A Polynomial-Time Alpha-Algorithm for Process Mining

Albana Roci
albana.roci@uis.no

Reggie Davidrajuh
reggie.davidrajuh@uis.no

Electrical and Computer Engineering
University of Stavanger
Stavanger, Norway

***Abstract* — This paper presents an efficient Alpha-algorithm for process mining. Firstly, a short literature review is given on the available algorithms for process mining. Secondly, a new polynomial-time algorithm is proposed, which is efficient as it takes only O(LT2 + T3) time, where L and T are the number of traces (rows) of the event log and the total number of unique tasks, respectively. Also, this algorithm is elegant and easy to understand as it mainly manipulates four simple matrices (such as Serial Binary Matrix, Parallel Binary Matrix, input incidence matrix, and output incidence matrix). The operations performed on these matrices are also simple. Thirdly, a case study is given as a proof-of-concept showing the Alpha-algorithm in action. In the case study, from a medium-sized event log, a Petri net model is generated by the proposed algorithm, which is implemented using the General-purpose Petri Net Simulator (GPenSIM).**

Keywords - Alpha-algorithm; Process Mining; Petri Nets; GPenSIM

I. INTRODUCTION

Nowadays, there are more enterprises that are creating massive data and event logs, which are generated by complex business and manufacturing processes. From the event logs, to understand the underlying processes and to improve them, it is necessary to create the abstraction of the processes, in other terms - the model. The field that deals with this kind of problem is Process Mining (PM). The main purpose of process mining is to discover meaningful insight into the processes and to make a graphical model of it so that the model can be used to improve the processes [1]. Process mining has the following three main steps: A) Process discovery, B) Conformance and C) Performance Analysis [2]. This paper is on the first step, proposing a new algorithm that will create an abstract model of the system based on the event logs. Conformance, which is the second step of process mining, is to check whether the derived model satisfies the data. While the third step of PM is to improve the generated model.

Literature study reveals several algorithms that are used to generate the abstraction of the model in the discovery step. One of them is α-algorithm and its extensions [3]. This paper proposes an efficient α-algorithm that takes polynomial (more specifically, cubic) time. The proposed α-algorithm is also elegant as it mainly manipulates matrices.

In this paper: section-II presents a short literature review. Section-III presents the new Alpha-algorithm. Section-IV presents a simulation study as a proof-of-concept. A discussion of the benefits and shortcomings of the proposed algorithm are given in Section-V.

II. LITERATURE REVIEW

In addition to the α-algorithms, literature study also reveals some other algorithms for the discovery of models, such as Genetic Algorithms [4, 5], Language Based Region algorithm [6, 7], and State Discovery algorithm [8]. α-algorithm operates based on the relationships between the tasks in the event logs. The basic α-algorithm deals with the choice, the parallel and the sequential relationships between tasks [3]. The extensions of α-algorithm such as α+ algorithm which deals with short-loop and α++ algorithm deals with non-free choice contraction [9, 10]. Further, α# algorithm is proposed to deal with invisible transitions, while α* algorithm is intended to solve the problems by duplicated transitions [11, 12].

Nevertheless, none of the algorithms mentioned above is complete in the sense that they fail to solve all the problem cases of process mining. In order for an algorithm to work, it is essential that the model should have one unique starting place and one unique ending place. In addition, the event logs should be complete and free from noise, something not possible achieve in the event logs from companies.

There are many process modelling tools (such as UML, BPMN, EPCs, etc.) with which it can create models in process mining [13, 14]. In this research, Petri Net is chosen as the modelling tool due to its graphical nature and its other useful properties such as explicit models and industry acceptance [15, 16]. Petri Net (PN) is a mathematical tool that is used to model the discrete event systems.

The fundamental elements of a Petri Net model are transitions, places and the arcs (connections between the places and the transitions). The transitions present the active components (e.g., machines) in a discrete-event system, and

the places represent the passive components (e.g., buffers). Figure 1 illustrates a Petri Net model.

