Journal of Data Analysis
April 2013, Vol.8 No.2, pp.1-30

Process Mining: Process Model from Event Logs

Dennis K. J. Lin[1]* Yun-Shiow Chen[2] Min-Hsun Kuo[3]

Abstract

Modeling a workflow design is a complicated and time-consuming process in today's competitive market. It has received a great deal of attention in many fields, such as software engineering and workflow management. Process mining is such a technique to analyze the stream-data from the workflow process. Modern information technologies allow us to collect complete global stream-data in an efficient manner. Process mining helps in understanding the actual process from these stream-data. In this paper, we develop an algorithm for process mining by modifying the α-algorithm to handle complex activity relationships involving concurrence and alternative. The detailed procedure of the proposed algorithm is discussed, with an example for a thorough illustration. A real-life case study is provided, and comparison with existing algorithms is also made. It is shown that our proposed method can handle more complicated situations than the existing methods.

Keywords: α-algorithm, Petri nets, Process model, Stream-data.

---

[1] Department of Statistics, Pennsylvania State University, University Park, USA and Department of Statistics, Fudan University, Shanghai, China.
[2] Department of Industrial Engineering and Management, Yuan Ze University, Taoyuan, Taiwan.
[3] Department of Industrial Engineering and Management, Yuan Ze University, Taoyuan, Taiwan.
*Correspondence to : Dennis K. J. Lin, *Department of Statistics, 317 Thomas Building, Pennsylvania State University, University Park, PA 16802-2111, USA.* E-mail: DKL5@psu.edu
Manuscript received: 2012.9.7 ; Revised: 2012.11.26 ; Accepted: 2013.3.1

1. INTRODUCTION

Process mining is a process management technique which analyzes business processes based on event logs. It is often used when no formal depiction of the process can be obtained by other approaches, or when the quality of an existing documentation is doubtful. This is a relatively new area, but has increasingly received attentions in various fields; for example, the audit trails of a workflow management system, the transaction logs of an ERP system, and the log of the exchange of messages with other parties of Business-to-business (B2B) systems [17].

Take Table 1 as an example. The business processes are first collected and tabulated in an event logs data. An event logs table basically consists of two columns (contributes), case number and task number, as shown in Table 1 (adapted from [7]). All cases are then summarized into traces, as shown in Table 2. This can be easily done. Based upon all these traces, the ultimate goal here is to build up a process model describing all workflows. One solution is the one shown in Figure 1, which is capable of capturing all traces in Table 2.

For the basic utilization of a process model, Humphrey et al.[11] introduced four desirable properties, including enabling effective communication regarding the process, facilitating process reuse, supporting process evolution and facilitating process management. They suggested that a process model should be able to (1) represent the way the work is actually preformed; (2) provide a flexible and understandable, yet powerful, framework for representing and enhancing the process; and (3) be refinable to whatever level of detail is needed. The relationship of activities can be classified into four basic patterns: sequent, concurrence, alternative, and loop [7, 20]. A relationship that involves concurrence and alternative simultaneously is called an "option". An option is rather common in practice but is difficult to identify using most existing algorithms. Our objective here is to propose an algorithm for building up an understandable process model

which is able to (a) incorporate all traces contained in event logs and (b) deal with the problem of the identifying options.

This paper is organized as follows. In Section 2, a literature review of the process mining is presented. Section 3 proposes a new algorithm for process mining with detailed descriptions. An example to illustrate the proposed method is presented in Section 4. Section 5 presents a case study from a hospital in Taiwan. Both the proposed algorithm and the traditional algorithm are implemented and compared. Discussion and comparisons with other existing algorithms are given in Section 6. Section 7 provides a conclusion and future work.

Table 1. Sample event logs

Case identifier	Task identifier
Case 1	Task A
Case 2	Task A
Case 3	Task A
Case 3	Task B
Case 1	Task B
Case 1	Task C
Case 2	Task C
Case 4	Task A
Case 2	Task B
Case 2	Task D
Case 5	Task E
Case 4	Task C
Case 1	Task D
Case 3	Task C
Case 3	Task D
Case 4	Task B
Case 5	Task F
Case 4	Task D

Table 2. Sample Events Logs Data (by Case)

Case 1: A→B→C→D	{ABCD}
Case 2: A→C→B→D	{ACBD}
Case 3: A→B→C→D	{ABCD}
Case 4: A→C→B→D	{ACBD}
Case 5: E→F	{EF}

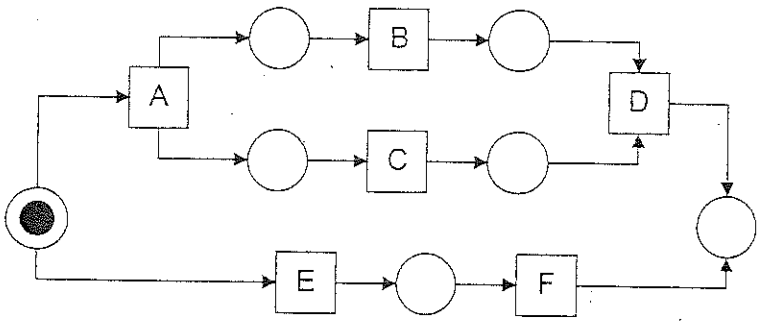

Figure 1. A process model corresponding to the process logs in Table 1 [7]

2. LITERATURE REVIEW

Process mining was initially applied in software engineering to design a realization of the performance processes of a system. Cook and Wolf [3, 4] developed three methods (RNet, Ktail, and Markov) based on grammar inference. These proposed methods are limited, merely suitable to handle a process model with sequential. Cook and Wolf [5] later added four matrices (entropy, event type counts, periodicity, and causality) to model concurrent behavior. Agrawal *et al.* [1] applied process mining in the context of workflow management. For process mining, Van der Aalst *et al.* [2, 6, 18, 19] presented eleven challenges for process model mining. Some of them remain unsolved.

The α-algorithm is based upon the relationship of activities and represents the process in Petri net language. The α-algorithm has been applied in supporting security efforts at various levels, ranging from low-level instruction detection to high-level fraud prevention [16]. Recently, Van der Aalst *et al.*[17] used the α-algorithm to capture concurrency in business processes. Many modified α-algorithms have been developed. Van der Aalst *et al.* [7] improved the α-algorithm to the α+-algorithm for handling short-loops. The α++-algorithm was developed for dealing with more complex problems such as non-free-choice constructs. The process mining software "Little thumb" was proposed by [18] to handle loops and noise by considering direct and indirect successors. Huang *et al.* [10] developed an algorithm for capturing a process in execution according to Synchro-Net. Synchro-Net is a synchronization-based model of workflow logic and workflow semantics. Huang's algorithm is able to deal with problems such as hidden tasks and short-loops.

