Part IV

Techniques for Process Design

12 Process Mining

W.M.P. van der Aalst and A.J.M.M. Weijters
Department of Technology Management, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.

12.1 INTRODUCTION

ToDo: Check changes and do a final pass through the paper for consistency and typos.

The basic idea of process mining is to extract knowledge from event logs recorded by an information system. Until recently, the information in these event logs was rarely used to analyze the underlying processes. Process mining aims at improving this by providing techniques and tools for discovering process, control, data, organizational, and social structures from event logs. Fuelled by the omnipresence of event logs in transactional information systems (cf. WFM, ERP, CRM, SCM, and B2B systems), process mining has become a vivid research area [4, 5]. In this chapter we provide an overview of process mining techniques and tools and discuss one algorithm (the α algorithm) in detail.

As explained in Chapter 1 many information systems have become process-aware. This awareness can be used in various ways, e.g., the process-aware information system may *enforce* a specific way of working but may also just *monitor* the process and suggest alternative ways of working. Workflow Management (WFM) systems such as Staffware, IBM MQSeries, COSA, etc. (cf. Chapter 3) can be used to enforce a specific way of working but may also allow for predefined choices based on human judgment, properties of the case being handled, or a changing context. Case Handling (CH) systems such as FLOWer (cf. Chapter 19) allow for more flexibility by enabling alternative paths that are implicitly defined (e.g. the ability to skip, rollback, or change the order of activities). Both for WFM and CH systems there is an explicit process model that is *actively* used to support the process. In many other process-aware systems the process model plays a less explicit role. For example, although ERP (Enterprise Resource Planning) systems such as SAP, PeopleSoft, Baan,

and Oracle offer a workflow component, process models are often hard-coded or used in a passive way. SAP supports a wide variety of processes. Parts of these processes are hard-coded in the software while other parts of the process are only described in so-called *reference models*. These reference models describe how people *should* use the system. While process models in a WFM are used *actively*, these reference models are only used *passively*. Another example is a Hospital Information System (HIS) supporting clinical guidelines. These guidelines describe the treatment of a patient having a specific health problem and can be used in an active way (e.g., automatically suggest actions to the medical staff) or a passive way (e.g., the medical staff can consult the clinical guideline when needed). Other process-aware information systems such as CRM (Customer Relationship Management) software, SCM (Supply Chain Management) systems, B2B (Business to Business) applications, etc. may use process models actively or passively and these models may be hard-coded in the software, implicit (as in a CH system), or explicit (as in a WFM system). Despite the different ways in which models are used, most of these systems log events in some way. In this chapter, we do not focus on the design of these models but instead we focus on techniques for *monitoring* enterprise information systems (i.e., WFM, ERP, CRM, SCM-like systems).

As mentioned, many of today's enterprise information systems store relevant events in some structured form. For example, workflow management systems typically register the start and completion of activities [2]. ERP systems like SAP log all transactions, e.g., users filling out forms, changing documents, etc. Business-to-business (B2B) systems log the exchange of messages with other parties. Call center packages but also general-purpose CRM systems log interactions with customers. These examples show that many systems have some kind of *event log* often referred to as "history", "audit trail", "transaction log", etc. [4, 7, 10, 16]. The event log typically contains information about events referring to an *activity* and a *case*. The case (also named process instance) is the "thing" which is being handled, e.g., a customer order, a patient in a hospital, a job application, an insurance claim, a building permit, etc. The activity (also named task, operation, action, or work-item) is some operation on the case. Typically, events have a *time stamp* indicating the time of occurrence. Moreover, when people are involved, event logs will typically contain information on the person executing or initiating the event, i.e., the *originator*. Based on this information several tools and techniques for process mining have been developed [1, 3, 6, 7, 8, 11, 12, 14, 16, 19].

It is important to note that all enterprise information systems allow for some form of freedom and that the system is not able to control the entire process. Even in a WFM system there is some degree of freedom, e.g., work items are not allocated to a single user but to a group of users, the routing may be determined by the user or by the arrival of external triggers (e.g., a cancellation by the customer), etc. Note that a WFM cannot control its environment, e.g., if work is offered to a user, then the user will determine when and how to perform it. Other systems typically offer even more freedom.

In many systems the user can deviate from the predefined process model, e.g., in an ERP system the user does not need to follow the reference model completely (it is just a guideline). The fact that all systems allow for some form of freedom makes it interesting to see how people actually work. This motivates the use of process mining techniques as discussed in this chapter.

Process mining is useful for at least two reasons. First of all, it could be used as a tool to find out how people and/or procedures really work, i.e., *process discovery*. Consider for example processes supported by an ERP system like SAP (e.g., a procurement process). Such a system logs all transactions but does not (completely) enforce a specific way of working. In such an environment, process mining could be used to gain insight in the actual process. Another example would be the flow of patients in a hospital. Note that in such an environment all activities are logged but information about the underlying process is typically missing. In this context it is important to stress that management information systems typically provide information about key performance indicators like resource utilization, flow times, and service levels but *not* about the underlying business processes (e.g., causal relations, ordering of activities, etc.). Second, process mining could be used for *Delta analysis*, i.e., comparing the actual process with some predefined process. Note that in many situations there is a descriptive or prescriptive process model. Such a model specifies how people and organizations are assumed/expected to work. By comparing the descriptive or prescriptive process model with the discovered model, discrepancies between both can be detected and used to improve the process. Consider for example the so-called reference models in the context of SAP. These models describe how the system should be used. Using process mining it is possible to verify whether this is the case. In fact, process mining could also be used to compare different departments/organizations using the same ERP system.

Process mining can be used to monitor coordination in enterprise information systems. Some of the coordination is done by humans while other coordination tasks are done by software. As indicated, similar interaction patterns occur at the level of software components, business processes, and organizations. Therefore, process mining can be done at many levels.

The topic of process mining is related to management trends such as Business Process Reengineering (BPR, see also Chapter 10), Business Intelligence (BI), Business Process Analysis (BPA), Continuous Process Improvement (CPI), and Knowledge Management (KM). Process mining can be seen as part of the BI, BPA, and KM trends. Moreover, process mining can be used as input for BPR and CPI activities. Note that process mining is not a tool to (re)design processes. The goal is to understand what is really going on. Despite the fact that process mining is not a tool for designing processes, it is evident that a good understanding of the existing processes is vital for any redesign effort.

The remainder of this chapter is organized as follows. In Section 12.2 we introduce process mining. Using an example, we illustrate the concept

of process mining, discuss the information required to do process mining, and show the various perspectives that can be mined (process perspective, organizational perspective, and case perspective). Section 12.3 focuses on the process perspective and provides a concrete algorithm: The α-algorithm. In Section 12.4 we discuss some limitations of the α-algorithm and possible solutions. To conclude, we provide exercises in Section 12.6.