This paper uses the tool GPenSIM to implement Petri Net models on the MATLAB platform. GPenSim is a user-friendly tool developed by the second author of this paper [17] and is considered as a flexible tool for modelling and simulation of large-scale industrial systems [18, 19].

III. A New Algorithm

A process is a composition of different activities or tasks. Activities are interlinked with each other either in the sequential, parallel or choice routing. For example in figure 1, task *g* and task *h* are in sequential routing, task *j* and task *k* are in choice routing while task c is in parallel with task *d* and task *e*. To translate the event logs into the process model, it is important to distinguish one of those types of the connections.

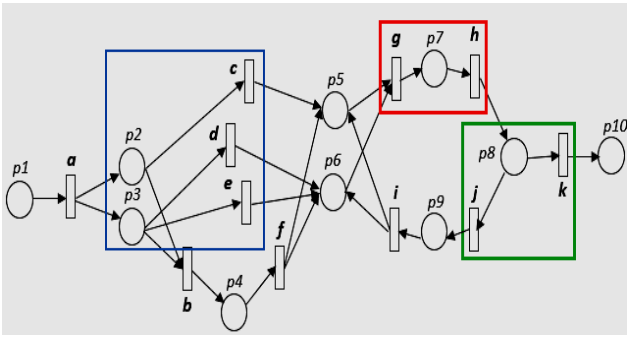

Figure 1. The process model of event logs in table 1

As already mentioned, the algorithm is implemented using the GPenSim tool. GPenSIM provides input and output incident matrices of the Petri Net models. Input incidence matrix represents the connections between the input places to a transition, whereas output incidence matrix shows the connections to the output places. The following parts of this section will introduce the algorithm of converting the event logs to the Petri Nets model. It will continue with the restraint of generating the models that contain short-loops and the suggestions to these problems.

A. The Proposed Algorithm

The main idea of the algorithm is to convert the given event logs to the representative sequential and parallel matrices and in the end to transform them into the incident matrices.

The overall representation of the algorithm is shown above:

- Process the event logs: read the input event log and register each distinct task.
- Find the tasks: find the preceding and the succeeding tasks for each of the given tasks.
- Generate the sequential and the concurrent binary matrices
- Generate the incidence matrices
- Convert the incident matrices into the Petri Net model

A1. Processing the Event Logs

As an illustration, it will consider the event logs shown in table 1. This example process contains eleven different activities, and they take part in eight distinct event logs.

TABLE 1. EVENT LOGS OF THE ARTIFICIAL PROCESS.

EVENT LOGS
<*a, b, f, g, h, k*>
<*a, c, e, g, h, k*>
<*a, d, c, g, h, j, i, g, h, k*>
<*a, c, d, g, h, k*>
<*a, b, f, g, h, j, i, g, h, k*>
<*a, d, c, g, h, k*>
<*a, e, c, g, h, k*>
<*a, e, c, g, h, j, i, g, h, k*>

A2. Finding the Tasks

From the given event log, for each of the tasks, the following are registered in a table: 1) the immediately preceding task, and 2) the immediate succeeding task; see table 2. The tasks that do not have any predecessor are considered as starting tasks while the ones that do not have any successor are consider as ending tasks.

TABLE 2. PRECEDING AND SUCCEEDING TASKS FOR EACH OF THE TASKS.

	Input	***output***
a	‘’	*‘b, c, d, e’*
b	*‘a’*	*‘f’*
c	*‘a, d, e’*	*‘d, e, g’*
d	*‘a, c’*	*‘c, g’*
e	*‘a, c’*	*‘c, g’*
f	*‘b’*	*‘g’*
g	*‘c, d, e, f, i’*	*‘h’*
h	*‘g’*	*‘j, k’*
i	*‘j’*	*‘g’*
j	*‘h’*	*‘i’*
k	*‘h’*	‘’

