- $M_{E,D} = 0$ because $E \lll_L D$.

## 3.2 Block Discovery

The block discovery is an issue that we first presented in (Boushaba et al., 2013). Unlike traditional methods which inspect relation between every two tasks of a workflow log,our idea of process discovery is to discover the relation between a block of tasks.

As we define it, two tasks are in the same block if and only if they have the same behavior with the other tasks. Using matrix representation, we can present a formalisation of this concept as follows:

Let L be the log composed of n tasks, $(A_i)_{1\leq i\leq n}$ the set of tasks in the log, and $(M_{i,j})_{\substack{1\leq i\leq n\\1\leq j\leq n}}$the corresponding indirect succession matrix.
Let $I \subsetneq [1,n]$ be the set of the tasks $(A_i)_{i\in I}$is considered as a block if and only if:
$\forall\, i,j \in I\ i<j, \forall\, k \in [1,n] \backslash I\ M_{i,k} = M_{j,k}$(same values in rows).
$\forall\, i,j \in I\ i<j, \forall\, k \in [1,n] \backslash I\ M_{k,i} = M_{k,j}$(same values in columns).

Using the indirect succession matrix for log L presented in (Table 1), we can say that:

- B and C are in block because, out of the red square, tasks B and C have the same values in row and in column.

Now, as the concept of block is well defined, we need to design ate the block type. For this purpose we will present design patterns to determine relations between tasks forming a block.

## 3.3 Design Patterns

This subsection presents the design patterns illustrating relations between tasks in a block using Petri net graphical notation (Murata, 1989).

### 3.3.1 Succession Pattern

The succession pattern represents tasks in succession. Let A, B, C and D be four tasks in succession presented in Petri net as follows:

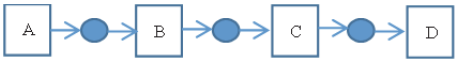


Figure 2: Succession pattern.

The corresponding indirect succession matrix is presented in Table 2:

Table 2: Indirect succession matrix corresponding to succession pattern.

| ⋙ | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 |
| B | 0 | 0 | 1 | 1 |
| C | 0 | 0 | 0 | 1 |
| D | 0 | 0 | 0 | 0 |

### 3.3.2 Parallel Pattern

The parallel pattern represents tasks running in concurrence. A parallel pattern is presented in Petri as:

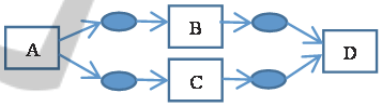


Figure 3: Parallel pattern.

The corresponding indirect succesion matrix is:

Table 3: Indirect succession matrix for the parallel pattern.

| ⋙ | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 |
| B | 0 | 0 | 1 | 1 |
| C | 0 | 1 | 0 | 1 |
| D | 0 | 0 | 0 | 0 |

### 3.3.3 Xor Pattern

The Xor pattern presents tasks running in mutual exclusion. Xor pattern is presented in Petri net as follows illustrated in Figure 4:

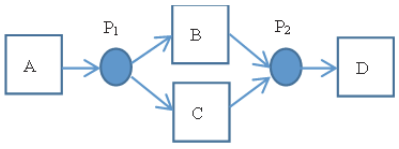


Figure 4: Xor pattern.

The corresponding indirect succesion matrix is presented in Table 4:

Table 4: Indirect succession matrix for Xor Pattern.

| ⋙ | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 |
| B | 0 | 0 | 0 | 1 |
| C | 0 | 0 | 0 | 1 |
| D | 0 | 0 | 0 | 0 |

### 3.4.4 Loop Pattern

Loop pattern presented in the following figure can be seen as a specific Xor patternt (Task B and task C, are in concurrence as they share places $P_1$ and $P_2$).

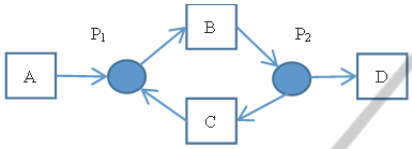


Figure 5: Length two loop.