12.2 PROCESS MINING: AN OVERVIEW

The goal of process mining is to extract information about processes from transaction logs [4]. We assume that it is possible to record events such that (i) each event refers to an *activity* (i.e., a well-defined step in the process), (ii) each event refers to a *case* (i.e., a process instance), (iii) each event can have a *performer* also referred to as *originator* (the person executing or initiating the activity), and (iv) events have a *time stamp* and are totally ordered. Table 12.1 shows an example of a log involving 19 events, 5 activities, and 6 originators. In addition to the information shown in this table, some event logs contain more information on the case itself, i.e., data elements referring to properties of the case. For example, the case handling system FLOWer logs every modification of some data element.

Event logs such as the one shown in Table 12.1 are used as the starting point for mining. We distinguish three different perspectives: (1) the process perspective, (2) the organizational perspective and (3) the case perspective. The *process perspective* focuses on the control-flow, i.e., the ordering of activities. The goal of mining this perspective is to find a good characterization of all possible paths, e.g., expressed in terms of a Petri net [15] (cf. Chapter 7) or Event-driven Process Chain (EPC) [13, 12] (cf. Chapter 6). The *organizational perspective* focuses on the originator field, i.e., which performers are involved and how they are related. The goal is to either structure the organization by classifying people in terms of roles and organizational units or to show relations between individual performers (i.e., build a social network [17]). The *case perspective* focuses on properties of cases. Cases can be characterized by their path in the process or by the originators working on a case. However, cases can also be characterized by the values of the corresponding data elements. For example, if a case represent a replenishment order it is interesting to know the supplier or the number of products ordered.

The process perspective is concerned with the "How?" question, the organizational perspective is concerned with the "Who?" question, and the case perspective is concerned with the "What?" question. To illustrate the first two consider Figure 12.1. The log shown in Table 12.1 contains information about five cases (i.e., process instances). The log shows that for four cases (1, 2, 3, and 4) the activities A, B, C, and D have been executed. For the fifth case only three activities have been executed: activities A, E, and D. Each case starts with the execution of A and ends with the execution of D.

case id	activity id	originator	time stamp
case 1	activity A	John	9-3-2004:15.01
case 2	activity A	John	9-3-2004:15.12
case 3	activity A	Sue	9-3-2004:16.03
case 3	activity B	Carol	9-3-2004:16.07
case 1	activity B	Mike	9-3-2004:18.25
case 1	activity C	John	10-3-2004:9.23
case 2	activity C	Mike	10-3-2004:10.34
case 4	activity A	Sue	10-3-2004:10.35
case 2	activity B	John	10-3-2004:12.34
case 2	activity D	Pete	10-3-2004:12.50
case 5	activity A	Sue	10-3-2004:13.05
case 4	activity C	Carol	11-3-2004:10.12
case 1	activity D	Pete	11-3-2004:10.14
case 3	activity C	Sue	11-3-2004:10.44
case 3	activity D	Pete	11-3-2004:11.03
case 4	activity B	Sue	11-3-2004:11.18
case 5	activity E	Clare	11-3-2004:12.22
case 5	activity D	Clare	11-3-2004:14.34
case 4	activity D	Pete	11-3-2004:15.56

Table 12.1 An event log.

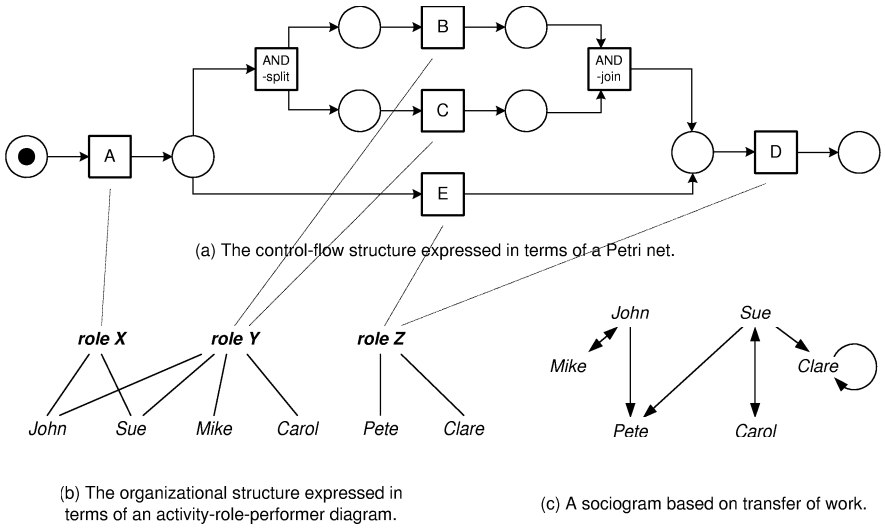

Fig. 12.1 Some mining results for the process perspective (a) and organizational (b and c) perspective based on the event log shown in Table 12.1.

If activity B is executed, then also activity C is executed. However, for some cases activity C is executed before activity B. Based on the information shown in Table 12.1 and by making some assumptions about the completeness of the log (i.e., assuming that the cases are representative and a sufficient large subset of possible behaviors has been observed), we can deduce the process model shown in Figure 12.1(a). The model is represented in terms of a Petri net. The Petri net starts with activity A and finishes with activity D. These activities are represented by transitions. After executing A there is a choice between either executing B and C concurrently (i.e., in parallel or in any order) or just executing activity E. To execute B and C in parallel two non-observable activities (AND-split and AND-join) have been added. These activities have been added for routing purposes only and are not present in the event log. Note that for this example we assume that two activities are concurrent if they appear in any order. By distinguishing between start events and completion events for activities it is possible to explicitly detect parallelism.

Figure 12.1(a) does not show any information about the organization, i.e., it does not use any information concerning the people executing activities. Information about performers of activities however, is included in Table 12.1. For example, we can deduce that activity A is executed by either John or Sue, activity B is executed by John, Sue, Mike or Carol, C is executed by John, Sue, Mike or Carol, D is executed by Pete or Clare, and E is executed by Clare. We could indicate this information in Figure 12.1(a). The information could also be used to "guess" or "discover" organizational structures. For example, a guess could be that there are three roles: X, Y, and Z. For the execution of A role X is required and John and Sue have this role. For the execution of B and C role Y is required and John, Sue, Mike and Carol have this role. For the execution of D and E role Z is required and Pete and Clare have this role. For five cases these choices may seem arbitrary but for larger data sets such inferences capture the dominant roles in an organization. The resulting "activity-role-performer diagram" is shown in Figure 12.1(b). The three "discovered" roles link activities to performers. Figure 12.1(c) shows another view on the organization based on the transfer of work from one individual to another, i.e., not focus on the relation between the process and individuals but on relations among individuals (or groups of individuals). Consider for example Table 12.1. Although Carol and Mike can execute the same activities (B and C), Mike is always working with John (cases 1 and 2) and Carol is always working with Sue (cases 3 and 4). Probably Carol and Mike have the same role but based on the small sample shown in Table 12.1 it seems that John is not working with Carol and Sue is not working with Carol.[1] These examples show that the event log can be used to derive relations between performers of activities, thus resulting in a sociogram. For example,

