Chapter 11 - Process Monitoring

The Context of Process Monitoring

- Process monitoring is about using the data generated by the execution of a business process in order to extract insights about the actual performance of the process and to verify its conformance with respect to norms, policies or regulations.
- Business process executions generate event records. A collection of even records is called an **event log**.

Types of Process Monitoring Techniques

- **Offline Process Monitoring**: the input is event logs that covers cases completed in a period of time, an historic analysis of process executions. Provide the picture of the performance of the process, why it's performing badly and the conformance of the process with the rules and expected behaviors.
- **Online Process Monitoring**: the input is incomplete traces of ongoing cases to assess the performance of currently running process instances and generate triggers in the case objectives or rules are not fulfilled. Ex: a customer request remains unanswered for a long time.
- There also exists **statistics-based** (uses mean, standard deviation, min, max to measure performance) and **model-based** (analyze the execution of cases based on process models) techniques.

Process Performance Dashboards

- **Process Performance Dashboards**: graphical representation of one or more performance measures or other characteristics of a business process.

Operational Dashboards

- **Target**: process participants and operational managers.
- Display performance of ongoing/completed cases in a way that allows process participants and their managers to plan their short-term work.
- **Typical measures**: number of ongoing cases classified by type of cases. Ex: cases that are on time, overdue, at risk of being overdue. Can be represented by pie charts or histograms.

Tactical Dashboards

- **Target**: process owners and functional managers.
- Display performance of a process **over a long period of time** (ex: a month or a quarter) to highlight bad performance with their possible causes, bottlenecks, defects and deviations.
- **Typical measures**: number of completed cases, cycle times, waiting times, processing times, defect rate, cost per case etc... Represented by histograms and can include statistics (mean, max, min, st deviation...).

- Can be defined at the level of "per-task" and "per-team".
- Provide **longitudinal view**: variation of a performance measure over time (can observe trends this way). Also **cross-sectional** to show the distribution of a performance measure according to a given classification of cases. Ex: cases by geographical region.

Strategic Dashboards

- **Target**: executive managers.
- Provide a picture of the performance of groups of processes along multiple performance dimensions.
- Built by aggregating performance measures in two ways:
  - Aggregating performance measures for individual processes into performance measures for groups of processes.
  - Aggregating multiple performance measures related the same performance dimension into a single measure (ex: measures related to efficiency).

Introduction to Process Mining

- **Process Mining**: techniques that analyze the performance and conformance of business processes based on event logs produced during their execution. **They complement tactical dashboards**.

Process Mining Techniques

Automated Process Discovery

- Takes event log as input and produces a business process model that matches the behavior observed or implied in the event log.

Conformance Checking

- Take event log and process model as input and give a list of differences between the process model and the event log as output. Very useful to catch exceptions that are not represented. Can also take rules instead of a model as input to verify if the log fulfills these business rules.

Performance Mining

- Take event log and process model as input and produce an enhanced process model as output, makes it easy to pinpoint bottlenecks. The model provided can be provided by the analyst or an automatically-discovered one.

Variants Analysis

- Takes two event logs (two variants of a process - typically a positive and negative) as input and produces a list of differences as output. Ex: one log may be cases that the customer ended up satisfied and the other cases where the customer wasn't satisfied. Helps knowing why the execution of the process doesn't always end up in a good outcome.

Event Logs

- **Event Log**: collection of timestamped event records. Each event record tells the execution of a work item (hence a task). Can tell if it has started or completed or if another relevant event has happened.
- **Minimum Attributes for an Event Log**:
    - **Case ID** - in which case the event occurred
    - **Event Class** - which task the event refers to
    - **Timestamp** - when the event occurred

Challenges for Log Data Extraction

- **Correlation Challenge**: problems identifying which case the event belongs to, therefore need to investigate which attribute will serve as the case ID. Usually order number, invoice number or shipment number.
- **Timestamping Challenge**: lots of systems don't do logging as a primary task and only do it during idle times or little load, therefore lots of events will appear with the same timestamp, not only if different time zones are involved. Resolved with domain knowledge (if we know the order the events occur).
- **Longevity Challenge**: for long running processes, some cases might still be incomplete and we might be mixing incomplete with complete cases and getting a distorted picture of the process. Consider removing incomplete events from an event log.
- **Granularity Challenge**: the granularity of an event log is much finer that each task might be mapping to a set of event so they will show repeatedly in the logs even though they represent only a single task. Difficulty defining a mapping between these two levels of abstraction.