Other related work can be found in Schimm *et al.* [13, 14], Herbst *et al.* [8, 9], Cook *et al.* [4], Van der Aalst *et al.* [12, 19, 20], Wainer *et al.* [23], Van der Aalst [21], and De Medeiros *et al.* [2,15]. Most algorithms can handle the simple relationships of activities like simple sequent, concurrence, alternative, and loop, but none of them can deal with the option properly.

3. PROPOSED METHOD

"Options" are very common in practice—they simultaneously involve two patterns of the relationship of activities: concurrence and alternative. The popular α-algorithm does not work well for options. In this paper, we modify the α-algorithm to improve its ability to handle options. Our proposed process model is represented by Petri nets. Basically, a Petri net consists of a series of transitions, places and arcs. Transitions and places are connected through directed arcs in such a way that (i) places and transitions have at least one incident edge and (ii) in every path, transitions and places alternate (no place is connected to a place and no transition is connected to a transition).

We first define two matrices: an Activity-place matrix is used to describe the route from an activity to a place; while a Place-activity matrix to present a route from a place to an activity. Once these two matrices are constructed, it is rather straightforward to build up our process model, by a proper combination of the basic elements in Petri nets. A basic element in Petri nets includes sequent, AND (concurrence) and OR (alternative), as shown in Figure 2. Specifically, from the original event logs, a From-To table can be constructed. All activities can then be classified into sequent, concurrence, alternative, or loop (see detailed description below). Based on the classification results, we then modify the From-To table to construct Activity-place and Place-activity matrices. From these two matrices, our final model can then be established. Details will be discussed in Section 3.3.

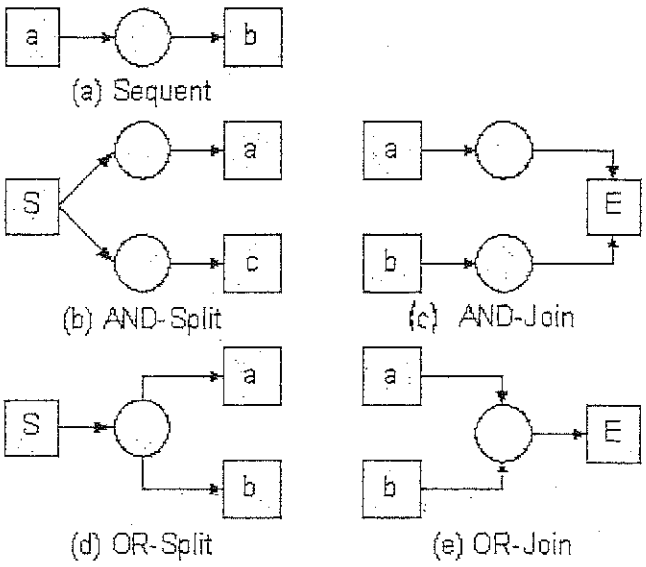

Figure 2. The basic components in the α-algorithm

When building up a process model based on the α-algorithm, two problems need to be resolved： (1) how to distinguish between concurrence and a 2-length loop (a loop involved with only two activities), and (2) how to distinguish between concurrence and alternative. We next discuss these problems in detail.

3.1. Concurrence and 2-length loop

When a process model includes a 2-length loop, we are unable to distinguish between concurrence and the loop using the From-To matrix. For example, if a loop involves activity B and activity C, there are two possibilities as shown in Figure 3(a) and (b). Note that the relationship of activities B and C in Figure 3(c) is a concurrence. Potential traces are {ABCD}, {ABCBCD}, {ABCBCBCD}... {ABCBC...CD} in Figure 3(a); {ABD}, {ABCBD}, {ABCBCBD}... {ABCBC.....BD} in Figure 3(b); and {ABCD} and {ACBD} in Figure 3(c). From the From-To matrices based upon these traces in Figure 3(a), (b) and (c), we are unable to distinguish the exact relationship between B and C directly. However, we can distinguish them by their predecessors and successors. In Figure 3(a), the predecessors of activity B are activities A and C and the only predecessor of activity C is activity B. On the other hand, the only successor of activity B is activity C and the successors of activity C are activities B and D. This implies that there exists a sequential rule between activities B and C starting from activity B and ending at activity C. A similar situation can be found in Figure 3(b). In Figure 3(c), the predecessors of activity B are activities A and C and the predecessors of activity C are activities A and B; on the other hand, the successors of activity B are activities C and D and the successors of activity C are activities B and D; there is no sequential rule between activities B and C. It turns out that we can distinguish concurrence and loop by the predecessors and successors of all activities involved. Namely, if the predecessors and successors are the same, the relationship is a concurrence; otherwise, it is a loop.

3.2. Concurrence and Alternative

We next consider three types of concurrence and alternative, including simple concurrence, simple alternative, and complex concurrence & alternative.

(1) Simple concurrence

When mutually exclusive sets of activities have the same predecessor and all the involved activities are contained in the same traces, then they are a simple concurrence. For example, in Figure 4, the relationship of activity B and activity C only includes concurrence and they always appear together in a trace (see also Figure 2(b)).

(2) Simple alternative

When mutually exclusive sets of activities have the same predecessor and no more than one set of activities appears in the same traces, then they are a simple alternative. In Figure 5, for example, the relationship of activity B and activity C only includes an alternative and they have never occurred simultaneously in any trace (see also Figure 2(c)).

(3) Complex concurrence and alternative

We subdivide the complex concurrence and alternative case into three types:

Type 1: multi-sets of relationships of activities involving simple concurrence;

Type 2: multi-sets of relationships of activities involving simple concurrence or simple alternative; and

Type 3: the relationship of activities including both concurrence and alternative.

The relationship of activities including both concurrence and alternative is the most difficult to model. Detailed illustrations are shown as follows.