TABLE 3. SEQUENTIAL BINARY MATRIX (SBM)

	a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***
a	0	1	1	1	1	0	0	0	0	0	0
b	0	0	0	0	0	1	0	0	0	0	0
c	0	0	0	0	0	0	1	0	0	0	0
d	0	0	0	0	0	0	1	0	0	0	0
e	0	0	0	0	0	0	1	0	0	0	0
f	0	0	0	0	0	0	1	0	0	0	0
g	0	0	0	0	0	0	0	1	0	0	0
h	0	0	0	0	0	0	0	0	0	1	1
i	0	0	0	0	0	0	1	0	0	0	0
j	0	0	0	0	0	0	0	0	1	0	0
k	0	0	0	0	0	0	0	0	0	0	0

A3. Generating the SBM and PBM

Based on table 2, two different binary matrices are generated, known as the sequential and the parallel binary matrices (SBM and PBM, respectively). Their dimensions are *n* x *n* where *n* is the number of tasks. In the SBM, the rows indicate the succeeding tasks and the columns the preceding tasks, shown table 3. For example, task a is succeeded by c, thus *SBM(a, c) = 1*. Also, task c is succeeded by g, thus *SBM(c, g) = 1*.

Table-2 depicts the parallel tasks. For example, task d is both a predecessor and a successor to task c. This means tasks c and d are in parallel. If the tasks are in the parallel mode, then they will be registered in the PBM, as shown in table 4. For example, tasks c and d are in parallel to each other. Thus, the entries in the PBM are *PBM(c, d) = 1* and *PBM(d, c) = 1*.

After creating the SBM and the PBM matrices, the next (and the most important) step is to convert these matrices into the incident matrices. The difficulty in this process is to make a distinction between the parallel and the choice routing. To visit all the tasks, Breadth First Search (BFS) algorithm is used. It will initiate with one of the starting tasks and will proceed with the successors, and it will continue until the terminal task is hit. The successors can have parallel and/or choice routing.

TABLE 4. PARALLEL BINARY MATRIX (PBM).

	a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***
a	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0	1	1	0	0	0	0	0	0
d	0	0	1	0	0	0	0	0	0	0	0
e	0	0	1	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	0

For instance, task *a* is succeeded by four different tasks {*b, c, d, e*}. These four tasks can have different relationships between them. If the tasks are in choice routing, just one place is necessary for all the outgoing transitions. However, if they are in parallel, different places are needed.

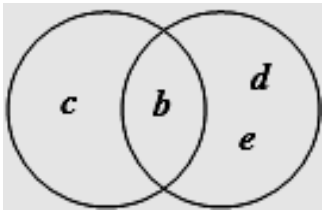

Figure 2. The group division between the {b, c, d, e} activities

In input and output incident matrices, each column denotes a place. For instance, in PBM shown in table-4, the following entries possess true values: PBM(c, d), PBM(d, c), PBM(c, e), and PBM(e, c). This means that task *d* and task *e* are in the choice mode, and these two tasks are in parallel with task c, while task b is in choice with all of them. Figure 3 shows the group division between the tasks. This means that there are just two different places needed that will divide task c from tasks {d, e} and task b are connected by both of the places.

A4. Generating the Incidence Matrices

In both incidence matrices will be added two columns. In the input incidence matrix, one column will provide information about *b* and *c* tasks and the other column for *b, d* and *e*, as shown in table 5 (p2 and p3 columns). *p2* and *p3* columns in output incidence matrix will keep information just for the task *a.* In the meantime, it checks if there is any other task that can be on the choice routing with task *a.* On condition that, if they are in choice routing, they need to use the same places. For instance, when task *c* is generated, it will proceed with its successors, in this case, task *g.* In the meantime, it will check if there are some other tasks that income to task *g*. From the *g* column of the SBM, there are found tasks {b, d, e, i}. It will group the tasks as shown in figure 3 and will add the needed columns in the incident matrices.

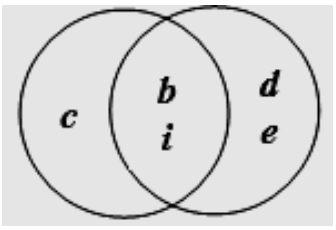

Figure 3. The group division between {b, c, d, e, i} activities

Based on the event logs of table 1, the following incident matrices are generated, shown in table 5 and table 6. They contain eleven rows, and ten different columns are discovered.