Execution Log Format

- Log format
  - (caseID, activity)
- Example
  - *Check Invoice for Invoice No. 4567 completed on 12.11.2010 at 9:19:57*
  - *Function StoreCustomerData(„Müller“, c1987, „Bad Bentheim“) completed on 12.11.2010 at 9:22:24*
  - *Send Invoice for Invoice No. 4567 completed on 12.11.2010 at 9:23:18*
- *Resulting Log*
  - *(4567, Check Invoice), (c1987, StoreCustomerData), (4567, Send Invoice), etc.*

Execution Log

- Further abstraction
  - A‘s and B‘s
  - **(case id, task id)**
- Additional information
  - Event type, time, resource, data
  - Not considered here
- Assumption
  - Activity execution captured by one event
  - No intermediate activities

```
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
```

Execution Logs

Execution sequences:

Case 1: ABCD

Case 2: ACBD

Case 3: ABCD

Case 4: ACBD

Case 5: EF

Resulting workflow log:
***W* = {ABCD, ACBD, EF}**

```
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
```

Automated Process Discovery

Dependency Graphs

- **Dependency Graph**: graph where each node represents an event class (task) and each arc represents a "directly follows" relation.
- An arc exists between two event classes A and B if there is at least one trace in which B comes immediately after A. The arcs in a dependency graph may be annotated with an integer indicating the number of times that B directly follows A (hereby called the absolute frequency).
- Provide **abstraction** since we can remove nodes or arcs in order to obtain a smaller dependency graph.

10 × a,b,c,g,e,h
10 × a,b,c,f,g,h
10 × a,b,d,g,e,h
10 × a,b,d,e,g,h
10 × a,b,e,c,g,h
10 × a,b,e,d,g,h
10 × a,c,b,e,g,h
10 × a,c,b,f,g,h
10 × a,d,b,e,g,h
10 × a,d,b,f,g,h

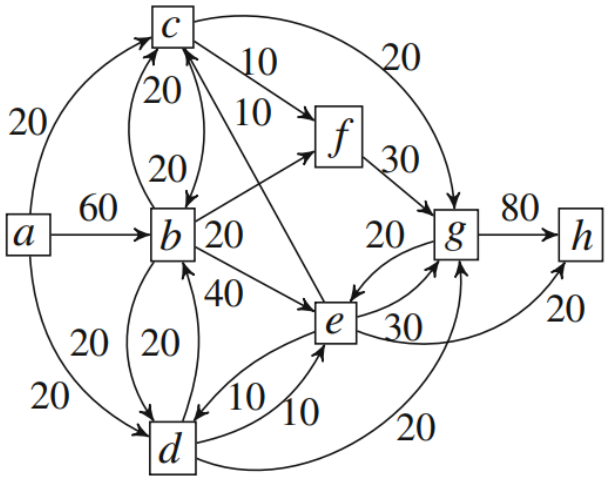


Fig. 11.8 Event log and corresponding dependency graph

The α-Algorithm

- Algorithm for discovering discovering process models from event logs.
- Assumes the event log has **behavioral completeness**: if whenever a task a can be directly followed by a task b in the actual process, then there is at least one case in the event log where we observe ab.
- **This algorithm has two phases**:
  - 1st phase: extract from the workflow log a set of order relations (pairs of tasks that directly follow one another in the log) between pairs of events.
  - 2nd phase: construct the process model in a stepwise fashion from the identified relations.

Log based order relations for pairs of activities a, b ∈ T in a workflow log W:

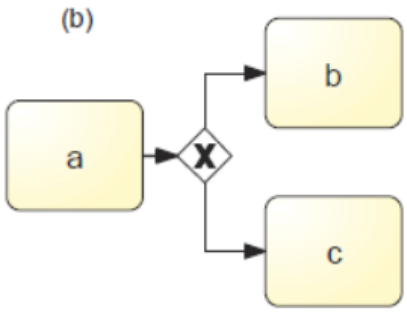
- Direct successor
  $a > b$ i.e. in an execution sequence *b* directly follows a
- Causality
  $a \rightarrow b$ i.e. a > b and not b > a
- Concurrency
  $a \parallel b$ i.e. a >b and b >a
- Exclusiveness
  $a \# b$ i.e. not a >b and not b >a (*Activity pairs which never succeed each other)*

- Idea (a)