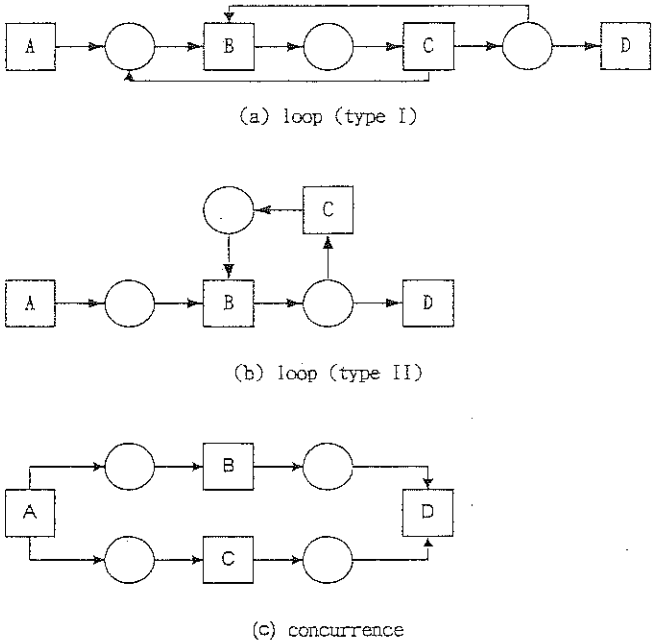

Figure 3. Examples of loop and concurrence

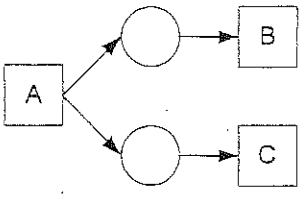

Figure 4. Simple concurrence

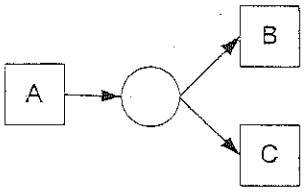

Figure 5. Simple alternative

Type 1: multi-sets of relationships of activities involving simple concurrence

Here, the involving activities include several groups and the relationship of activities in the same group is a simple concurrence, while different groups might have the same activities. For example, for the process model shown as Figure 6, its traces could be {ABCE}, {ACBE}, {ADCE} and {ACDE}. Activities B and C belong to a group involved a concurrence and activities C and D also belong to another group involved a concurrence.

We thus set a place to connect with activity C and another place to connect with activities B and D, since there are only two actions following activity A: one goes to activity C and the other goes to activity B or D. However, if these involved relationships of activity can not be successfully merged without relationship, then these activities should be treated as an alternative by adding suitable fictitious activities.

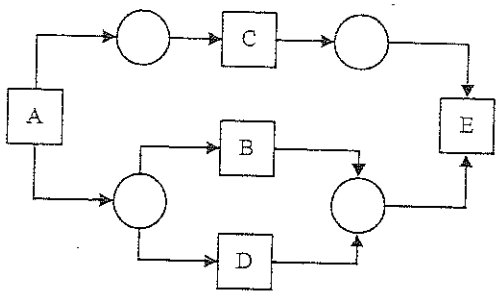

Figure 6. Concurrence and alternative: Type 1

Type 2: multi-sets of relationships of activities involving simple concurrence or alternative

Here, the involving activities can be divided into several groups based upon their traces. For example, from an event log including traces {ABCD}, {ACBD}, {AED} and {AFD}, the involving activities B, C, E, and F can be divided into three groups. The first group contains the activities B and C, the second group contains the activity E and the third group contains activity F. Only the first group involves concurrence. Also, the groups appear alternately in the traces. The number of the actions after activity A depends on the selection of groups. If the first group is involved, there are two actions—activities B and C. For the other two groups, there is only one action—activity E or F. To overcome such a problem, we set a place to connect the above-mentioned three groups. The relationship of activities B and C is concurrence and two places are needed to show the relationship in Petri nets. Furthermore, to match the requirements of Petri nets, we add two fictitious activities to this model to handle this conflict. The process model is shown as Figure 7.

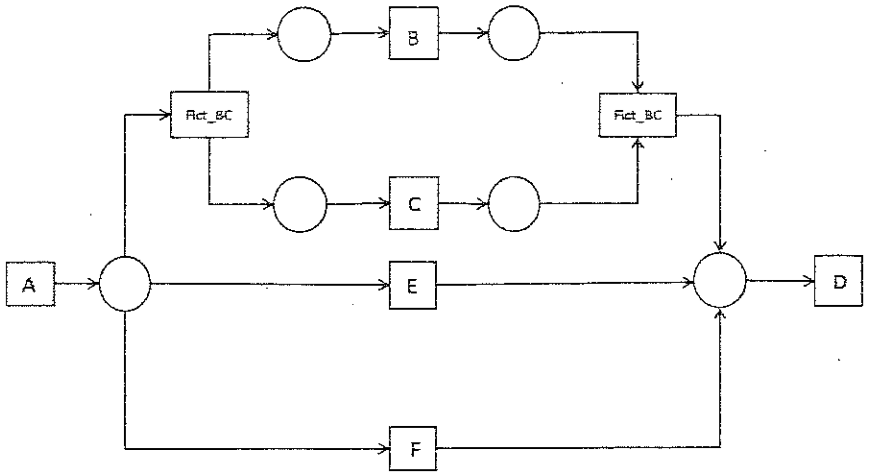

Figure 7. Concurrence and alternative: Type 2

Type 3: the relationship of activities including both concurrence and alternative

Here, the relationships of the involving activities are concurrence for some traces, alternative for other traces, or they do not occur at all. For example, from traces {ABCD}, {ACBD}, {ABD}, {ACD} and {AD}, we find that activities B and C involve concurrence in the first two traces, but they involve alternative in the third and fourth traces, and none in the fifth trace. We thus set places to connect activity A with its successors in all traces. If a successor occurs in some traces, we set a place to connect a fictitious activity. Take traces {ABCD}, {ACBD}, {ABD} and {AD} as an example. In the first two traces, the relationship of activities B and C is concurrence; but in third trace, there is only activity B, and in the fourth trace, both activities B and C are not involved. The successors of activity A should include activities B, C and D; activity D is also the successor of activities B and C, therefore, the fourth trace can be considered as neither activities B nor C, after activity A and before activity D. Figure 8 presents the model for this case. In Figure 9, "No B" and "No C" are the fictitious activities.

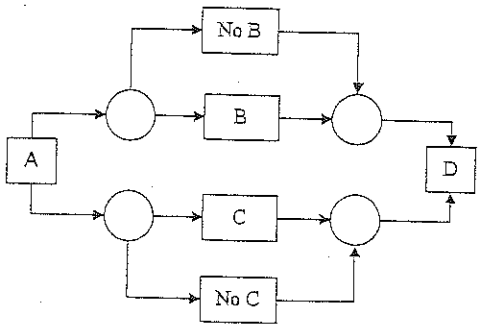

Figure 8. Concurrence and alternative: Type 3

3.3 The proposed algorithm

Step 1 : Build up a $n \times n$ From-To table, based on the event logs. The (i,j)-th element in the From-To table, M_{ij}, is the path from activity a_i to activity a_j which can be defined as

$$M_{ij} = \begin{cases} 1, & \text{if activity } a_j \text{ is directly after activity } a_i; \\ 0, & \text{otherwise .} \end{cases}$$

Step 2 : Build up a modified $n \times n$ From-To table by distinguishing *concurrence* from *loop*.