TABLE 5. INPUT INCIDENCE MATRIX

	p1	***p2***	***p3***		***p4***	***p5***	***p6***	***p7***	***p8***	***p9***	***p10***
a	1	0	0		0	0	0	0	0	0	0
b	0	1	1		0	0	0	0	0	0	0
c	0	1	0		0	0	0	0	0	0	0
d	0	0	1		0	0	0	0	0	0	0
e	0	0	1		0	0	0	0	0	0	0
f	0	0	0		1	0	0	0	0	0	0
g	0	0	0		0	1	1	0	0	0	0
h	0	0	0		0	0	0	1	0	0	0
i	0	0	0		0	0	0	0	0	1	0
j	0	0	0		0	0	0	0	1	0	0
k	0	0	0		0	0	0	0	1	0	0

TABLE 6. OUTPUT INCIDENCE MATRIX

	p1	*p2*	*p3*	*p4*	*p5*	*p6*	*p7*	*p8*	*p9*	*p10*
a	0	1	1	0	0	0	0	0	0	0
b	0	0	0	1	0	0	0	0	0	0
c	0	0	0	0	1	0	0	0	0	0
d	0	0	0	0	0	1	0	0	0	0
e	0	0	0	0	0	1	0	0	0	0
f	0	0	0	0	1	1	0	0	0	0
g	0	0	0	0	0	0	1	0	0	0
h	0	0	0	0	0	0	0	1	0	0
i	0	0	0	0	1	1	0	0	0	0
j	0	0	0	0	0	0	0	0	1	0
k	0	0	0	0	0	0	0	0	0	1

B. Running Time

Let ***L*** be the number of traces (rows) of the event log and the ***T*** be the total number of unique tasks. As shown in section-III.A, the algorithm is broken into a number of routines. Let us find the time taken by the individual routines:

- Processing the event logs takes $O(L \times T)$ time.
- Finding the preceding and the succeeding tasks for each of the given tasks takes $O(L \times T^2)$ time.
- Generating the SBM and the PBM matrices takes $O(T^2)$ time.
- Generating the incidence matrices takes $O(T^3)$ time.
- Converting the incident matrices into the Petri Net model takes $O(T^2)$.

Hence, the new algorithm takes a polynomial (cubic) time of $O(LT^2 + T^3)$.

C. Short Loops: A Shortcoming of the Algorithm

The new algorithm is incapable of dealing with the so-called "short-loops." Two different examples of short-loops are shown in Figures 5 and 7. Figures 4 and 6 illustrate the models generated by the proposed algorithm. In the first example (Figure 4), according to the event logs, task *d* and task *e* are inside a loop. Therefore, they are preceding and succeeding tasks of each other, and the algorithm will detect as parallel routing. If there are in parallel routing, they will fire in the same time that means of incorrect model detection. The correct model is shown in Figure 5.

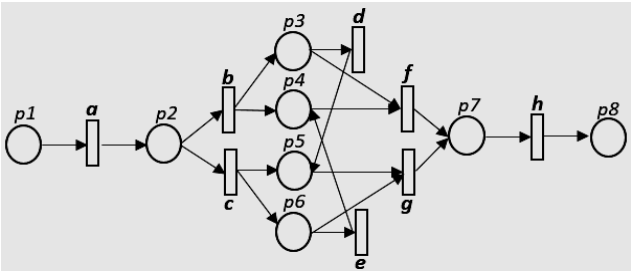

Figure 4. The wrong generated model from the following event logs {<a, b, d, g, h>, <a, b, f, h>, <a, b, d, e d, e, f, h>, <a, c, g, h>, <a, c, e, d, g, h>, <a, c, e, d, e, f, h>}