$a \rightarrow b$

$a \rightarrow b$ i.e. a > b and not b > a

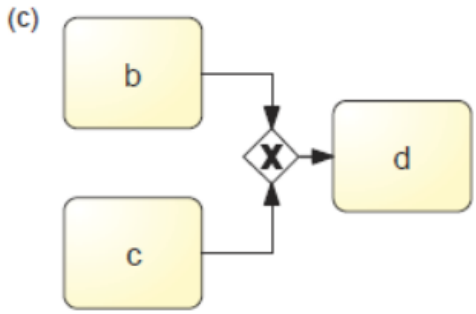
- Idea (b)



$$a \rightarrow b,\ a \rightarrow c \text{ and } b \# c$$

$a \rightarrow b$ i.e. a > b and not b > a
$a \# b$ i.e. not a >b and not b >a

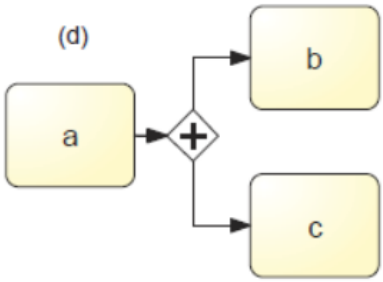
- Idea (c)



$$b \rightarrow d,\ c \rightarrow d \text{ and } b \# c$$

$a \rightarrow b$ i.e. a > b and not b > a
$a \# b$ i.e. not a >b and not b >a

- Idea (d)

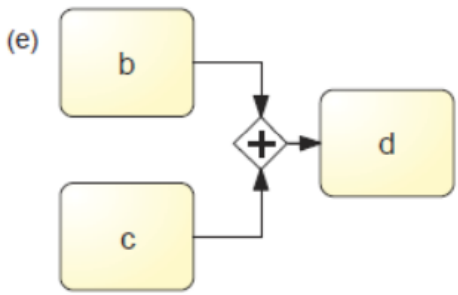


$$a \rightarrow b,\ a \rightarrow c \text{ and } b \parallel c$$

$a \rightarrow b$ i.e. a > b and not b > a

$a \parallel b$ i.e. a >b and b >a

- Idea (e)



$$b \rightarrow d,\ c \rightarrow d \text{ and } b \parallel c$$

$a \rightarrow b$ i.e. a > b and not b > a

$a \parallel b$ i.e. a >b and b >a

Example

Consider the two cases <a, b, g, h, j, k, i, l> and <a, c, d, e, f, g, j, h, i, k, l>.

- The basic order relations $>$ refer to pairs of tasks in which one task directly follows the another. These relations can be directly read from the log:

$a > b$	$h > j$	$i > l$	$d > e$	$g > j$	$i > k$
$b > g$	$j > k$	$a > c$	$e > f$	$j > h$	$k > l$
$g > h$	$k > i$	$c > d$	$f > g$	$h > i$	

- The causal relations can be found when an order relation does not hold in the opposite direction. This is the case for all pairs except (h, j) and (i, k), and their opposites. We get:

$a \rightarrow b$	$j \rightarrow k$	$a \rightarrow c$	$d \rightarrow e$	$f \rightarrow g$	$h \rightarrow i$
$b \rightarrow g$	$i \rightarrow l$	$c \rightarrow d$	$e \rightarrow f$	$g \rightarrow j$	$k \rightarrow l$
$g \rightarrow h$					

- The potential parallelism relation holds true for $h||j$ as well as for $k||i$ (and the corresponding symmetric cases).
- The remaining relation of no-direct-succession can be found for all pairs that do not belong to $\rightarrow$ and $||$. It can be easily derived when we write down the relations in a matrix as shown in Figure 11.11. This matrix is also referred to as the *footprint matrix* of the log.

The algorithm

- It takes an event log L and its α relations.
- **Basic Rule**: whenever a task directly follows another, the two tasks should be connected in the process model.
    - If there is more than one task that can follow another, must determine whether the set of next tasks are partially exclusive or concurrent.
    - **Exception**: when two tasks are potentially parallel (||)

The Alpha-Algorithm (simplified)

1. Identify the set of all tasks in the log as T_L.
2. Identify the set of all tasks that have been observed as the first task in some case as T_I.
3. Identify the set of all tasks that have been observed as the last task in some case as T_O.

4. Identify the set of all connections to be potentially represented in the process model as a set X_L. Add the following elements to X_L:
   - a. Pattern (a): all pairs for which hold a→b.
   - b. Pattern (b): all triples for which hold a→(b#c).
   - c. Pattern (c): all triples for which hold (b#c)→d.