The corresponding indirect succesion matrix is presented as follows:

Table 5: Indirect succession matrix for the loop Pattern.

| ⋙ | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 |
| B | 0 | 1 | 1 | 1 |
| C | 0 | 1 | 1 | 1 |
| D | 0 | 0 | 0 | 0 |

## 4 DETECTION FILTERS

A block as defined earlier is a set of tasks having the same behavior with all other tasks of the process model.In order to automate blocks detection, we use a combinatorial logic operator that extracts the difference and the similarity between two binary vectors.

## 4.1 Logical Similarity Operator

We use the combinatorial logic operator that gives a score of 1 if the two bits used as input are equal and 0 otherwise. For two tasks, this operator, denoted ($\oplus$), is equivalent to Not Xor operator, whose truth table is given in Table 6:

Table 6: Not Xor truth table.

| A | B | $\overline{A \oplus B}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Where $A \oplus B = A + B \ (mod\ 2)$.

We can generalize our operator to handle more than two tasks as follows:

For a set of tasks$(A_i)_{1 \le i \le n}$:

$$\oplus_{i=1}^{n} A_i = \begin{cases} 0, & \sum_{i=1}^{n} A_i (mod\ n) = 0 \\ 1, & \sum_{i=1}^{n} A_i (mod\ n) > 0 \end{cases}$$

So$\oplus_{i=1}^{n} A_i = 0 \ \ if \ \ \forall i\ A_i = 0\ or\ \forall i\ A_i = 1$

Then$\overline{\oplus_{i=1}^{n} A_i} = 1$ if and only if all $(A_i)_{1 \le i \le n}$ have the same value.

## 4.2 Detecting the First Task

Let L be the log file for a given process model P, and let M be its indirect succession matrix.

**Theorem1:** The task $A_i$ is the first task if and only if the sum of the corresponding column values of the indirect succession matrix equals 0 (i.e.$\sum_{j=1}^{n} M_{j,i} = 0$).

## 4.3 Detecting Last Task

**Theorem2:** The task $A_i$ is the last task if and only if the sum of the corresponding row in the indirect succession matrix equals to 0 (i.e.$\sum_{j=1}^{n} M_{i,j} = 0$)

## 4.4 Detecting Patterns

In this subsection, we explain how we can detect each one of our patterns and loops using logical similarity operator.

### 4.4.1 Succession Pattern

The succession pattern present tasks in succession, by applying logical similarity operator to the indirect succession matrix. results are given in Table 7:

Table 7: Logical similarity operator applied to succession pattern.

| ⋙ | A | B | C | D | $\overline{B \oplus C}$ |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 1 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 1 |
| $\overline{B \oplus C}$ | 1 | 1 | 0 | 1 | |

The application of logical similarity operator to succession pattern gives the following results:

- B and C have the same behavior with task A,

because the index of $\overline{B \oplus C}$ related to A equals to 1 in row and in column (see red cells). The same for D, so B and C form a block.

- If we apply the operator for B, C and D, the index related to A is equal to 1 in row and in column, that means B, C and D are in the same block compared with the task A.

### 4.4.2 Parallel Pattern

The parallel pattern represents tasks in competition (figure 3). By applying the logical similarity operator to its indirect succession matrix we obtain:

Table 8: Logical similarity operator applied to parallel pattern.

| ⋙ | A | B | C | D | $\overline{B \oplus C}$ |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 1 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 1 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| $\overline{B \oplus C}$ | 1 | 0 | 0 | 1 | |

We can conclude from Table 8 that B and C have the same behavior with task A and D (see red cells).

### 4.4.3 Xor Pattern

The Xor pattern represents tasks in mutual exclusion. By applying the logical similarity operator to its indirect succession matrix we obtain

Table 9: Logical similarity operatorapplied to Xor pattern.

| ⋙ | A | B | C | D | $\overline{B \oplus C}$ |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 1 |
| B | 0 | 0 | 0 | 1 | 1 |
| C | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 1 |
| $\overline{B \oplus C}$ | 1 | 1 | 1 | 1 | |

The application of logical similarity operator to Xor pattern gives as result:

- B and C have the same behavior with task A and D (see red cells).
- In addition, $\overline{B \oplus C}$ change signature from one pattern to another (see the green cells in the different pattern). therefore the pattern detection becomes self-evident.