[1]Clearly the number of events in Table 12.1 is too small to establish these assumptions accurately. However, real event logs will contain thousands or more events.

it is possible to generate a sociogram based on the transfers of work from one individual to another as is shown in Figure 12.1(c). Each node represents one of the six performers and each arc represents that there has been a transfer of work from one individual to another. The definition of "transfer of work from A to B" is based on whether, in the same case, an activity executed by A is directly followed by an activity executed by B. For example, both in case 1 and 2 there is a transfer from John to Mike. Figure 12.1(c) does not show frequencies. However, for analysis purposes these frequencies can added. The arc from John to Mike would then have weight 2. Typically, we do not use absolute frequencies but weighted frequencies to get relative values between 0 and 1. Figure 12.1(c) shows that work is transferred to Pete but not vice versa. Mike only interacts with John and Carol only interacts with Sue. Clare is the only person transferring work to herself.

Besides the "How?" and "Who?" question (i.e., the process and organization perspectives), there is the case perspective that is concerned with the "What?" question. Figure 12.1 does not address this. In fact, focusing on the case perspective is most interesting when also data elements are logged but these are not listed in Table 12.1. The case perspective looks at the case as a whole and tries to establish relations between the various properties of a case. Note that some of the properties may refer to the activities being executed, the performers working on the case, and the values of various data elements linked to the case. Using clustering algorithms it would for example be possible to show a positive correlation between the size of an order or its handling time and the involvement of specific people.

Orthogonal to the three perspectives (process, organization, and case), the result of a mining effort may refer to *logical* issues and/or *performance* issues. For example, process mining can focus on the logical structure of the process model (e.g., the Petri net shown in Figure 12.1(a)) or on performance issues such as flow time. For mining the organizational perspective, the emphasis can be on the roles or the social network (cf. Figure 12.1(b) and (c)) or on the utilization of performers or execution frequencies. To illustrate the fact that the three perspectives and the type of question (logical or performance oriented) are orthogonal, some examples are given in Table 12.2.

To address the three perspectives and the logical and performance issues we have developed a set of tools including EMiT [1], Thumb [19], and MinSoN [3]. These tools share a common XML format. Recently, the functionality of these three tools have been merged into the *ProM Framework*. The ProM tool not only supports variants of the α-algorithm: It also supports alternative approaches, e.g., approaches based on genetic algorithms. For more details we refer to http://www.processmining.org.

perspective	*examples of logical properties*	*examples of performance properties*
process perspective	activity A is always followed by B; activities C and D may be executed in parallel	the average processing time of activity A is 35 minutes; activity A is executed for 80 percent of the cases
organisational perspective	John and Mary are in the same team; Pete is the manager of department D	John handles on average 30 cases per day; Mary and Pete work together on 50 percent of the cases
case perspective	cases of more than 5000 euro are handled by John; activity A is only executed for private customers	80 percent of cases of more than 5000 euro are handled within 2 days; the average flow time of cases handled by John and Mary is 2 weeks

Table 12.2 Some examples of properties that may be investigated using process mining.

12.3 PROCESS MINING WITH THE α-ALGORITHM

In this chapter we focus on the process perspective. In fact, we consider a specific algorithm: the α-algorithm. Before describing the algorithm, we first discuss the input format.

12.3.1 Input

Table 12.1 shows an event log. The basic algorithm only considers the case id and the activity id and not the timestamp and originator of the event. For the α-algorithm the ordering of events within a case is relevant while the ordering of events amongst cases is of no importance. In Table 12.1 it is important that for case 1 activity A is followed by B within the context of case 1 and not that activity A of case 1 is followed by activity A of case 2. Therefore, we define an event log as follows. Let T be a set of activities. $\sigma \in T^*$ is an *event trace*, i.e., an arbitrary sequence of activity identifiers. $W \subseteq T^*$ is an *event log*, i.e., a set of event traces. Note that since W is a set and not a multiset (bag), every event trace can appear only once in a log. In an event log like the one shown in Table 12.1 this is not the case. However, for inferring the structure of a process with the α algorithm the frequency of an event trace is irrelevant, i.e., it does not add information. In more practical

mining tools as presented in Section 12.4.2 frequencies become important. If we use this notation to describe the log shown in Table 12.1 we obtain the set $W = \{ABCD, ACBD, AED\}$. Note that cases 1 and 3 have event trace $ABCD$, cases 2 and 4 have trace $ACBD$, and case 5 is the only one having trace AED. Also note that when dealing with noise, frequencies are of the utmost importance, cf. Section 12.4.2 and [18]. However, for the moment we abstract from noise and simply look at the presence of a trace rather than its frequency.

To find a process model on the basis of an event log, the log should be analyzed for causal dependencies, e.g., if an activity is always followed by another activity it is likely that there is a causal relation between both activities. To analyze these relations we introduce the following notations. Let W be an event log over T, i.e., $W \subseteq T^*$. Let $a, b \in T$:

- $a >_W b$ iff there is a trace $\sigma = t_1 t_2 t_3 \ldots t_n$ and $i \in \{1, \ldots, n-1\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$,
- $a \rightarrow_W b$ iff $a >_W b$ and $b \not>_W a$,
- $a \#_W b$ iff $a \not>_W b$ and $b \not>_W a$, and
- $a \|_W b$ iff $a >_W b$ and $b >_W a$.

Consider the event log $W = \{ABCD, ACBD, AED\}$ (i.e., the log shown in Table 12.1). Relation $>_W$ describes which activities appeared in sequence (one directly following the other). Clearly, $A >_W B$, $A >_W C$, $A >_W E$, $B >_W C$, $B >_W D$, $C >_W B$, $C >_W D$, and $E >_W D$. Relation $\rightarrow_W$ can be computed from $>_W$ and is referred to as the *(direct) causal relation* derived from event log W. $A \rightarrow_W B$, $A \rightarrow_W C$, $A \rightarrow_W E$, $B \rightarrow_W D$, $C \rightarrow_W D$, and $E \rightarrow_W D$. Note that $B \not\rightarrow_W C$ because $C >_W B$. Relation $\|_W$ suggests concurrent behavior, i.e., potential parallelism. For log W activities B and C seem to be in parallel, i.e., $B \|_W C$ and $C \|_W B$. If two activities can follow each other directly in any order, then all possible interleavings are present and therefore they are likely to be in parallel. Relation $\#_W$ gives pairs of transitions that never follow each other directly. This means that there are no direct causal relations and parallelism is unlikely.