If $M_{ij} = M_{ji} = 1$, but $M_{cj} \neq M_{ci}$, for $c = 1, 2, \ldots, n$, (and $c \neq i, j$), then the relationship between activities a_i and a_j is a loop. If $M_{ij} = M_{ji} = 1$, and $M_{cj} = M_{ci} = 1$, for $c = 1, 2, \ldots, n$, (and $c \neq i, j$), then we have to further investigate the relationship between the activities a_i and a_j from the event logs. If two activities a_i and a_j have never occurred successively, then the relationship between these activities only involves concurrence; otherwise the relationship is a complex concurrence & loop. If it is a simple concurrence, we modify the From-To table by replacing 0 to the values of M_{ij} and M_{ji}.

Step 3 : Build up a relationship table by distinguishing *concurrence* from *alternative* to describe the relationship of activities, as previously discussed.

Step 4 : Set up an $(n+x)\times S$ Activity-place matrix based on the first two columns in the relationship table: "Predecessors" and "Places". The (i,j)-th element in the Activity-place matrix, A_{ij}, is the path from activity a_i to place S_j, with

$$A_{ij} = \begin{cases} 1, \text{ if activity } a_j \text{ is directly after place } S_j; \\ 0, \text{otherwise}. \end{cases}$$

Step 5 : Set up an $S\times(n+x)$ Place-activity matrix based on the last two columns in the relationship table: "Places" and "Successors". The (i,j)-th element in the Place-activity matrix, P_{ij}, is the path from place S_i to activity a_j, where P_{ij} is defined as:

$$P_{ij} = \begin{cases} 1, \text{ if place } S_i \text{ is directly after activity } a_j; \\ 0, \text{otherwise}. \end{cases}$$

Step 6 : Generate a graphical process model using Petri nets according to the Activity-place and Place-activity matrices obtained in Steps 4 and 5.

3.4 A Detailed Description for the Construction of the Relationship Table

The relationship of activity table consists of three columns, "Predecessors", "Places", and "Successors". A start-activity and an end-activity are also given. A start-activity is one whose column sum is minimal in the From-To table, while an end-activity is one whose row sum is minimal in the From-To table. Suppose activity a_p is a predecessor of activities a_i and a_j. The following expression will be used:

- $a_p \rightarrow (\{a_i\}\{a_j\})$, when the activity between a_i and a_j is a simple concurrence;
- $a_p \rightarrow a_i$, $a_i \rightarrow a_j$, $a_j \rightarrow a_i$, when the activity between a_i and a_j is a simple loop;

- $a_p \rightarrow (\{a_i\}\{a_j\}), a_i \rightarrow a_j, a_j \rightarrow a_i$, when the activity between a_i and a_j is a concurrence & loop.

For each non-zero entry in the modified From-To table in Step 2, we generate one row in the relationship table for both the Predecessor (From) and Successor (To) elements, taking into account the above classification. Once this is completed, we next compress the relationship table by merging those rows with identical Predecessors. The corresponding successors will be integrated accordingly. Special care is needed for concurrence—a fictitious activity (say, no-a_p) will be required when a concurrence only appears in some cases. Finally, we assign the places to the middle column in the relationship table. The number of braces in the set of successors is the number of places we have assigned to each row of the Places column.

4. AN ILLUSTRATION EXAMPLE

The proposed algorithm is illustrated via a simple example as shown in Table 3 below. This example contains 5 cases and 7 different activities.

The procedure of creating a process model is as follows:

Step 1: Convert the event logs data in Table 3 into the From-To table as Table 4 below.

Step 2: Table 4 indicates that there are both B→ C and C→B (as well as E→F and F→E). The relationship of activities B and C only involves concurrence, because there is no sequential order between them; while the relationship of activities E and F is a loop, because they have no common predecessor and successor. We thus modify the From-To table and get a Modified From-To table as in Table 5.

Table 3. Event logs of an example

Step\Case ID	Case 1	Case 2	Case 3	Case 4	Case 5
1	S	S	S	S	S
2	B	C	D	B	B
3	C	B	E	E	E
4	E	E	F	F	F
5	F	F	E	G	E
6	E	G	F		F
7	F		G		G
8	G				

Table 4. From-To matrix

From\To	S	B	C	D	E	F	G
S	0	1	1	1	0	0	0
B	0	0	1	0	1	0	0
C	0	1	0	0	1	0	0
D	0	0	0	0	1	0	0
E	0	0	0	0	0	1	0
F	0	0	0	0	1	0	1
G	0	0	0	0	0	0	0

Table 5. The modified From-To matrix After reconsidering {B,C} and {E,F}

From\To	S	B	C	D	E	F	G
S	0	1	1	1	0	0	0
B	0	0	0	0	1	0	0
C	0	0	0	0	1	0	0
D	0	0	0	0	1	0	0
E	0	0	0	0	0	1	0
F	0	0	0	0	1	0	1
G	0	0	0	0	0	0	0

Step 3: From Table 5, activities S and G are chosen to be the start-activity and the end-activity, respectively. Table 6(a) is constructed, based upon all non-zero entries in Table 5. Table 6(b) is the result of compressing all identical Predecessors in Table 6(a). The concurrence ({B}{C}) needs further investigation. A fictitious activity No-C (NC) is created because C did not occur in some cases when B occurs, and thus ({B}{C}) is

modified to be ({B}{C, NC}). Likewise, since the successors of S and the predecessors of E also involve concurrence, we create fictitious activities X_{SBC} and X_{BCE} respectively for the connection of S to ({B}{C, NC}) and ({B}{C, NC}) to E, as shown in Table 6(c). After adding the proper number of places for each predecessor to be connected to its successors Table 6(d) displays the final relationship table.

Table 6(a). The table for the original relationship of activities

Predecessors	Places (name)	Successors
S		{B}{C}
S		D
{B}{C}		E
D		E
E		F
F		E
F		G
G		

Step 4 : A 10×10 Activity-place matrix is built to record the route from each activity to a place. If the route exists, then a "1" is marked in the corresponding location; otherwise a "0" is marked, as displayed in Table 7.

Table 6(b). The table for the first step to establish relationship of activities

Predecessors	Places (name)	Successors
S		{({B}{C}),
{({B}{C}), D }		E
E		F
F		{E, G}
G		

Table 6(c). The table for the second step to establish relationship of activities

Predecessors	Places (name)	Successors
S		$\{X_{SBC}, D\}$
X_{SBC}		({B}{C,NC})
$\{(X_{BCE}, D\}$		E
({B}{C,NC})		X_{BCE}
E		F
F		{E, G}
G		

Table 6(d) The table for the final step to establish relationship of activities

Predecessors	Places (name)	Successors
	S0	S
S	S1	$\{X_{SBC}, D\}$
X_{SBC}	S2, S3	({B}{C,NC})
$\{(X_{BCE}, D\}$	S4	E
({B}{C,NC})	S5, S6	X_{BCE}
E	S7	F
F	S8	{E, G}
F	S4	E
G	End	

Step 5: Likewise, a 10×10 Place-activity matrix is constructed and displayed in Table 8.