### 4.4.4 Loop Processing

A loop processing can be seen in an event log as repeated treatment in the same trace.For exemple the log $L = [(a, b, c, b, c, d), (a, b, d)]$ contains a loop because the sequence (a, b, c, b, c, d) contains the repeated treatment of the tasks b and c.

To detect the loop pattern we apply the logical similarity operator to its indirect succession matrix, the results are as follows:

Table 10: logical similarity operator applied to the loop pattern.

| ⋙ | A | B | C | D | $\overline{B \oplus C}$ |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 1 |
| B | 0 | 1 | 1 | 1 | 1 |
| C | 0 | 1 | 1 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 1 |
| $\overline{B \oplus C}$ | 1 | 1 | 1 | 1 | |

The differentiation between a loop and an Xor pattern can be made by verifying the apparition of 0 in the four cells of the indirect succession matrix (see blue cells in table 10).

## 5 BLOCK DISCOVERY METHOD

Our approach for block discovery aims to extract block of tasks from an event log. Because each block can be examined independently we're able to reduce the algorithm complexity. We begin by finding the biggest block which is the group of tasks having a maximum number of successors, equivalent in indirect succession matrix, to selecting tasks having row's sum maximal.

### 5.1 Block Detection Steps

The algorithm presented in (Boushaba et al., 2013) is unable to deal with loops; in this paper we extend our method by adding filters (section 4) to differentiate between a loop and an Xor Pattern. To achieve this purpose the following steps are executed:

- First step: compute the indirect succession matrix, and calculate the sum of its rows and the columns;
- Second step: select the group of tasks (two or more) having maximum values in rows sum;
- Third step: for the selected tasks, we apply the logical similarity operator;
- Fourth step: if the selected tasks are in a block, we try to detect its type:
  - First, we look for Xor pattern and loops else we jump to next step;
  - Second, we look for the parallel pattern else we jump to next step;
  - Third, we assume that it is a succession pattern;

- Fifth step: create a frame of discovered Petri Net
- Sixth step: we repeat all steps for the unselected tasks.
- Seventh step: we regroup the result frames of the Petri net

### 5.2 Application for Block Detection

We implemented the algorithm, subject of this paper, in a web application. (Html/ JavaScript) The latter takes as input an event log and returns:

- The corresponding indirect succession matrix,
- First tasks of the process,
- The last tasks of the process,
- And compute the logical similarity operator.

## 6 CASE STUDY

To verify our method we use the following event log which is not representative for real-life event log in terms of size and complexity. However, because of its simplicity, it serves as a suitable illustration of our concepts.

L=[(A,B,D,E,G,H),(A,B,C,B,D,E,G,H),(A,B,D, G,E,H),(A,B,C,B,D,G,E,H),(A,B,D,F,H),(A,B,C, B,D,F,H)(A,B,C,B,C,B,D,F,H)]

We compute the indirect succession matrix (using our application for block discovery presented in subsection 5.2)

Table 11: Indirect succession matrix of L.

| ⋙ | A | B | D | E | G | H | C | F | ∑ |
|---|---|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 |
| B | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 |
| D | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 4 |
| E | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 2 |
| G | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 2 |
| H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 |
| F | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| ∑ | 0 | 3 | 3 | 5 | 5 | 7 | 3 | 4 | |

The application generates the first task (task A) and last task (task H).

We select the group of tasks having (∑row maximal) (A, B, C having ∑row= 7) and we apply the logical similarity operator (We note $\overline{ABC} = A \oplus B \oplus C$):

Table 12: Logical similarity operator applied to tasks A, B and C.

| ⋙ | A | B | D | E | G | H | C | F | ∑ | $\overline{ABC}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 | 0 |
| B | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 | 0 |
| D | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 4 | 1 |
| E | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 1 |
| G | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | 1 |
| H | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| C | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 | 0 |
| F | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| ∑ | 0 | 3 | 3 | 5 | 5 | 7 | 3 | 4 | | |
| $\overline{ABC}$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | |

The selected tasks have the same behavior with the rest of tasks (value 1 in red boxes) which implies that the tasks A, B and C are in a block. To determine the nature of this block, let's extract the sub matrix composed of the three tasks:

Table 13: Sub-matrix related to tasks A, B and C.