12.3.2 The algorithm

The α algorithm uses notions such as $>_W$, $\rightarrow_W$, $\|_W$, and $\#_W$ to obtain information about the underlying process. The α algorithm represents the discovered process in terms of a Petri net. Let W be an event log over T. $\alpha(W)$ is defined as follows.

1. $T_W = \{t \in T \mid \exists_{\sigma \in W} t \in \sigma\}$ (the set of activities appearing in the log),
2. $T_I = \{t \in T \mid \exists_{\sigma \in W} t = \mathit{first}(\sigma)\}$ (the set of initial activities),
3. $T_O = \{t \in T \mid \exists_{\sigma \in W} t = \mathit{last}(\sigma)\}$ (the set of final activities),

4. $X = \{(A, B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$ (all causality relations),
5. $Y = \{(A, B) \in X \mid \forall_{(A', B') \in X} A \subseteq A' \wedge B \subseteq B' \Longrightarrow (A, B) = (A', B')\}$ (only the minimal causality relations),
6. $P_W = \{p_{(A,B)} \mid (A, B) \in Y\} \cup \{i_W, o_W\}$ (the set of places in the resulting Petri net, $p_{(A,B)}$ is a place connecting transitions in A with transitions in B, i_W is the unique input place denoting the start of the process, and o_W is the unique output place denoting the end of the process),
7. $F_W = \{(a, p_{(A,B)}) \mid (A, B) \in Y \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y \wedge b \in B\} \cup \{(i_W, t) \mid t \in T_I\} \cup \{(t, o_W) \mid t \in T_O\}$ (the set of connecting arcs in the resulting Petri net), and
8. $\alpha(W) = (P_W, T_W, F_W)$ (the resulting Petri net with places P_W, transitions T_W, and arcs F_W).

The α algorithm transforms a log W into a Petri net (P_W, T_W, F_W). The algorithm only uses basic mathematics, the relations $>_W$, $\rightarrow_W$, $\|_W$, and $\#_W$, and the functions *first* and *last* to get the first and last element from a trace.

To illustrate the α algorithm we show the result of each step using the log $W = \{ABCD, ACBD, AED\}$ (i.e., a log like the one shown in Table 12.1):

1. $T_W = \{A, B, C, D, E\}$,
2. $T_I = \{A\}$,
3. $T_O = \{D\}$,
4. $X = \{(\{A\}, \{B\}), (\{A\}, \{C\}), (\{A\}, \{E\}), (\{B\}, \{D\}), (\{C\}, \{D\}), (\{E\}, \{D\}), (\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}), (\{D\}), (\{C, E\}), (\{D\})\}$,
5. $Y = \{(\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\}, \{D\})\}$,
6. $P_W = \{i_W, o_W, p_{(\{A\},\{B,E\})}, p_{(\{A\},\{C,E\})}, p_{(\{B,E\},\{D\})}, p_{(\{C,E\},\{D\})}\}$,
7. $F_W = \{(i_W, A), (A, p_{(\{A\},\{B,E\})}), (p_{(\{A\},\{B,E\})}, B) \ldots, (D, o_W)\}$, and
8. $\alpha(W) = (P_W, T_W, F_W)$ (as shown in Figure 12.2).

It is interesting to note that $\alpha(W)$ shown in Figure 12.2 differs from the Petri net shown in Figure 12.1. This may suggest that the result is not correct. However, from a behavioral point of view Figure 12.2 and Figure 12.1(a) are equivalent if we abstract from the AND-split and AND-join. Note that every event trace in $W = \{ABCD, ACBD, AED\}$ can be realized by Figure 12.2. Also every possible firing sequence corresponds to an event trace in W. Therefore, we conclude that although Figure 12.2 and Figure 12.1(a) differ, the α algorithm is able to correctly mine the log shown in Table 12.1.

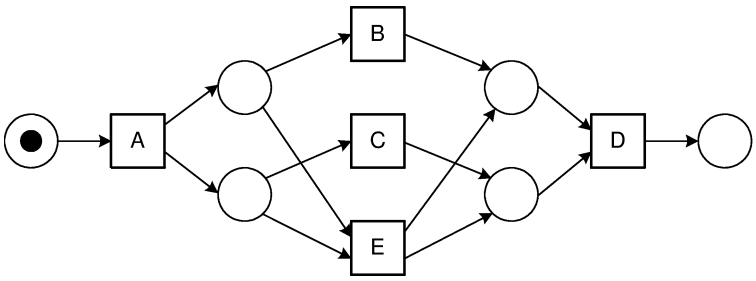

Fig. 12.2 Another process model corresponding to the event log shown in Table 12.1. In Section 12.4 we will discuss the problem of "invisible activities", i.e., activities that are not recorded in the log.

12.3.3 How does it work?

The fact that the α algorithm is able to "discover" Figure 12.2 based on the event log $W = \{ABCD, ACBD, AED\}$ triggers the question: *How does it work?* To understand the basic idea of the α algorithm consider Figure 12.3. The algorithm assumes that two activities x and y (i.e, transitions) are connected through some place if and only if $x \rightarrow_W y$ (Figure 12.3(a)). If activities x and y are concurrent, then they can occur in any order, i.e., x may be directly followed by y or vice versa. Therefore, the α algorithm assumes activities x and y are concurrent if and only if $x \|_W y$. This is illustrated by Figure 12.3(b). If $x \rightarrow_W y$ and $x \rightarrow_W z$, then there have to be places connecting x and y on the one hand and x and z on the other hand. This can be one place or multiple places. If $y \|_W z$, then there should be multiple places to enable concurrency (cf. Figure 12.3(b)). If $y \#_W z$, then there should be a single place to ensure that only one branch is chosen (cf. Figure 12.3(c)). Note that in the latter case y and z never follow one another directly as expressed by $y \#_W z$ (i.e., $y \not>_W z$ and $z \not>_W y$). Figure 12.3(d) shows the AND-join (i.e., counterpart of the AND-split shown in Figure 12.3(b)) and Figure 12.3(e) shows the XOR-join (i.e., counterpart of the XOR-split shown in Figure 12.3(c)).

The basic relations shown in Figure 12.3 are the starting point for the α algorithm. Note that the relations do not always hold, i.e., one can think of them as heuristics. For example, it is assumed that the log is *complete* with respect to $>_W$. (Note that $\rightarrow_W$, $\|_W$, and $\#_W$ are derived from $>_W$.) This implies that if one activity can be followed by another this should happen at least once in the log. We will return to the issue of completeness in Section 12.4.2.