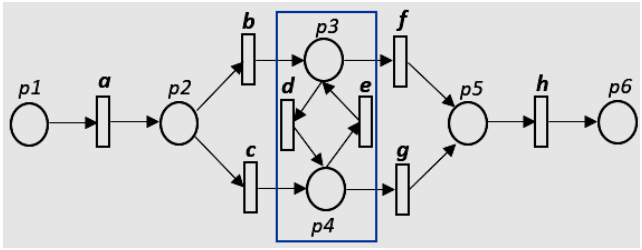

Figure 5. The correct generated model by the above event logs

While in Figure 6, the task *d* can fire several times consecutively. The algorithm can detect the loop, but it considers the task *d* to be in parallel with the outgoing tasks. Therefore, it generates the incorrect model shown in Figure 5, which is corrected in Figure 7.

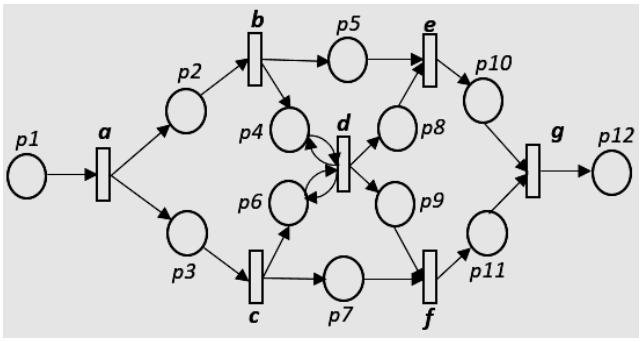

Figure 6. The wrong generated model from the following event logs {<a, b, c, d, d, d, e, f, g>, <a, c, b, d, d, d, e, f, g>, <a, c, b, d, f, e, g> <a, c, b, f, e, g>, <a, b, c, f, e, g>, <a, c, b, e, f, g>, <a, b, c, e, f, g>, <a, b, e, c, f, g>, <a, c, f, b, e, g>}

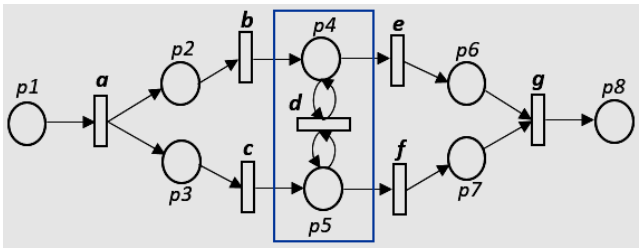

Figure 7. The correct generated model by the above event logs

To deal with short-loops, the algorithm is extended by additional functions. Considering the first case, the algorithm detects consecutive preceding and succeeding relationships between the tasks. Hence, it will create two different types of places for those transitions. In addition, it should also identify the preceding task of the first task and the succeeding task of the second task. Also, it should determine their groups and then join them. Figures 5 and 8 show that the algorithm has detected and corrected the output models.

In the second case (Figure 6), the algorithm will determine whether the task is consecutively repeating. Then, it detects the places of all preceding and succeeding tasks from the incident matrices. Then the task will be added as

an output transition to the places that are in choice routing with the other succeeding tasks. The same method will be applied for output places of the repeating transitions. The figures 7 and 9 illustrate the corrected models by the extension of the algorithm.

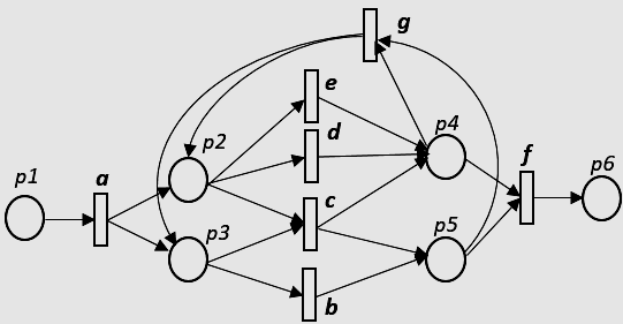

Figure 8. The Petri Net model of the following event logs
{<a, b, f>, <a, c, d, f>, <a, c, e, f>, <a, d, c, f>, <a, e, c, f>, <a, b, g, b, f>, <a, b, g, c, d, f>, <a, d, c, g, c, d, f>, <a, c, e, g, b, f>, <a, c, e, g, e, c, f>, <a, e, c, g, d, c, f>, <a, b, g, c, e, f>, <a, c, d, g, d, c, f>}