| ⋙ | A | B | C | ∑ | $\overline{B \oplus C}$ |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 2 | 1 |
| B | 0 | 1 | 1 | 2 | 1 |
| C | 0 | 1 | 1 | 2 | 1 |
| ∑ | 0 | 3 | 3 | | |
| $\overline{B \oplus C}$ | 1 | 1 | 1 | | |

Tasks B and C have the same behavior with A (value 1 red box), and are in loop or Xor (blue boxes).The green square correspond to a loop pattern. We reduce the matrix by substituting B, C by a unique activity denoted BC:

Table 14: Indirect succession matrix related to tasks A and block B and C.

| ⋙ | A | BC |
|---|---|---|
| A | 0 | 1 |
| BC | 0 | 0 |

The Petri net corresponding to this matrix is


Figure 6: Petri net corresponding to table 17.

By replacing the block BC by the two tasks that are in loop, the previous Petri net becomes:

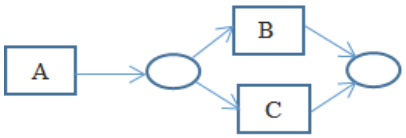


Figure 7: Splitting the block BC.

Now we extract the sub matrix D, E, G, H and F

constituting a block as follows:

Table 15: Indirect succession matrix for L.

| ⋙ | D | E | G | H | F | ∑ | $\overline{E \oplus G}$ |
|---|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 1 | 4 | 1 |
| E | 0 | 0 | 1 | 1 | 0 | 2 | 0 |
| G | 0 | 1 | 0 | 1 | 0 | 2 | 0 |
| H | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| F | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| ∑ | 0 | 2 | 2 | 4 | 1 | | |
| $\overline{E \oplus G}$ | 1 | 0 | 0 | 1 | 1 | | |

We select E and G having an equal value in sum of the rows, by computing the logical similarity operator, we detect that these two tasks are in a parallel block. By substituting tasks E and G by a unique activity EG, our indirect succession matrix becomes:

Table 16: Indirect succession matrix (reduced).

| ⋙ | D | EG | H | F | ∑ | $\overline{EG \oplus F}$ |
|---|---|---|---|---|---|---|
| D | 0 | 1 | 1 | 1 | 3 | 1 |
| EG | 0 | 0 | 1 | 0 | 1 | 1 |
| H | 0 | 0 | 0 | 0 | 0 | 1 |
| F | 0 | 0 | 1 | 0 | 1 | 1 |
| ∑ | 0 | 1 | 3 | 1 | | |
| $\overline{EG \oplus F}$ | 1 | 1 | 1 | 1 | | |

Idem, the tasks EG and F which are in Xor are substituted with a representative activity (EFG):

Table 17: Indirect succession matrix (reduced).

| ⋙ | D | EFG | H | ∑ |
|---|---|---|---|---|
| D | 0 | 1 | 1 | 2 |
| EFG | 0 | 0 | 1 | 1 |
| H | 0 | 0 | 0 | 0 |
| ∑ | 0 | 1 | 2 | |

The matrix corresponds to succession pattern. The Petri representing to the matrix is:


Figure 8: Petri Net corresponding to the Matrix.

As EFG is not an elementary element, we split the block EFG into EG and F having an XOR relation. The Petri net becomes:

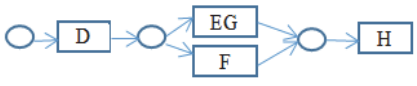


Figure 9: Petri net after splitting the block EFG.

As EG is not an elementary element, we split it into tasks E and G having parallel relation. So, the Petri net becomes:

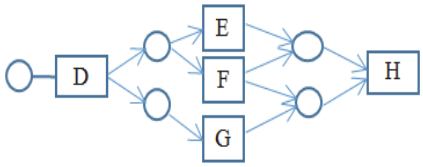


Figure 10: The Petri net after splitting the block EG.

Now, we can substitute the two blocks (ABC and DEFGH) by the representative elements. So the Indirect succession matrix of the log L is:

Table 18: Indirect succession matrix related to block ABC and block DEFGH.

| ⋙ | ABC | DEFGH |
|---|---|---|
| ABC | 0 | 1 |
| DEFGH | 0 | 0 |

The matrix corresponds to succession pattern, and the Petri net corresponding to the matrix is shown in Figure 11:

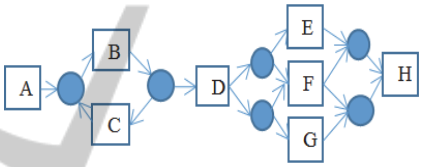


Figure 11: Final petri net.