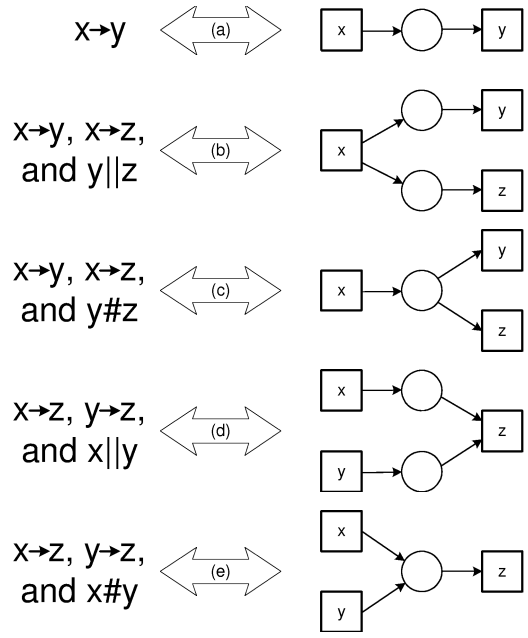

Fig. 12.3 Relating the log-based relations $>_W$, $\rightarrow_W$, $\|_W$, and $\#_W$ to basic Petri-net constructs.

12.3.4 Examples

Figure 12.4 is an example of a Petri in which an AND-split and OR-split are embedded in an AND-split. Given a complete event log, this kind of nesting will not harm the α algorithm in rediscovering the original Petri net.

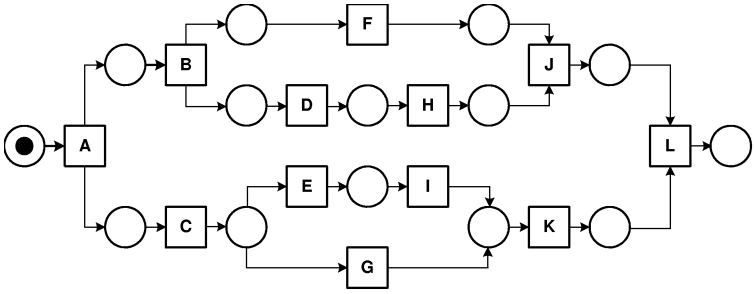

Fig. 12.4 An example of a Petri net with nested AND/XOR-splits correctly mined by the α algorithm.

In Figure 12.5 an example of a Petri net with loops is given. Remark that an event log with all possible event traces of this Petri net is an *infinite set*. However, a *complete* event log does not need to contain all possible traces. A sufficiently large subset with, on a binary level (i.e. with respect to $>_W$) all possible pairs is sufficient. If we have such a complete event log, the α algorithm will, without any problem, correctly rediscover the Petri net of Figure 12.5.

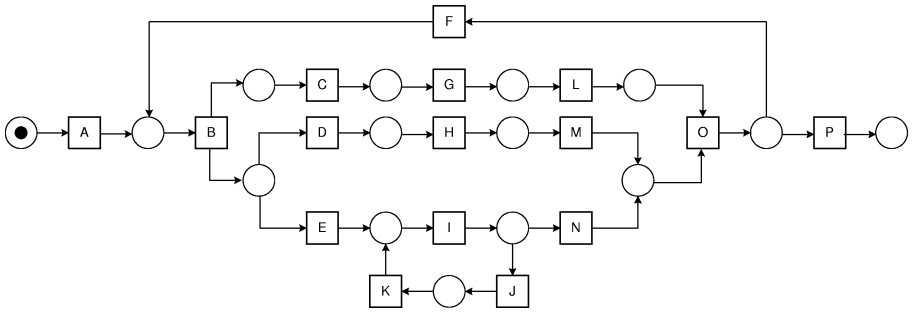

Fig. 12.5 Example of a more complex Petri correctly mined by the α algorithm.

The Petri net of the last example (Figure 12.6) is less abstract and specifies the interactions between a contractor and a subcontractor. First, the contractor sends an order to the subcontractor. Then, the contractor sends a detailed specification to the subcontractor and the subcontractor sends a cost statement to the contractor. Based on the specification the subcontractor manufactures the desired product and sends it to the contractor. There is no

clear owner of the resulting interaction process, but the combined information registered by both parties contains enough information to mine the process (i.e. the process itself and other aspects of it).

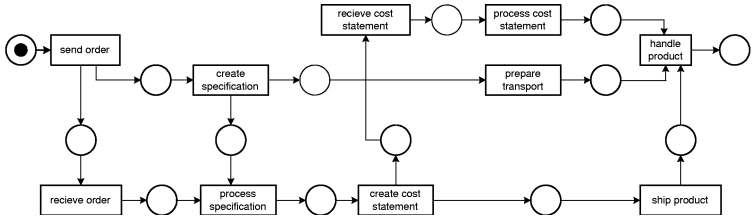

Fig. 12.6 The interaction between contractor and subcontractor.

12.4 LIMITATIONS OF THE α-APPROACH AND POSSIBLE SOLUTIONS

In this paper, we only consider the basic α algorithm to illustrate the concept of process mining. The α algorithm focuses exclusively on the process perspective (i.e., control-flow). As indicated in Section 12.2, process mining can be used to analyze other perspectives (to answer "Who?" and "What?" questions). Despite its focus, the basic α algorithm is still unable to successfully discover some processes. In this section we identify two classes of problems: logical problems and problems resulting from noise (incorrectly logged data), exceptions (rare events not corresponding to the "normal" behavior), and incompleteness (i.e., too few observations).

12.4.1 Logical problems

In [6] a formal characterization is given for the class of nets that can be mined correctly. It turns out that assuming a weak notion of completeness (i.e., if one activity can be followed by another this should happen at least once in the log), any so-called SWF-net without short loops and implicit places can be mined correctly. SWF-nets are Petri nets with a single source and sink place satisfying some additional syntactical requirements such as the free-choice property. In this chapter, we will not elaborate on formal characterizations of the class of processes that can be successfully mined. Instead, we focus on the practical limitations of the α algorithm, i.e., the problems when dealing with invisible activities, duplicate activities, short loops, etc.