Table 7. Modified Activity-place matrix

From\to	S0	S1	S2	S3	S4	S5	S6	S7	S8	End
S	0	1	0	0	0	0	0	0	0	0
X_{SBC}	0	0	1	1	0	0	0	0	0	0
B	0	0	0	0	0	1	0	0	0	0
C	0	0	0	0	0	0	1	0	0	0
NC	0	0	0	0	0	0	1	0	0	0
D	0	0	0	0	1	0	0	0	0	0
E	0	0	0	0	0	0	0	1	0	0
F	0	0	0	0	1	0	0	0	1	0
X_{BCE}	0	0	0	0	1	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	1

Table 8. Modified Place-activity matrix

From\to	S	X_{SBC}	B	C	NC	D	E	F	X_{BCE}	G
S0	1	0	0	0	0	0	0	0	0	0
S1	0	1	0	0	0	1	0	0	0	0
S2	0	0	1	0	0	0	0	0	0	0
S3	0	0	0	1	1	0	0	0	0	0
S4	0	0	0	0	0	0	1	0	0	0
S5	0	0	0	0	0	0	0	0	1	0
S6	0	0	0	0	0	0	0	0	1	0
S7	0	0	0	0	0	0	0	1	0	0
S8	0	0	0	0	0	0	1	0	0	1
End	0	0	0	0	0	0	0	0	0	0

Step 6: Tables 7 and 8 above allow us to develop a process model using Petri nets followed by the Activity-Place and the Place-Activity matrices obtained. Figure 9 depicts the resulting process model.

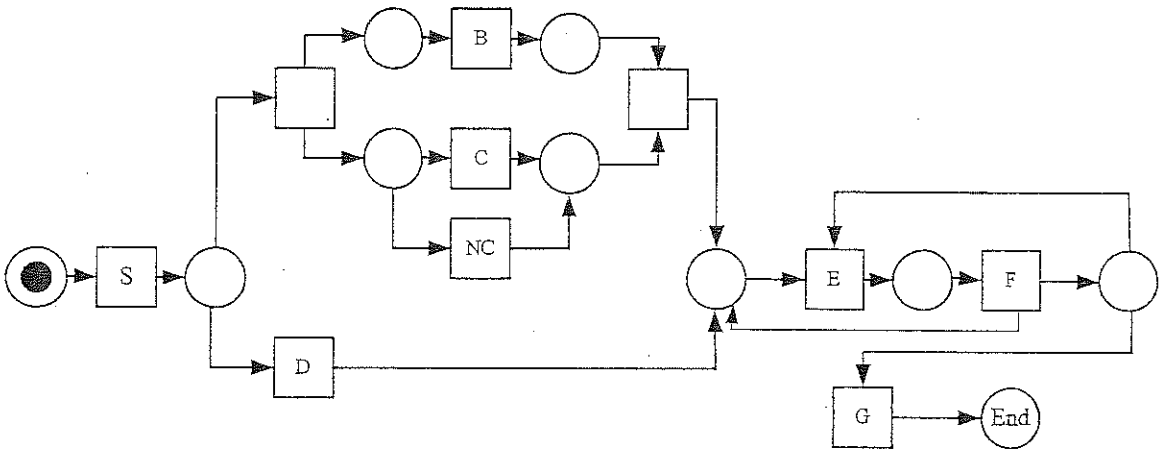

Figure 9. The resulting process model

5. CASE STUDY

The proposed algorithm has successfully implemented in the Pediatrics Department at a private Taiwan hospital. Due to the shortage of staff nurses, the superintendent hopes to investigate the workflow of the staff nurses in details. The ultimate goal here is to lighten

the working loads without causing any medical negligence. This is now becoming possible, due to the support of modern information system.

The database consist daily activities of 85 nurses (Cases). The first five cases are given in Appendix, as examples. There are a total of 33 different activities in a daily nurse work. Each nurse will involve most of them (but not all)—the actual activities involved depends on the number of patients in the specific day (the average number of activities involved is around 30, among all 85 cases). The detailed description on each activity ("sign_in", "exchange_report", etc.) is not listed here, but is available upon request. Both the proposed algorithm and the α/α++-algorithm are applied for the comparison purpose. The resulting process models are displayed respectively in Figure 10(a) and Figure 10(b), for the α++-algorithm and the proposed algorithm (note that the α++-algorithm used here is downloaded at the website <http://promtools.org/prom5/>).

The current workflow process is mainly based upon experience. Our workflow process model provides a clear presentation and understanding for the entire procedure. As shown in Figure 11(b), a "typical" workflow is clearly illustrated. Specifically, a staff nurse first signs in ("sign_in") at the hospital and participates a meeting hosted by the night-shift leader of reporting the patients' current situations ("exchange_report"). Next, the staff nurses will do one of the followings: (a) reviewing the techniques of caring of the sickness ("exchange_TR")—typically held on Monday, Wednesday and Friday, or (b) reporting some special cases ("exchange_CR")—typically held on Tuesday or (c) do nothing on special date ("fict_1"), and so on.

The resulting workflow model given in Figure 10(b) is proved to be useful to the superintendent—not only s/he has a much better understanding the entire process for future improvements, but also s/he is able to identify some important details being ignored. This will be reported elsewhere. Here we will focus on the comparisons of the models given in Figures 11(a) and 10(b). It is shown that Figure 10(b) obtained by the proposed algorithm is much closer to the reality than Figure 11(a). For example, the nurse needs to

help patients putting updated medical information in order ("MC_A"); checking the intravenous drip ("MC_B"); checking the oxygen mask ("MC_C"); and checking the patient with unusual BT ("MC_D"). These four activities need to perform concurrently. However, in reality, these four activities could be accomplished by team, and not necessarily by individual (meaning, they may be done by different nurses—not every single nurse needs to do them all). Case 5 in the Appendix B, for example, indicates that this nurse completed "MC_A", "MC_B" and "MC_C", but not "MC_D". However, Figure 10(a)—the process model constructed by theα/α++ -algorithm, is not capable to deal with such an option. It indicates that all these four activities need to be done concurrently; while Figure 10(b)—the process model constructed by the proposed algorithm, is more realistic to reflect the facts.