Note that triples for which Pattern (d) a→(b||c) or Pattern (e) (b||c)→d hold are not included in X_L.

	a	b	c	d	e	f	g	h	i	j	k	l
a	#	→	→	#	#	#	#	#	#	#	#	#
b	←	#	#	#	#	#	→	#	#	#	#	#
c	←	#	#	→	#	#	#	#	#	#	#	#
d	#	#	←	#	→	#	#	#	#	#	#	#
e	#	#	#	←	#	→	#	#	#	#	#	#
f	#	#	#	#	←	#	→	#	#	#	#	#
g	#	←	#	#	#	←	#	→	#	→	#	#
h	#	#	#	#	#	#	←	#	→	‖	#	#
i	#	#	#	#	#	#	#	←	#	#	‖	→
j	#	#	#	#	#	#	←	‖	#	#	→	#
k	#	#	#	#	#	#	#	#	‖	←	#	→
l	#	#	#	#	#	#	#	#	←	#	←	#

Fig. 11.11 Footprint represented as a matrix of the workflow log $L = [\langle a, b, g, h, j, k, i, l\rangle, \langle a, c, d, e, f, g, j, h, i, k, l\rangle]$

Limitations of the α-Algorithm

- Cannot distinguish **short loops** from true parallelism.

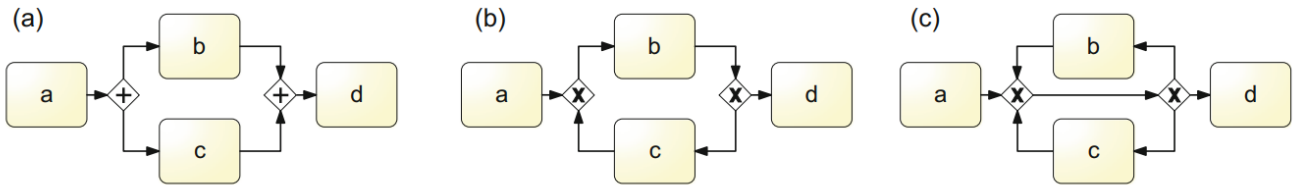
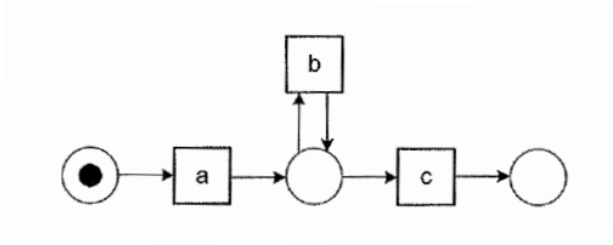


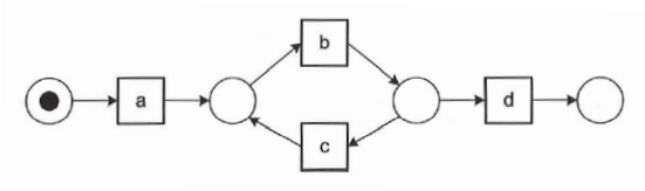
Fig. 11.13 Examples of two short loops, (**b**) and (**c**), that cannot be distinguished from model (**a**) by the α-algorithm

Short loops of length 1



b>b and not b>b implies b→b (impossible!)

Short loops of length 2



c>b and b>c implies c||b and b||c instead of c→b and b→c

- Sensitive to "noise": tasks that start and never end, those that end but never started will cause an inconsistent state for this algorithm or if an event is not recorded the algorithm will crumble (can't sustain errors of the system).
- Cannot deal with the incompleteness of an event log. Processes that have a lot of parallelism will cause this algorithm to fail.

Log Completeness

- Level of completeness required for a log
  - Assume for the execution sequence EF, there is a log missing
  - Then, the correct process model cannot be derived

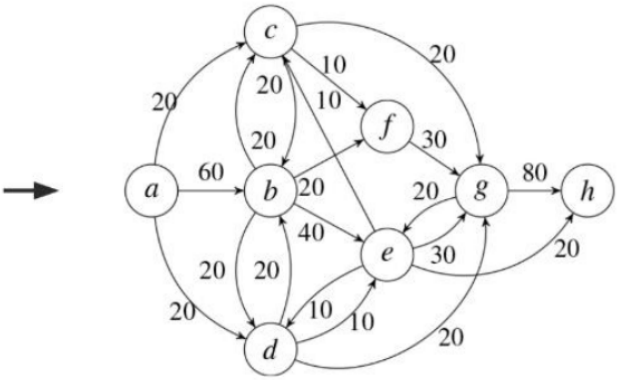
- Basic assumption: each execution sequence must be part of the log
  - Consequence: the complete behaviour is visible
  - Problem: amount of required instances grows dramatically
  - Example:
    - 10 activities are executed in parallel
    - Amount of potential execution sequences:
      10! = 3.628.800