12.4.1.1 Invisible activities One of the basic assumptions of process mining is that each event (i.e., the occurrence of an activity for a specific case) is registered in the log. Clearly, it is not possible to find information about ac-

tivities that are not recorded. However, given a specific language it is possible to register that there is a so-called "hidden activity". Consider, for example, Table 12.1 where A, B, and C are visible but the AND-split in-between A, and B and C is not. Clearly, the basic α algorithm is unable to discover activities not appearing in the log. Therefore, the Petri net shown in Figure 12.2 is different from the Petri net shown in Figure 12.1(a). However, both nets are equivalent if we abstract from the AND-split and AND-join. Unfortunately, this is not always the case. Consider for example Figure 12.2 where activity E is not visible. The resulting log would be $W = \{ABCD, ACBD, AD\}$ and the α algorithm would be unable to construct the correct model, i.e., $\alpha(W) = (\{i_W, o_W, p_{(\{A\},\{B\})}, p_{(\{A\},\{C\})}, p_{(\{B\},\{D\})}, p_{(\{C\},\{D\})}, p_{(\{A\},\{D\})}\}, \{A, B, C, D\}, \{(i_W, A), (A, p_{(\{A\},\{B\})}), \ldots, (A, p_{(\{A\},\{D\})}), (p_{(\{A\},\{D\})}, D), (D, o_W)\})$. The resulting net is shown in Figure 12.7, i.e., the original net shown in Figure 12.2 without E but with an additional place connecting A and D. Note that the resulting model does not allow for the event trace AD.

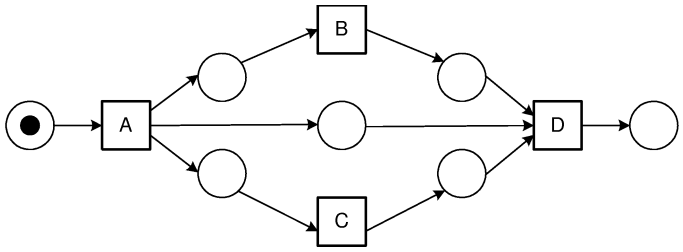

Fig. 12.7 If activity E is not visible, the algorithm returns an incorrect model because it does not allow for AD.

12.4.1.2 Duplicate activities The problem of duplicate activities refers to the situation that one can have a process model (e.g., a Petri net) with two nodes referring to the same activity. Suppose that in Table 12.1 and Figure 12.1 activity E is renamed to B (see Figure 12.8). Clearly, the modified log could be the result of the modified process model. However, it becomes very difficult to automatically construct a process model from Table 12.1 with E renamed to B because it is not possible to distinguish the"B" in case 5 from the "B's" in the other cases. Note that the presence of duplicate activities is related to hidden activities. Many processes with hidden activities but with no duplicate activities can be modified into equivalent processes with duplicate activities but with no hidden activities.

12.4.1.3 Non-free-choice constructs Free-choice Petri nets are Petri nets where there are no two transitions consuming from the same input place but where one has an input place which is not an input place of the other [9]. This excludes the possibility to merge choice and synchronization into one

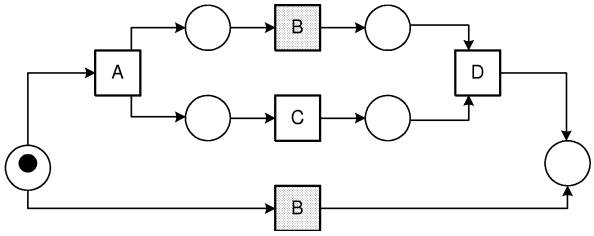

Fig. 12.8 A process model with duplicate activities.

construct. Free-choice Petri nets are a well-known and widely used subclass of Petri nets. However, many processes cannot be expressed in terms of a free-choice net. Unfortunately, most of the mining techniques (also those that are not using Petri nets) assume process models corresponding to the class of free-choice nets. Non-free-choice constructs can be used to represent "controlled choices", i.e., situations where the choice between two activities is not determined inside some node in the process model but may depend on choices made in other parts of the process model. Clearly, such non-local behavior is difficult to mine for mining approaches primarily based on binary information ($a >_W b$) and may require many observations.

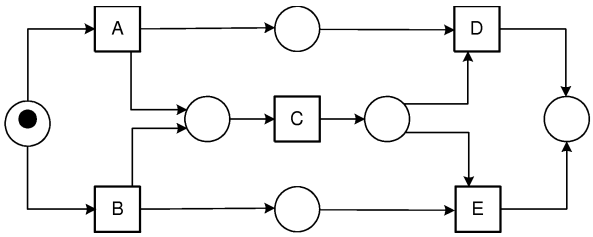

Fig. 12.9 A process model with a non-free-choice construct.

Figure 12.1 is free-choice since synchronization (activity D) is separated from the choice between B and C, and E. Figure 12.9 shows a non-free-choice construct. After executing activity C there is a choice between activity D and activity E. However, the choice between D and E is "controlled" by the earlier choice between A and B. Note that activities D and E are involved in a choice but also synchronize two flows. Clearly such constructs are difficult to mine since the choice is non-local and the mining algorithm has to "remember" earlier events.

To illustrate that there are also non-free-choice constructs that can be mined correctly using the α algorithm we consider Figure 12.10. Now the choice can be detected because of the two new activities X and Y. Note that X may be directly followed by D but not by E. Hence the place in-between X and D is discovered.

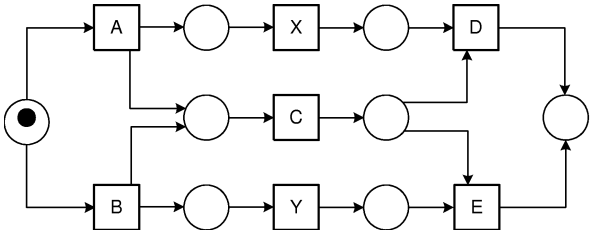

Fig. 12.10 A process model with a non-free-choice construct that can be mined correctly.

12.4.1.4 Short loops In a process it may be possible to execute the same activity multiple times. If this happens, this typically refers to a loop in the corresponding model. Figure 12.11 shows an example with a loop. After executing activity B, activity C can be executed arbitrarily many times, i.e., possible event sequences are BD, BCD, BCCD, BCCCD, etc. Loops like the one involving activity C are easy to discover. However, loops can also be used to jump back to any place in the process. For more complex processes, mining loops is far from trivial since there are multiple occurrences of the same activity in a given case. Some techniques number each occurrence, e.g., B1 C1 C2 C3 D1 denotes BCCCD. These occurrences are then mapped onto a single activity.

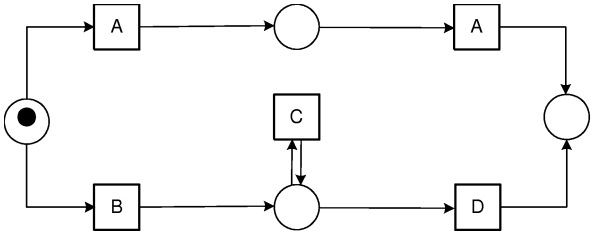

Fig. 12.11 A process model with a loop.

As illustrated by Figure 12.11 there is a relation between loops and duplicate activities. In Figure 12.11 activity A is executed multiple times (i.e., twice) but is not in a loop. Many mining techniques make some assumptions about loops which restricts the class of processes that can be mined correctly.