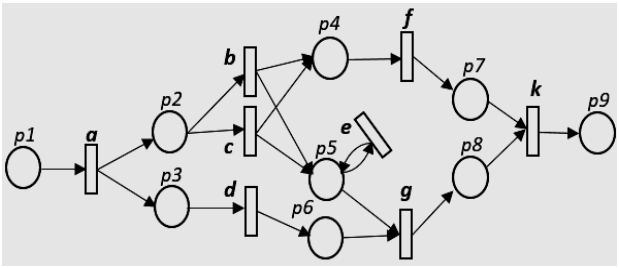

Figure 9. The Petri Net model of the following event logs
{<a, b, d, e, f, g, k>, <a, d, b, g, f, k>, <a, d, b, e, e, e, g, f, k>, <a, b, f, e, d, g, k>, <a, b, f, d, e, g, k> <a, c, d, f, e, g, k>, <a, d, c, f, e, e, e, g, k>, <a, d, c, e, e, g, f, k>, <a, d, c, g, f, k>}

IV. Case Study

This section demonstrates implementation of the proposed algorithm. The implementation is done with GPenSIM [20]. Due to brevity, only parts of the main files are shown in this section. However, the interested reader can download the complete files from the website [21], for the full reproduction of the results shown in this section.

A Petri Net model realized with GPenSIM consists of a set of M-files: the main simulation file (MSF), the Petri Net definition File (PDF), and the processor files. Figure 10 shows the GPenSIM code that generates the PDF file of the first example. The input and output incident matrices generate this PDF file. The generated PDF file is shown in Figure 11.

```
PDF_Filename = 'pm_pdf.m';
PN_name = 'PN Model!';
createPDF(global_info.input, global_info.output,...
    PDF_Filename, PN_name);
pns = pnstruct('pm_pdf');
mfs();
```

Figure 10. The code that generates PDF file

```
% This PDF file was generated by "createPDF" function on
% On 11-Sep-2018   at 13:29:46
% PDF: pm_pdf.m

function [png] = pm_pdf()

png.PN_name = 'PN Model!';
png.set_of_Ps = {'p1','p2','p3','p4','p5','p6','p7','p8','p9'};
png.set_of_Ts = {'t1','t2','t3','t4','t5','t6','t7','t8'};
png.set_of_As = {'p1','t1',1, 't1','p2',1, 't1','p3',1, ... % t1
                 'p2','t2',1, 't2','p4',1, 't2','p5',1, ... % t2
                 'p2','t3',1, 't3','p4',1, 't3','p5',1, ... % t3
                 'p3','t4',1, 't4','p6',1, ... % t4
                 'p4','t5',1, 't5','p4',1, ... % t5
                 'p5','t6',1, 't6','p7',1, ... % t6
                 'p4','t7',1, 'p6','t7',1, 't7','p8',1, ... % t7
                 'p7','t8',1, 'p8','t8',1, 't8','p9',1, ... % t8
                };
```

Figure 11. PDF file that contains the information for PN model

As discussed in section-III, there are some processes for which the algorithm is not able to generate the correct model. Thus, by analysing the main properties of the Petri Nets model, we can improve the correctness of the model.

```
dyn.ft={'allothers', 1};
dyn.m0={'p1',1};
pni = initialdynamics(pns, dyn);
cotree(pni, 1, 1);
```

Figure 12. The code that generates Coverability Tree

Coverability Tree is one of the methods that can show some of the required properties of the Petri Net model. The coverability tree can be used to check the properties such as Liveness, Safeness and reachability. Figure 13 shows the coverability tree generated by the GPenSIM code that is demonstrated in figure 12. From the coverability tree (e.g., figure 13), it is easy is to determine whether if the model suffers from deadlocks, the safeness, and if the model is over-fitted or under-fitted [22].