Other Algorithms

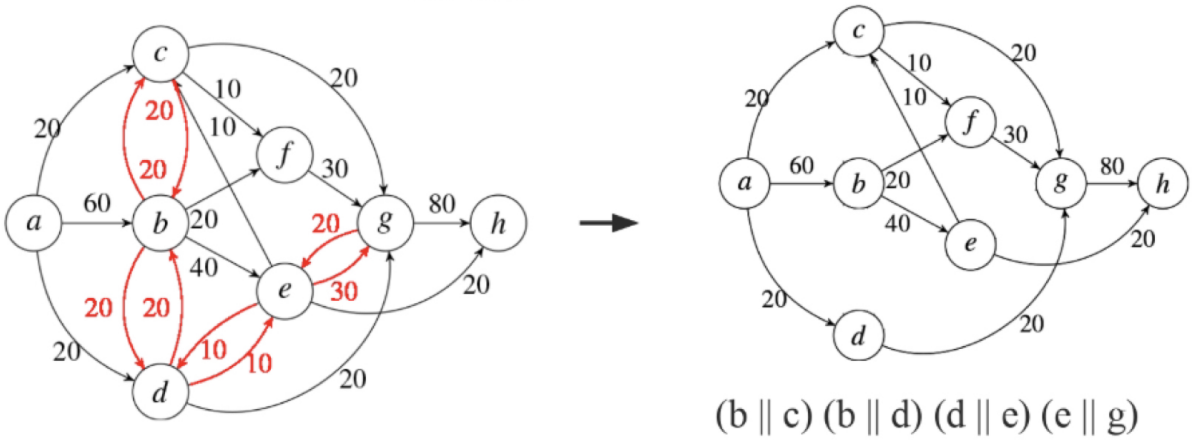
- **Alpha miner (α-miner)** - Simple, limited, not robust.
- **Heuristics miner (and derivatives, including Fodina)** - Robust to noise, fast, but can produce incorrect models.
- **Inductive miner** - Ensures that models are block-structured & correct.
- **Split miner** - Produces deadlock-free but not necessarily structured models.

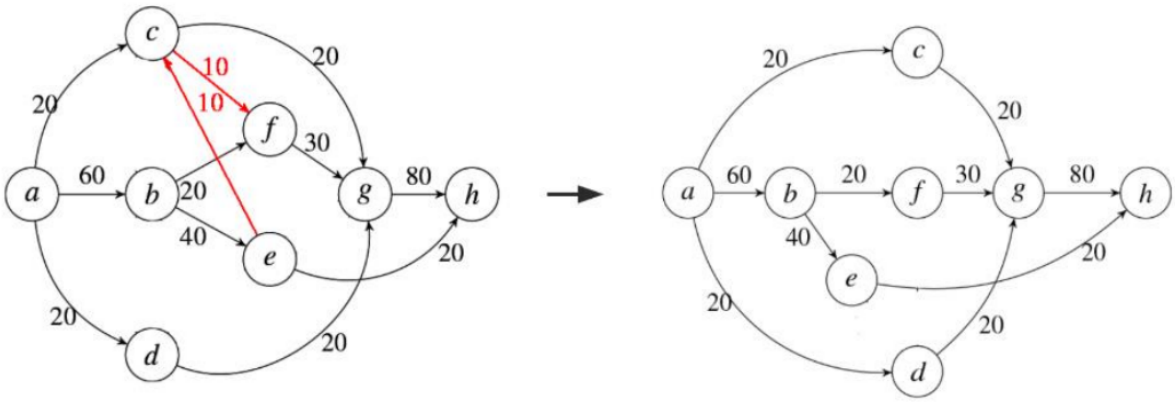
- Example of Split Miner

Trace	#obs
a » b » c » g » e » h	10
a » b » c » f » g » h	10
a » b » d » g » e » h	10
a » b » d » e » g » h	10
a » b » e » c » g » h	10
a » b » e » d » g » h	10
a » c » b » e » g » h	10
a » c » b » f » g » h	10
a » d » b » e » g » h	10
a » d » b » f » g » h	10