The logical problems described all apply to the α algorithm. Some of the problems can be resolved quite easily by using a more refined algorithm. Other problems are more fundamental and indicate theoretical limits [6].

12.4.2 Noise, exceptions and incompleteness

The formal approach presented in the preceding section presupposes perfect information: (i) the log must be complete (i.e., if an activity can follow another activity directly, the log should contain an example of this behavior) and (ii) we assume that there is no noise in the log (i.e., everything that is registered in the log is correct). However, in practical situations logs are rarely complete and/or noise free. Especially the differentiation between errors, low frequency activities, low frequency activity sequences, and exceptions is problematic. Therefore, in practice, it becomes more difficult to decide if between two activities say A and B, one of the three basic relations (i.e., $A \rightarrow_W B$, $A \#_W B$, or $A \|_W B$) holds. For instance the causality relation as used in the α-algorithm ($A \rightarrow_W B$) only holds if and only if in the log there is a trace in which A is directly followed by B (i.e., the relation $A >_W B$ holds) and there is no trace in which B is directly followed by A (i.e., not $B >_W A$). However, in a noisy situation one erroneous example can completely mess up the derivation of a right conclusion. Even if we have thousands of log traces in which A is directly followed by B, then one $B >_W A$ example based on an incorrect registration, will prevent a correct conclusion. As noted before, frequency information is not used in the formal approach. For this reason heuristic mining techniques are developed which are less sensitive to noise and the incompleteness of logs.

As an illustration of a heuristic approach we shortly discuss the ideas to discover the causality relation as implemented in the heuristic mining tool Little Thumb [18]. In this approach a frequency based metric is used to indicate how certain we are that there is truly a causal relation between two events A and B (notation $A \Rightarrow_W B$). The calculated $\Rightarrow_W$ values between the events of an event log are used in a heuristic search for the right relations between events (i.e. $A >_W B$, $A \#_W B$ or $A \|_W B$). Below we first define the $\Rightarrow_W$ metric. After that we will illustrate how we can use this metric in a simple heuristic in which we search for reliable causal relations (the $A \rightarrow_W B$ relation).

Let W be an event log over T, and $a, b \in T$:

- $|a >_W b|$ is the number of times $a >_W b$ occurs in W,
- $a \Rightarrow_W b = \left(\dfrac{|a >_W b| - |b >_W a|}{|a >_W b| + |b >_W a| + 1} \right)$

First, remark that the value of $a \Rightarrow_W b$ is always between -1 and 1. Some simple examples demonstrate the rationale behind this definition. If we use this definition in the situation that, in 5 traces, activity A is directly followed by activity B but the other way around never occurs, the value of $A \Rightarrow_W B = 5/6 = 0.833$ indicating that we are not completely sure of the causality relation (only 5 observations possibly caused by noise). However if there are 50 traces in which A is directly followed by B but the other way around never occurs,

the value of $A \Rightarrow_W B = 50/51 = 0.980$ indicates that we are pretty sure of the causality relation. If there are 50 traces in which activity A is directly followed by B and noise caused B to follow A once, the value of $A \Rightarrow_W B$ is $49/52 = 0.94$ indicating that we are pretty sure of a causal relation.

A high $A \Rightarrow_W B$ value strongly suggests that there is a causal relation between activity A and B. But what is a high value, what is a good threshold to take the decision that B truly depends on A (i.e. $A \rightarrow_W B$ holds)? The threshold appears sensitive for the amount of noise, the degree of concurrency in the underlying process, and the frequency of the involved activities.

However, it appears unnecessary to use a threshold value. After all, we know that each non-initial activity must have at least one other activity that is its cause, and each non-final activity must have at least one dependent activity. Using this information in a heuristic approach we can limit the search and take *the best* candidate (with the highest $A \Rightarrow_W B$ score). This simple heuristic helps us enormously in finding reliable causality relations even if the event log contains noise. As an example we have applied the heuristic to an event log from the Petri net of Figure 12.1. Thirty event traces are used (nine for each of the three possible traces and three incorrect traces: ABCED, AECBD, AD). We first calculate the $\Rightarrow$-values for all possible activity combinations. The result is displayed in the matrix below.

$\Rightarrow_W$	A	B	C	D	E
A	0.0	0.909	0.900	0.500	0.909
B	0.0	0.0	0.0	0.909	0.0
C	0.0	0.0	0.0	0.900	0.0
D	-0.500	-0.909	-0.909	0.0	-0.909
E	0.0	0.0	0.0	0.909	0.0

As an illustration we now apply the basic heuristic on this matrix. We can recognize the initial activity A, it is the activity without a positive value in the A-column. For the dependent activity of A we search for the highest value in row A of the matrix. Both B and E are high (0.909). We arbitrarily choose B. If we use the matrix to search for the cause for B (the highest value of the B column) we will again find A as the cause for B. D is the depending activity of B (D is the highest value of the B row). The result of applying the same procedure on activity B, C, and E is presented in Figure 12.12; remark that only the causal relations are depicted in a so called dependency graph. The numbers in the activity boxes indicate the frequency of the activity, the numbers on the arcs indicate the reliability of each causal relation and the numbers on the nodes the frequencies. In spite of the noise, the causal relations are correctly mined.

The illustrated heuristic procedure is not complete. For example, we need searching procedures for the other basic relations (i.e. $a\#_W b$ and $a\|_W b$).

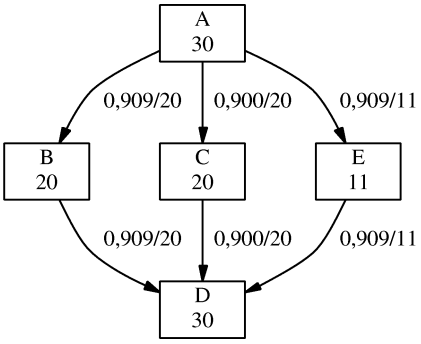

Fig. 12.12 A dependency graph resulting from applying the heuristic approach to a noisy log based on the Petri-net of Figure 12.1.

Given the correct basic relations we can use the α-algorithm to construct a Petri net. In [18] the experimental results of such an approach to noisy data are presented.

12.5 CONCLUSION

This chapter introduced the topic of process mining by first providing an overview and then zooming in on a specific algorithm for the process perspective (i.e., control flow): the α algorithm. It is important to realize that this algorithm only tackles one of the cells shown in Table 12.2 (the top-left one). For the other cells other approaches are needed. However, even within this single cell there are many challenges as demonstrated in this chapter. The wide applicability of process mining makes is worthwhile to tackle problems such as noise, incompleteness, etc. For more information and to download mining tools, we refer to http://www.processmining.org.