The proposed algorithm is a modification of the $^{\alpha}$-algorithm by including more complicate processes—the proposed algorithm has an additional task to confirm the inferred relationship of activities which involving concurrence and alternative. It will take more times to perform. In this case study (85 cases and 33 potential activities), the computing times of the proposed algorithm is longer than that of $\alpha/\alpha++$-algorithm, but the difference is marginal (10 minutes versus 2 minutes). However, when the data is huge, the computing time for the proposed algorithm could be far more than that for $\alpha/\alpha++$-algorithm. The processing time is indeed an issue that to overcome.

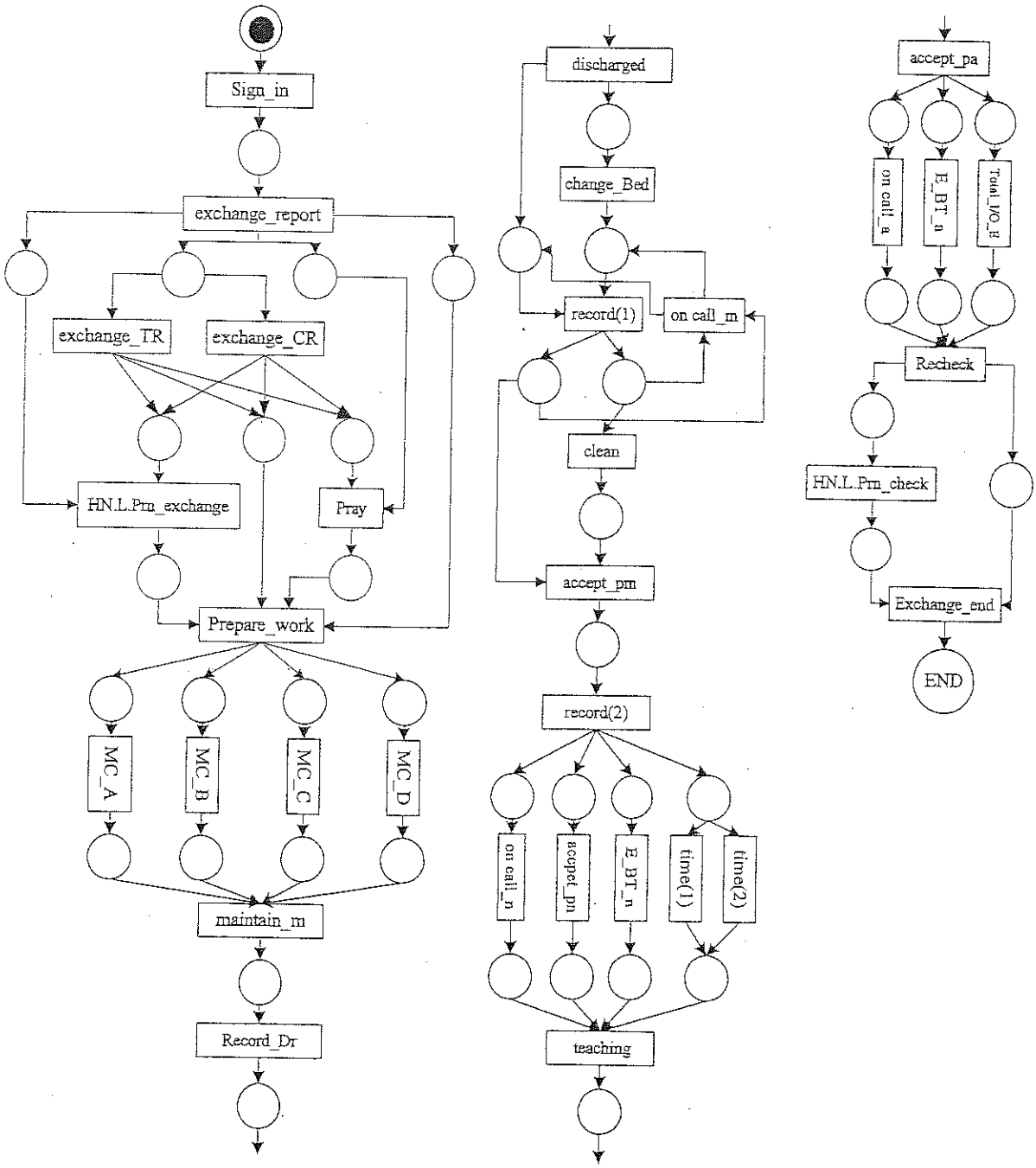

Figure 10 (a). The process model by α/α++-algorithm

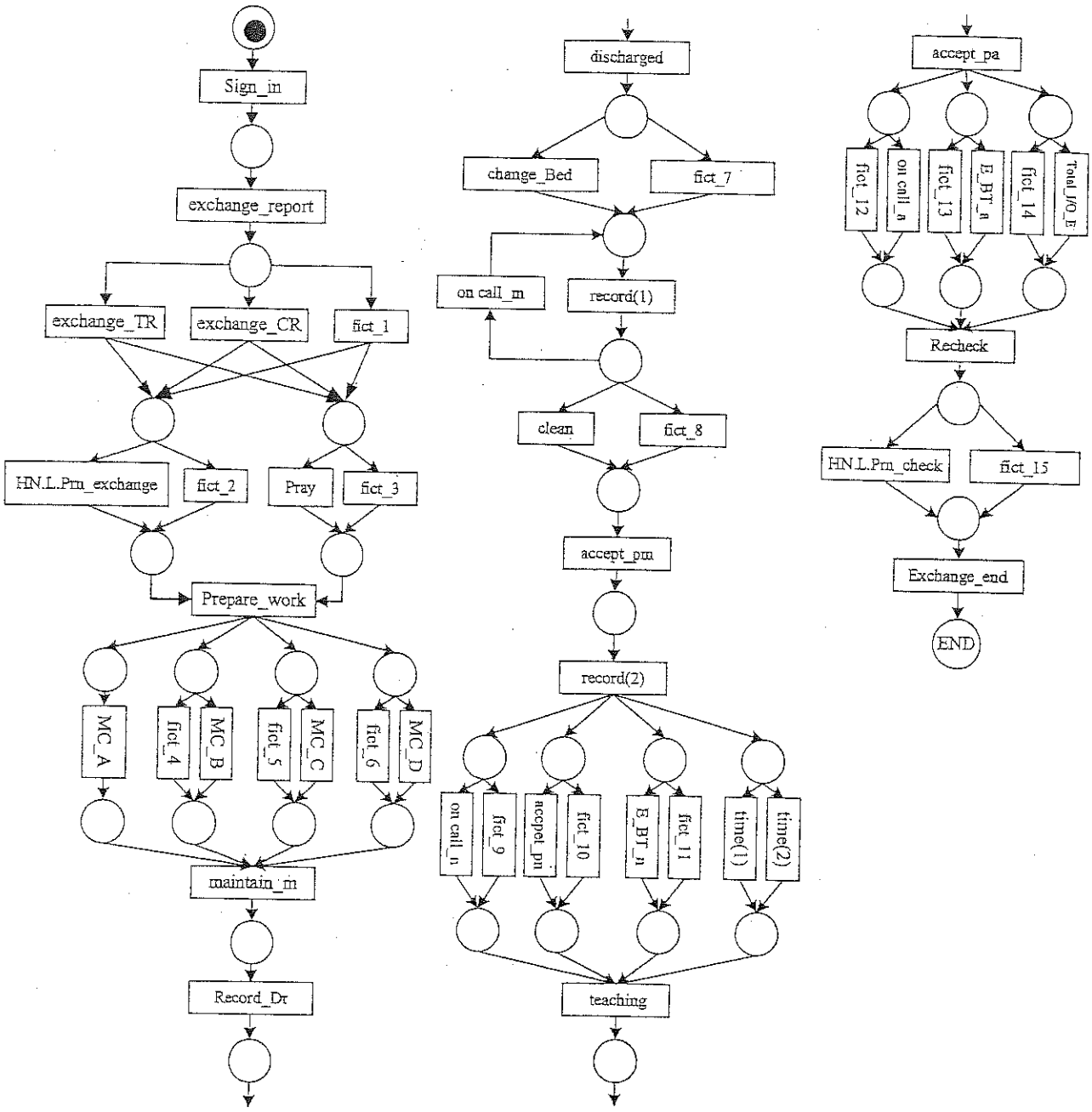

Figure 10 (b). The process model by the proposed method

6. DISCUSSION AND COMPARISON

The proposed algorithm, basically built upon the core idea of the α-algorithm, is to resolve the important situation of options. The comparison among our method, the α-algorithm and other related algorithms (including, the . -algorithm, and the . -algorithm) is shown in Table 9. The items/criteria to be compared include basic concurrence, alternative and concurrence, basic loops, arbitrary loops, and hidden tasks. Specifically, "Basic concurrence" means the relationship of activities including simply concurrence, while "Alternative and concurrence" indicates the relationship of activities involving alternative and concurrence simultaneously. A "Basic loop" represents a loop concerning only one or two activities, also known as a "short-loop." A "Hidden task" expresses that some activities should have been recorded, but these activities are not scheduled in the process model. If an algorithm has the ability to resolve the issue, we indicate this by a "Yes"; otherwise by a "No".

Table 9. Comparison among proposed method and a- related algorithms

	The	α-algorithm	α+-algorithm	α++-algorithm
Basic	Yes	Yes	Yes	Yes
Alternative	Yes	No	No	No
Basic loop	Yes	Yes	Yes	Yes
Hidden task	Yes	No	No	No

As indicated in Table 9, all these methods are able to handle the simple situations, such as basic concurrence and basic loops. The proposed algorithm is capable of solving for the more complicated situations of options, i.e., "Alternative and Concurrence". Note that we only modify the -algorithm to resolve the option problem. Thus, the comparison here is based on how many relationships can be resolved, not the efficiencies among the algorithms.

7. CONCLUSION

Our study here is to develop an algorithm for determining a process model based on the most confined data. Specifically, we partition the relationships of activities into three different kinds, simple concurrence, simple alternative, and complex concurrence & alternative. Compared with other existing algorithms, the α-algorithm the α+-algorithm and the α++-algorithm, only our proposed algorithm is capable to handle problems involving options (for which complex concurrence and alternative occur