## 7 CONCLUSION

Our approach for block discovery extends the algorithm presented in (Boushaba et al., 2013). The idea is taking a log file as input to generate a Petri net by creating a matrix representation and reducing its size iteratively. Filters are added to automate detection and loops are also handled.

The algorithm presented in this paper is more efficient than those presented in (Van Der Aalst et al., 2004) and (De Medeiros et al., 2003) in the sense that it is not necessary to go through the whole log file to discover each place and each transition of the discovered Petri net. Additionally, our algorithm is characterized by its ability to be parallelized in order to discover different blocks. Another advantage of our algorithm is that by dividing the log into sub logs reduces the whole complexity as it is faster to process each sub log separately than processing the entire log. Finally, our algorithm is O ($n^2$), which makes it appropriate to deal with large event logs (real logs) that can be composed of thousands activities unlike the existing ones (Van der Aalst,

2012).

In order to enrich our research, we intend to introduce the partial block concept, which should allow us to deal with incomplete logs. The application will be improved to generate all relationships and the discovered Petri Net.

## REFERENCES


Agrawal, R., Gunopulos, D., and Leymann, F. 1998. Mining Process Models from Workflow Logs. *Sixth International Conference on Extending Database Technology*, pages 469–483.

Van Der Aalst, W. M. P., Weijters, A. J. M. M., and Maruster L., 2004. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142.

De Medeiros, A. K. A. ,Van Der Aalst, W. M. P., and Weijters, A. J. M. M. 2003.Workflow Mining: Current Status and Future Directions Department of Technology Management, Eindhoven University of Technology P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, Springer,

Weijters, A. J. M. M., Van Der Aalst, W. M. P., and De Medeiros, A. K. A., 2006. *Process Mining with the Heuristics Miner Algorithm, BETA Working Paper Series*, WP 166 Eindhoven University of Technology, Eindhoven.

Boushaba, S., Kabbaj, M. I., Bakkoury, Z., 2013. Process mining: Matrix representation for bloc discovery, *Intelligent Systems: Theories and Applications (SITA), IEEE*.

Murata, T., 1989. Petri Nets: Properties, Analysis and Applications, *Apr. Proc. IEEE*, vol. 77, no. 4, pp. 541-580.

De Medeiros, A. K. A., Weijters, A. J. M. M., and Van Der Aalst, W. M. P, 2004.Using Genetic Algorithms to Mine Process Models: Representation, Operators and Results, *Eindhoven University of Technology*, Eindhoven.

Chen, Li., Manfred, R., and Wombacher, A., 2009. Discovering Reference Models by Mining Process Variants Using a Heuristic Approach.

IEEE Task Force on Process Mining: Process Mining Manifesto, 2012. In: *BPM Workshops. LNBIP*, vol. 99, pp. 169–194. Springer

Van der Aalst, W. M. P. 2011. Process Mining: Discovery, Conformance and Enhancement of Business Processes. *Springer*.

Chen, Li., Manfred, R., Andreas, 2011, Mining Business Process Variants: Challenges, Scenarios, Algorithms, Data & Knowledge Engineering Elsevier,

Wen L., Van der Aalst W. M. P., Wang J., and Sun J., 2007, Mining Process Models with Non-free- Choice Constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180.

Weijters, A. J. M. M. and van Der Aalst, W. M. P. 2003. Rediscovering Workflow Models from Event-Based Data Using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162.

Van der Aalst W. M. P., 2012. Desire Lines in Big Data, In proceeding of: Promoting Business Process Management Excellence in Russia (*PropelleR 2012*) 11-12.

Leemans, S. J. J., Fahland, D., van der Aalst, W.M.P. 2013. Discovering block-structured process models from event logs -a constructive approach. In: *Petri Nets. Lecture Notes in Computer Science*, vol. 7927, pp. 311–329. Springer

Leemans, Sander J. J., Fahland D. and van der Aalst W. M. P., 2013. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. fluxicon.com