Acknowledgements

The authors would like to thank Boudewijn van Dongen, Ana Karla Alves de Medeiros, Minseok Song, Laura Maruster, Eric Verbeek, Monique Jansen-Vullers, Hajo Reijers, and Peter van den Brand for their on-going work on process mining techniques and tools at Eindhoven University of Technology. Parts of this chapter have been based on earlier papers with these researchers.

12.6 EXERCISES

Exercise 12.6.1 *Consider the Petri net shown in Figure 12.13.*

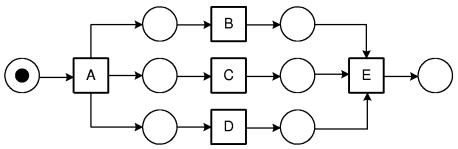

Fig. 12.13 A simple parallel Petri-net.

1. *Determine the event log W with all possible traces.*
2. *Try to determine a $>$-complete event log W' with W' a real subset of W ($W' \subset W$).*

Answer:

1. 6, $\{ABCDE, ABDCE, ACBDE, ACDBE, ADBCE, ADCBE\}$.
2. A example of a complete log is $W' = \{ABCDE, ABDCE, ACDBE, ADCBE\}$. Note that this is a real subset.

Exercise 12.6.2 *Given an event log $W = \{AFBCGD, AFCBGD, AED\}$ use the eight steps of the α algorithm to construct an accompanying Petri net.*

Answer:

1. $T_W = \{A, B, C, D, E, F, G\}$,
2. $T_I = \{A\}$,
3. $T_O = \{D\}$,
4. $X = \{(\{A\}, \{F\}), (\{A\}, \{E\}), (\{A\}, \{E, F\}), (\{F\}, \{B\}), (\{F\}, \{C\}),$ $(\{B\}, \{G\}), (\{C\}, \{G\}), (\{G\}, \{D\}), (\{E\}, \{D\}), (\{G, E\}, \{D\})\}$,
5. $Y = \{(\{A\}, \{E, F\}), (\{F\}, \{B\}), (\{F\}, \{C\}), (\{B\}, \{G\}),$ $(\{C\}, \{G\}), (\{G, E\}, \{D\})\}$,
6. $P_W = \{i_W, o_W, p_{(\{A\},\{E,F\})}, p_{(\{F\},\{B\})}, p_{(\{F\},\{C\})}, p_{(\{B\},\{G\})},$ $p_{(\{C\},\{G\})} p_{(\{G,E\},\{D\})}\}$,
7. $F_W = \{(i_W, A), (A, p_{(\{A\},\{E,F\})}), (F, p_{(\{F\},\{B\})}), (F, p_{(\{F\},\{C\})}), (B, p_{(\{B\},\{G\})}),$ $(C, p_{(\{C\},\{G\})}), (G, p_{(\{G,E\},\{D\})}), (E, p_{(\{G,E\},\{D\})})\}$, and
8. $\alpha(W) = (P_W, T_W, F_W)$ is the Petri net as shown in Figure 12.1 with the two non-observable activities AND-split and AND-join replaced by F and G.

Exercise 12.6.3 *Consider the following log $W = \{ABCDE, ABDCE, ACBDE, ACDBE, ADBCE, ADCBE\}$ originating from the Petri net of Exercise 12.6.1. Determinate $A \Rightarrow_W B$ and $B \Rightarrow_W C$.*

Answer: $A \Rightarrow_W B = 2/(2+0+1) = 0.66$ and $B \Rightarrow_W C = (2-2)/(2+2+1) = 0.0$.

Exercise 12.6.4 *Given the following event log* $W = [ABCDE, ABDCE, ACBDE, ACDBE, ADBCE, ADCBE, ABCDE, ABDCE, ACBDE, ACDBE, ADBCE, ADCBE]$ [2] *which originated from the Petri net of Exercise 12.6.2. Follow the heuristic of Subsection 12.4.2 to construct a dependency graph as presented in Figure 12.12.*

Answer: First calculate de different $\Rightarrow$-values for all possible activity combinations.

$\Rightarrow_W$	A	B	C	D	E
A	0.0	0.8	0.8	0.8	0.0
B	-0.8	0.0	0.0	0.0	0.8
C	-0.8	0.0	0.0	0.0	0.8
D	-0.8	0.0	0.0	0.0	0.8
E	0.0	-0.8	-0.8	-0.8	0.0

Then use the basic heuristics of Subsection 12.4.2. Each non-initial activity must have at least one activity that caused it (select the best candidate), and each non-final activity must have one dependent activity (select the best candidate). The resulting dependency graph is given be in Figure 12.14.

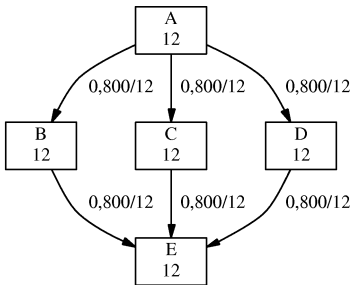

Fig. 12.14 The resulting dependency graph.

[2]To express the multiple appearing of traces we formally have to use the bag or multi set notation instead of the set notation.

REFERENCES

1. W.M.P. van der Aalst and B.F. van Dongen. Discovering Workflow Performance Models from Timed Logs. In Y. Han, S. Tai, and D. Wikarski, editors, *International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, volume 2480 of *Lecture Notes in Computer Science*, pages 45–63. Springer-Verlag, Berlin, 2002.

2. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT press, Cambridge, MA, 2002.

3. W.M.P. van der Aalst and M. Song. Mining Social Networks: Uncovering interaction patterns in business processes. In J. Desel, B. Pernici, and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer-Verlag, Berlin, 2004.

4. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

5. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.

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

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

8. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

9. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, Cambridge, UK, 1995.

10. D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. In P. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. Snodgrass, editors, *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168. Morgan Kaufmann, 2001.

11. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.

12. IDS Scheer. ARIS Process Performance Manager (ARIS PPM): Measure, Analyze and Optimize Your Business Process Performance (whitepaper). IDS Scheer, Saarbruecken, Gemany, http://www.ids-scheer.com, 2002.

13. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading MA, 1998.

14. M. zur Mühlen and M. Rosemann. Workflow-based Process Monitoring and Controlling - Technical and Organizational Issues. In R. Sprague, editor, *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS-33)*, pages 1–10. IEEE Computer Society Press, Los Alamitos, California, 2000.

15. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.

16. M. Sayal, F. Casati, U. Dayal and M.C. Shan. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.

17. J. Scott. *Social Network Analysis*. Sage, Newbury Park CA, 1992.

18. A.J.M.M. Weijters and W.M.P. van der Aalst. Workflow Mining: Discovering Workflow Models from Event-Based Data. In C. Dousson, F. Höppner, and R. Quiniou, editors, *Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 78–84, 2002.

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