simultaneously). The proposed algorithm in this paper is able to catch all the traces in the event logs with more complex relationships of activities involving options. The proposed algorithm basically is a creative modification over -algorithm, by adding a key step on checking the inferred relationships of activities from the traces. Thus it takes longer processing time than -algorithm. Indeed, this could be a problem for large dataset (an important problem to overcome). However, for any modest dataset, the difference on computing times is rather marginal (as shown in our case study).

The search for an "optimal" process model is rather primitive in the literature. Besides computational complexity, one optimality criterion currently under consideration is the "minimal process model" for which the resulting process model can not only capture all events in event logs, but also generate least amount of additional flow. This is currently under study. Another interesting problem is how to develop the process mining for event logs with noises. Cook et al. [4] and Van der Aalst et al. [12, 19, 20] presented a heuristic approach to construct workflow nets based on counting frequencies of dependencies between activities. We anticipate more decent research in this area can be accomplished in the near future.

REFERENCE

1. Agrawal R, Gunopulos D, Leymann F. Mining process models from workflow logs. *Lecture Notes in Computer Science* 1998; 1337:469.DOI:10.1007/BFb0100972
2. Alves de Medeiros AK, Weijters AJMM, Van der Aalst WMP. Genetic process mining: a basic approach and its challenges. In *BPM 2005 Workshops. LNCS 3812*, Bussler C et. al. (eds.). Springer-Verlag Berlin Heidelberg, 2006; 203-215.
3. Cook JE, Wolf AL. Automating process discovery through event-data analysis. *ACM ICSE '95* 1995; 73-82.
4. Cook JE, Wolf AL. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology* 1998; 7: 215-249.
5. Cook JE, Wolf AL. Event-based detection of concurrency. *ACM SIGSOFT Software Engineering Notes , Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering SIGSOFT '98/FSE-6* 1998; 35 – 45.
6. De Medeiros AKA, Van der Aalst WMP, Weijters AJMM. Workflow mining: Current status and future directions. In *CoopIS/DOA/ODBASE 2003. LNCS* 2888, Meersman R et. al. (eds.). Springer-Verlag Berlin Heidelberg, 2003; 389-406.
7. De Medeiros AKA, Van Dongen BF, Van der Aalst WMP, Weijters AJMM. Process mining: Extending the α-algorithm to mine short loops. http://fp.tm.tue.nl/beta/publications/working%20papers/Beta_WP113.pdf [1 August 2007]
8. Herbst J, Karagiannis D. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. http://citeseer.ist.psu.edu/cache/papers/cs/25888/http:zSzzSzhome.t-online.dezSzhomezSzjoachim.herbstzSzijisafm00.pdf/herbst98integrating.pdf. [1 August 2007]
9. Herbst J, Karagiannis D. Workflow mining with InWoLvW. *Computers in Industry* 2004; 53: 245-264. DOI: 10.1016/j.compind.2003.10.002.
10. Huang XQ, Wang LF, Zhao W, Zhang SK, Yuan CY. A workflow process mining algorithm based on synchro-net. *J. Comput. Sci. & Technol* 2006; 21: 66-71.