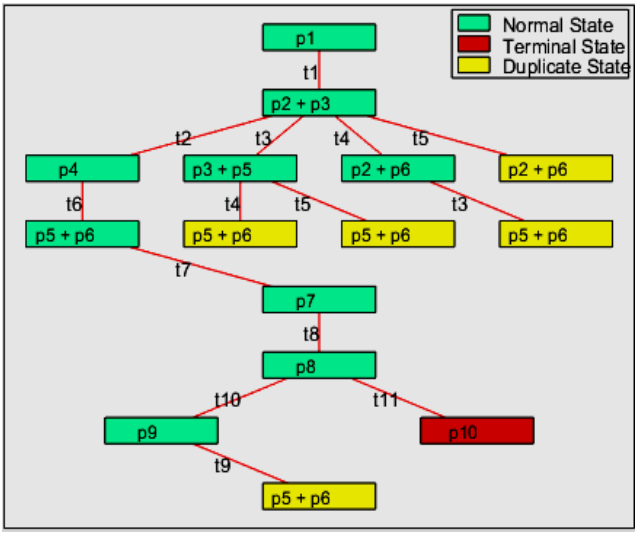

Figure 13. Coverability Tree

```
Boundedness:
p1 : 1
p10 : 1
p2 : 1
p3 : 1
p4 : 1
p5 : 1
p6 : 1
p7 : 1
p8 : 1
p9 : 1

Liveness:
Terminal States: [14]
```

Figure 14. Some part generated by Coverability Tree code

V. Conclusion

The novelty of the proposed algorithm in this paper are: a) its running time, and b) easy to understand.

As shown in section-III, the new algorithm takes a polynomial (cubic) time of $\boldsymbol{O(LT^2 + T^3)}$, where $\boldsymbol{L}$ and $\boldsymbol{T}$ are the number of traces (rows) of the event log and the total number of unique tasks, respectively. Also, the algorithm is compact, elegant, and can be easily understood as it manipulates four simple matrices (such as Serial Binary Matrix, Parallel Binary Matrix, input incidence matrix, and output incidence matrix). The operations performed on these matrices are also simple, limited to linear searching.

In comparison, the alpha-algorithm proposed in [23] takes non-polynomial time; in this algorithm, the footprint matrix takes the central stage. Repeatedly manipulating this footprint matrix using the operations '→' (succeeding element), '←' (preceding element), and '#' (never follow or no relationship), and frequently rearranging it makes the algorithm not only complicated but also non-polynomial timed. In the proposed algorithm described in this paper, the complexity is reduced by using the four simple matrices (such as SPM, PBM, input incidence matrix, and output incidence matrix) in parallel. Also, some simple operations (such as insertion, deletion, and linear search) are applied to these matrices, one at a time. This kind of "divide-and-conquer" approach makes the proposed algorithm elegant, easy to understand, simple to implement, and finally, polynomial timed.

Prospects of the proposed algorithm: In the case study given in this paper, the algorithm is implemented with GPenSIM, proving the usefulness of the algorithm. Though the case study describes a medium-sized event log for brevity, the real benefit of this algorithm will come to light if very-large-scale (big data) event logs are fed into the algorithm; due to the cubic running time, the algorithm will process very-large event logs efficiently. Since GPenSIM is a toolbox on the MATLAB platform, the implementation code of the algorithm can be automatically converted into C/C++ code, using the MATLAB Coder [24]. With this, there is potential to incorporate the code into future commercial packages.

References

[1] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," ACM Transactions on Software Engineering and Methodology, vol. 7, no. 3, pp. 215-249, 1998.

[2] M. Leemans and W. M. P. Van der Aalst, "Process Mining in Software Systems: Discovering Real-Life Business Transactions and Process Models from Distributed Systems," 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS) , pp. 44-53, 2015.

[3] B. F. Van Dongen, A. K. Alves de Medeiros and L. Wen, "Process Mining: Overview and Outlook of Petri Net Discovery Algorithms," Transactions on Petri Nets and Other Models of Concurrency II, vol. 5460, pp. 225-242.