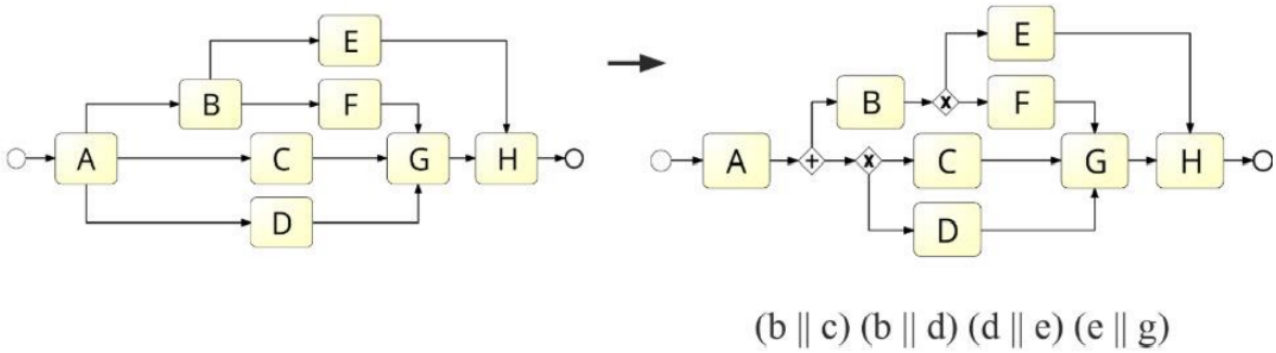


Remove all concurrent arcs

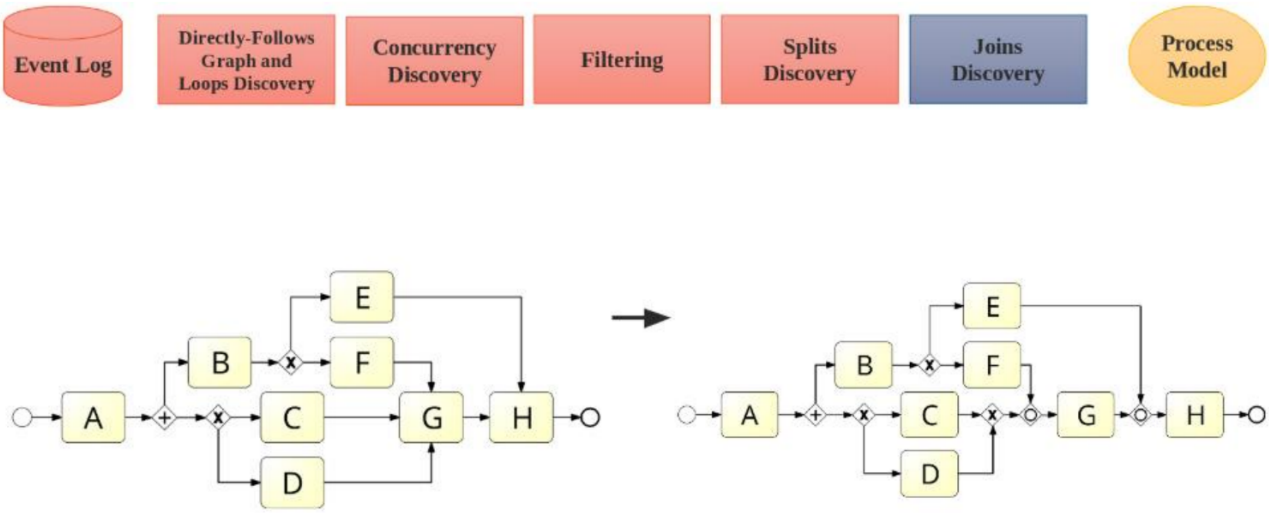




- If task a is directly followed by b and by c, but not by both (after b we do not observe c and vice versa), the heuristics miner will put an XOR-split between task a on the one hand, and tasks b and c on the other hand.
- If a is generally followed by both b and by c, the heuristics miner will put an AND-split after task a and before b and c.



- An XOR-join is placed between tasks b and c on the one hand, and task d on the other hand, if d is generally preceded by either task b or c but not both.
- An AND-join is placed there if d is generally preceded by both b and c.



Which algorithm to use?