11. Humphrey WS, Kellner MI. Software process modeling: principles of entity process models. *ACM* 1989; 331-342.
12. Maruster Laura, Weijters AJMM, Van der Aalst WMP, Van den Bosch A. Process mining: Discovering direct successors in process logs. In *DS 2003. LNCS* 2534, Lange S, Satoh K, Smith CH (eds.). Springer-Verlag Berlin Heidelberg 2002; 364-373.
13. Schimm G. Mining exact models of concurrent workflows. *Computers in Industry* 2004; 53: 265-281. DOI:10.1016/j.compind.2003.10.003
14. Schimm G. Mining Most Specific Workflow Models from Event-Based Data. In *BPM 2003. LNCS* 2678, Van der Aalst WMP et al. (eds.). Springer-Verlag Berlin Heidelberg, 2003; 25–40.
15. Van der Aalst WMP, Alves de Medeiros AK, Weijters AJMM. Genetic process mining. In *Applications and Theory of Petri Nets 2005. LNCS* 3536, Cortadella J, Reisig W(eds.). Springer-Verlag Berlin Heidelberg, 2005; 48-69.
16. Van der Aalst WMP, Alves de Medeiros AK. Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science* 2005; 121:3-21. DOI: 10.1016/j.entcs.2004.10.0.13.
17. Van der Aalst WMP, Reijers HA, Weijters AJMM, Van Dongen BF, Alves de Medeiros AK, Song M, Verbeek HMW. Business process mining: An industrial application. *Information Systems* 2007; 32:714-732. DOI:10.1016/j.is.2006.05.003.
18. Van der Aalst WMP, Van Dongen BF, Herbst J, Maruster L, Schimm G, Weijters AJMM. Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering* 2003; 47: 237-267.
19. Van der Aalst WMP, Weijters A JMM. Process mining: a research agenda. *Computers in Industry* 2004; 53: 231-244. DOI:10.1016/j.compind.2003.10.001
20. Van der Aalst WMP, Weijters AJMM, Maruster L. Workflow Mining: Discovering Process Models from Event Logs.
http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2004/BPM-04-06.pdf [1 August 2007]

21. Van der Aalst WMP. Matching observed behavior and modeled behavior: An approach based on Petri nets and integer programming. *Decision Support System* 2006; 42:1843-1859. DOI:10.1016/j.dss.2006.03.013.

22. Van Dongen BF, Mendling J, Van der Aalst WMP. Structural patterns for soundness of business process models. Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06) 2006; 00:116 – 128. DOI: 10.1109/EDOC.2006.56.

23. Wainer J, Kim K, Ellis CA. A workflow mining method through model rewriting. In *CRIWG 2005. LNCS* 3706, Fuks H, Lukosch S, Salgado AC (eds.). Springer-Verlag Berlin Heidelberg, 2005; 184-191.

24. Wen L, Van der Aalst WMP, Wang J, Sum J. Mining process models with Non-free-choice constructs. http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2006/BPM-06-23.pdf [1 August 2007].

Appendix. *Partial dataset of the Case Study*

(the complete data is available upon request)

Step	Case 1	Case 2	Case 3	Case 4	Case 5
1	sign_in	sign_in	sign_in	sign_in	sign_in
2	exchange_report	exchange_report	exchange_report	exchange_report	exchange_report
3	Prepare_work	exchange_TR	exchange_CR	HN.L.Prn_exchange	exchange_CR
4	MC_A	HN.L.Prn_exchange	HN.L.Prn_exchange	Prepare_work	HN.L.Prn_exchange
5	MC_B	Prepare_work	Pray	MC_A	Prepare_work
6	MC_C	MC_D	Prepare_work	maintain_m	MC_A
7	maintain_m	MC_B	MC_C	Record_Dr	MC_B
8	Record_Dr	MC_A	MC_B	discharged	MC_C
9	discharged	MC_C	MC_D	record(1)	maintain_m
10	record(1)	maintain_m	MC_A	clean	Record_Dr
11	on call_m	Record_Dr	maintain_m	accept_pm	discharged
12	record(1)	discharged	Record_Dr	record(2)	change_Bed
13	clean	change_Bed	discharged	E_BT_n	record(1)
14	accept_pm	record(1)	change_Bed	accpet_pn	on call_m
15	record(2)	clean	record(1)	on call_n	record(1)
16	accpet_pn	accept_pm	clean	time(2)	clean
17	on call_n	record(2)	accept_pm	teaching	accept_pm
18	time(2)	time(1)	record(2)	accept_pa	record(2)
19	E_BT_n	E_BT_n	time(2)	on call_a	E_BT_n
20	teaching	on call_n	E_BT_n	E_BT_a	accpet_pn
21	accept_pa	teaching	on call_n	Recheck	time(2)
22	E_BT_a	accept_pa	accpet_pn	Exchange_end	teaching
23	on call_a	on call_a	teaching		accept_pa
24	Total_I/O_E	E_BT_a	accept_pa		Total_I/O_E
25	Recheck	Recheck	on call_a		on call_a
26	Exchange_end	HN.L.Prn_check	E_BT_a		E_BT_a
27		Exchange_end	Recheck		Recheck
28			HN.L.Prn_check		Exchange_end
29			Exchange_end		

「數據分析」
2013年4月，8卷2期，頁1-30

流程探勘：建構流程模型新方法

林共進[1]*　陳雲岫[2]　郭珉旬[3]

摘　要

在現今充滿競爭的產業結構下，建構一工作流程是相當複雜且費時的工作，卻也是業界所必須面對的問題，尤其是涉及軟體工程與流程管理領域。而流程探勘的技術也在這環境中日益受重視。該技術主要是從工作流程所產生之記錄資料中分析、建構流程模型，以此提供使用者了解真正運作中的流程狀態。在這研究中，我們提出一個新的建構法，藉由改善 α-algorithm 使其能處理較爲複雜但卻是生活中常見的流程模型，如活動之間的關係涉及多擇多的狀況，並以簡單例子說明演算法之步驟，同時也將該演算法運用於依實際案例中，結果也證明我們所提出之演算法能較 α-algorithm 或其相關演算法萃取出更貼近事實的流程。

關鍵字: α-algorithm、派翠網路、流程模型、流程資料

---

[1] 美國賓州州立大學統計學系，教授
[2] 元智大學工業工程與管理學系，教授
[3] 元智大學工業工程與管理學系，博士
*Correspondence to : Dennis K. J. Lin, *Department of Statistics,317 Thomas Building, Pennsylvania State University, University Park, PA 16802-2111, USA.*　E-mail: DKL5@psu.edu
收件日期：2012.9.17　；修改日期：2012.11.26　接受日期：2013.3.1