[4] W. M. P. Van der Aalst, A. K. Alves de Medeiros and A. J. M. M. Weijters, "Genetic Process Mining," Applications and Theory of Petri Nets 2005 , pp. 48-69, 2005.

[5] A. K. A. De Medeiros, A. J. M. M. Weijters and W. M. P. van der Aalst, "Genetic process mining: an experimental evaluation," Data Mining and Knowledge Discovery, vol. 14, no. 2, pp. 245-304, 2007.

[6] R. Bergenthum, J. Desel, R. Lorenz and S. Mauser, "Process Mining Based on Regions of Languages," International Conference on Business Process Management, pp. 375-383.

[7] R. Lorenz, S. Mauser and G. Juhás, "How to Synthesize Nets from Languages - a Survey," Proceedings of the Wintersimulation Conference, 2007.

[8] E. Kindler, V. Rubin and W. Schäfer, "Process Mining and Petri Net Synthesis," Business Process Management Workshops , pp. 105-116, 2006.

[9] A. Alves De Medeiros, B. van Dongen, W. Van der Aalst and A. Weijters, "Process mining : extending the alpha-algorithm to mine short loops," Eindhoven: Technische Universiteit Eindhoven., vol. 113.

[10] L. Wen, W. M. P. van der Aalst, J. Wang and J. Sun, "Mining process models with non-free-choice constructs," Data Mining and Knowledge Discovery, vol. 15, pp. 145-180, 2007.

[11] L. Wen, J. Wang, W. M. P. van der Aalst, B. Huang and J. Sun, "Mining Process Models with Prime Invisible Tasks," Data & Knowledge Engineering, vol. 69, no. 10, pp. 999-1021, 2010.
[12] J. Li, D. Liu and B. Yang, "Process Mining: Extending α-Algorithm to Mine Duplicate Tasks in Process Logs," International Conference on Web-Age Information Management, pp. 396-407, 2007
[13] R. M. Dijkman, M. Dumas and C. Ouyang, "Semantics and analysis of business process models in BPMN," Information and Software Technology, vol. 50, no. 12, pp. 1281-1294, 2008.
[14] M. Dumas, W. M. P. Van der Aalst and A. H. M. Ter Hofstede, Process-aware information systems: Bridging People and Software Through Process Technology, Canada: Wiley - INTERSCIENCE, 2005.
[15] J. L. Peterson, Petri nets. ACM Computing Surveys (CSUR), 1977, 9(3), 223-252.
[16] T. Murata, "Petri nets: Properties, analysis and applications," in Proceedings of the IEEE, vol. 77, no. 4, pp. 541-580, April 1989. doi: 10.1109/5.24143
[17] R. Davidrajuh, Modeling Discrete-Event Systems with GPenSIM: An Introduction, Springer International Publishing, 2018.
[18] A. Cameron, M. Stumptner, N. Nandagopal, W. Mayer, and T. Mansell, "Rule-based peer-to-peer framework for decentralised real-time service oriented architectures," Science of Computer Programming, 2015, 97, 202-234.
[19] Lopez F., Barton K., Tilbury D., "Simulation of Discrete Manufacturing Systems with Attributed Hybrid Dynamical Nets," Unpublished, 2018.
[20] R. Davidrajuh, GPenSIM: A general purpose Petri net simulator, URL: http://www.davidrajuh.net/gpensim
[21] Website for the complete simulation Code: http://davidrajuh.net/gpensim/Pub/2018/IJSSST-Sep/
[22] W. M. P. Van der Aalst, V. Rubin, H. M. W. Verbeek, B. F. Van Dongen, E. Kindler and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting," Software & Systems Modeling, 2010
[23] W. van der Aalst, Process Mining: Data Science in Action, Berlin Heidelberg: Springer-Verlag Berlin Heidelberg, 2016.
[24] MATLAB Coder: Available: https://se.mathworks.com-/products/matlab-coder.html