- Inductive miner and split miner are reported to be the best, however this depends on a number of factors. **So, it depends!**

Factors to look for:

- **Fitness**: can the discovered process model replay the behavior contained in the log? A fitness of 1 means it can replay every trace in the log.
- **Precision**: does the discovered process model generate only the traces found in the log? A precision of 1 means any trace produced by the model is in the log also.

- **Generalization**: can the discovered process model capture traces that are not present in the log because of incompleteness but that are likely to be allowed?
- **Complexity**: how difficult is it to understand the model?

Process Performance Mining

- How to use event logs in order to assess the performance of a process along each dimension of the Devil's Quadrangle?

Time Dimension

- Measures like cycle time and waiting time are important performance measures.
- Event logs have timestamps that relate each event to the time of its occurrence that can be used for time analysis, therefore it is straightforward to plot events on the time axis.
- We can use classifiers to group events on a second axis like case ID or participant ID.
- We can do:
    - **Dotted charts**: simple and visual
    - **Timeline charts**: show the processing time (duration) and waiting times of a task. Usually done if we have both the start and end timestamp of each task. More informative than dotted because we can identify bottlenecks.

Cost Dimension

- Must take into account **direct** (like the four wheels for a car, determined case-by-case) and **indirect** (like machine or infrastructure depreciation) costs.
- We will use the **ABC method** (activity based costing). The problem of this algorithm is that we need to keep track of the task's duration and the start of tasks

Example 11.7 According to Figure 1.6 in Chapter 1 (see page 19), the rental process of BuildIT contains five major tasks. We observe the following durations in the event logs for the case of a bulldozer rental requested on 21st August:

- "Submit equipment rental request" is conducted by the site engineer. It takes the engineer 20 min to fill in the form on paper. The production of each paper form costs € 1. The site engineer gets an annual salary of € 60,000.
- The clerk receives the form, selects a suitable piece of equipment, and checks its availability. These tasks take altogether 15 min. The clerk has an annual salary of € 40,000.
- The works engineer reviews the rental request (annual salary of € 50,000). This review takes 10 mins.
- The clerk is also responsible for sending a confirmation including a purchase order for renting the equipment, which takes 30 min.

In order to work with these numbers we have to make some assumptions. First, at BuildIT the actual working year contains 250 working days of 8 h. Furthermore, all employees receive health insurance and pension contributions of 20% on top of their salary. Finally, people take on average 10 days of sick leave per year. Given the above, the labor cost of each participant per minute is $\frac{salary \times 120\%}{(250-10)\times 8\times 60}$. This is for the site engineer € 0.63 per minute, for the clerk € 0.42 per minute, and for the works engineer € 0.52 per minute. Altogether, this case created costs of 20 min × € 0.63 per minute + (15+30) min × € 0.42 per minute + 10 min × € 0.52 per minute, which sums up to € 36.70.

Now consider a case of a street construction process. We observe from the event log the following durations (processing times) for these two tasks:

- "Prepare the foundation" is conducted by four workers. It took one week at a specific construction case. The used excavator costs € 100,000. It is written off in 5 years and has annual maintenance costs of € 5,000. A worker gets an annual salary of € 35,000.
- "Tar the road" is conducted by six workers. It took 2 days in this case. The tarring machine costs € 200,000, is also written off in 5 years and costs € 10,000 in annual maintenance.

For this case, we can also take the costs of the machinery into account. The labor cost per day is € 175 for one worker. The excavator costs € 20,000 + 5,000 per annum for write-off and maintenance, which is € 104.17 per working day. For the preparation of the foundation, this amounts to 4 × 5 × € 175 + 5 × € 104.17 = € 4,020.85. For the taring of the road, the costs are 6 × 2 × € 175 + 2 × € 208.34 = € 2,516.68. □

- Check if there are repetitions in the execution logs because they usually occur if a task has not been completed successfully.
- We can find the probability of repetitions like this:

$$r = 1 - \frac{T}{CT}.$$

Both CT and T can now be determined using the data of the event logs. Consider the five cases in which we observe the following execution times for task a:

1. 5 min, 10 min;
2. 10 min;
3. 20 min, 6 min, 10 min;
4. 5 min;
5. 10 min, 10 min.

The cycle time CT of a can now be calculated as the average execution time of a per case, while the average execution time T is the average execution time of a per instantiation. Both can be determined based on the sum of all executions of a, which is 86 min here. We have five cases, such that $CT = 86/5 = 17.2$. Altogether, a is executed nine times yielding $T = 86/9 = 9.56$. Hence, we get $r = 1 - \frac{86/9}{86/5} = 1 - \frac{5}{9} = 0.44$. Of course, this calculation is only an approximation of the real value for r. It builds on the assumption that the duration of a task always follows the same distribution, no matter if it is the first, the second or another iteration.

- **Ticketing systems**: record which resource is working on a case, therefore it is easier to track repetitions.

Flexibility Dimension (degree of variation a process allows)

- The event logs the process produces can reveal info about flexibility, it might turn out the process is more flexible than what we want.
- Consider the process for renting equipment at BuildIT. The process requires an equipment rental request form to be sent by email. Some engineers however prefer to call the depot directly instead of filling the form. Since these engineers are highly distinguished, it is not easy for the clerk to turn down these calls. As a result, the clerk fills out the form while being on the phone. Not only does this procedure take more time, but, due to noise at the construction site, it also increases the likelihood of errors. This means that the rental process has two options to submit a request: by form (the standard procedure) and via the phone.

Accordingly, the number of *distinct executions* DE can be defined based on a workflow log L as

$$DE = |L|.$$

Conformance Checking

- Is concerned with the question whether or not the execution of a process follows predefined rules or constraints. To do this, inspect the event log. If the constraints do not hold we call it a **violation**.

Conformance of Control Flow

- Conformance can be analyzed in two ways:
  - **Based on constraints**
  - **Based on a normative process model**
- **Control Flow Constraints**:
  - **Mandatoriness**: making sure the task is executed.
  - **Exclusiveness**: two tasks cannot co-occur in the same case.
  - **Ordering**: two tasks must always occur in a given order.

Conformance of Data and Resources

- Search for cases in the log where a given data field takes a forbidden value.
- Participants usually require permissions because of their roles.

Process Mining in Practice

- Two approaches to apply process mining in practice:
  - **Explanatory Approach**: driven by curiosity with not particular question in mind. First, discover a process model or conformance check an existing model. Then, analyze the bottlenecks and longitudinal or cross-sectional analysis.
  - **Question-Driven Approach**: driven by a business problem. First, identify the scope of the problem, formulate question and apply process mining to answer them.

    In a question-driven approach, there are typically five phases involved: (1) framing the problem; (2) collecting the data; (3) analyzing the data; (4) interpreting the results; and (5) formulating a process improvement proposal.

    In the first phase of the question-driven approach, we need to formulate a top-level problem or question for the process mining project. For example, why does the process perform poorly, for example, in terms of cycle time? Or why do we have frequent defects (high error rates)?

    A time-consuming step within this phase is that of data pre-processing. Once we have the event log, we need to clean the data by filtering irrelevant events, cover gaps that may exist due to recording errors, and combine equivalent events from different tables if the data is scattered across tables (potentially originating from different systems, e.g., a CRM and an ERP system)—recall that we are interested in tracing the end-to-end business process. Often, for example, the control-flow data (events and their timestamps) are available in one table, while the resource data is in another table, and the business objects (containing the data attributes) are in another

The second phase of the question-driven approach is *data collection*. Here, we need to find relevant data sources, which are typically the databases of the enterprise systems over which the process is executed. As part of this extraction, we need to identify the process-related entities and their identifiers (in particular the case identifier) so as to be able to group different events into traces, which is necessary in order to produce an event log in the XES format. In the Suncorp case, the data was extracted from different tables of a claims handling system.

The third phase is that of *analysis*. In the Suncorp case, a data analyst took the log consisting of traces of low-value claims and divided it into two sub-logs: fast simple claims and slow simple claims, as defined above. The analyst applied a manual variants analysis technique. Specifically, the log was split into two by applying filtering operations (less than 5 days, more than 5 days) and two dependency graphs were discovered and visually compared. After some effort, it became apparent that there were some relevant differences, specifically two types of unwanted repetitions were occurring frequently in the slow cases, but rarely in the fast cases. These results were backed up by a statistical analysis of the repetitions.

The fourth phase is dedicated to extracting insights from the analysis results and to validating these insights with the domain experts and other stakeholders. In the Suncorp case study, the analysts discussed the findings with several process participants and other stakeholders knowledgeable of the process. They took concrete instances where unwanted repetitions occurred, and showed them to these stakeholders. Based on these focused discussions, they managed to determine the key reasons for the unwanted repetitions.

These two approaches are **complementary**. Oftentimes we start with an exploratory approach, and as we get a clearer picture of the process and its bottlenecks, we continue the analysis with concrete business-driven questions in mind.