Process Mining

Gabriel Rovesti

April 20, 2025

Contents

Chapter 1

Introduction to Process Mining

1.1 What is Process Mining?

Process Mining is a research discipline that sits at the intersection between process science and data science. It provides techniques to discover, monitor, and improve real processes by extracting knowledge from event logs readily available in today's information systems. Process mining aims to bridge the gap between traditional model-based process analysis (e.g., simulation and other business process management techniques) and data-centric analysis techniques such as machine learning and data mining.

The main idea of process mining is to extract knowledge from event logs recorded by an information system, to discover, monitor, and improve real processes (i.e., not assumed processes). With the increasing amount of data available in information systems, logs containing detailed information about past process executions have become widely available. Process mining techniques use these logs to uncover what really happened during the execution of a process, to detect bottlenecks, to anticipate problems, to record policy violations, to recommend countermeasures, and to streamline processes.

1.1.1 The Three Main Types of Process Mining

There are three main types of process mining:

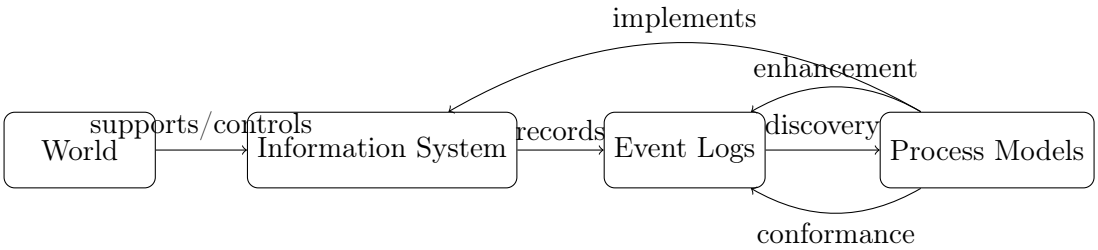


Figure 1.1: Process Mining overview

1. **Process Discovery**: Taking an event log and producing a model without using any a priori information. This is the most prominent process mining technique. For many organizations, it is surprising to see that existing techniques are able to discover real processes merely based on example executions in event logs.

2. **Conformance Checking**: An existing process model is compared with an event log of the same process. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa.

3. **Process Enhancement**: The idea is to extend or improve an existing process model using information about the actual process recorded in some event log. Whereas conformance checking measures the alignment between model and reality, this third type of process mining aims at changing or extending the a priori model.

1.1.2 Event Logs

Process mining starts with event logs. An event log consists of cases (i.e., process instances). Each case consists of events such that each event relates to precisely one case. Events within a case are ordered, and can have attributes like:

- **Case ID**: Identifier of the case the event belongs to
- **Activity**: The type of activity executed
- **Timestamp**: When the event occurred
- **Resource**: Who executed the activity
- **Other attributes**: Cost, location, etc.

Here's an example of a simple event log:

Case ID	Activity	Timestamp	Resource	Cost
1	Register Request	2023-01-01 10:00	John	50
1	Examine Thoroughly	2023-01-01 10:15	Mike	400
2	Register Request	2023-01-01 10:05	John	50
2	Examine Casually	2023-01-01 10:10	Sue	200
1	Check Ticket	2023-01-01 10:30	Pete	100
2	Check Ticket	2023-01-01 10:20	Mike	100

Table 1.1: Example of an event log

For process mining techniques to be applicable, events need to be related to cases and ordered. The minimal information needed for process mining is the case ID, activity, and some form of ordering (often timestamps).

1.2 The Process Mining Challenges

Process mining is an emerging discipline with many challenges:

1. **Finding, merging, and cleaning event data**: In real-life situations, event data may be distributed over various sources and may require substantial efforts to collect and prepare data suitable for process mining.
2. **Dealing with complex event logs having diverse characteristics**: Event logs may have millions of events and exhibit a high degree of variability making it difficult to extract a simple and representative process model.
3. **Creating representative benchmarks**: To compare the different approaches and tools for process mining, good benchmarks based on real-life data are needed.
4. **Dealing with concept drift**: Processes may change while being analyzed, thus complicating analysis and interpretation of results.
5. **Balancing between quality criteria**: There are four competing quality criteria for process mining: fitness, simplicity, precision, and generalization.
6. **Cross-organizational mining**: There is a need to compare processes within the same organization or similar processes in different organizations.
7. **Providing operational support**: Process mining is not limited to offline analysis and can also be used for online operational support.
8. **Combining process mining with other types of analysis**: The challenge is to combine automated process mining techniques with other approaches such as visual analytics.

1.2.1 Data, Process, and Science

Process mining connects data science with process science:

- **Data Science** involves data extraction, data cleaning, data integration, data analysis, etc.

- **Process Science** focuses on understanding, designing, controlling, and improving operational processes.
- **Process Mining** bridges the gap between the two.

1.3 The Importance of Process Mining

There are several reasons why process mining is becoming increasingly important:

- **More event data is becoming available**: Organizations record an unprecedented amount of events.
- **Processes are complex and need to be streamlined**: Operational processes need to be managed well to provide good services while reducing costs.
- **Compliance checking is required**: Organizations need to show that they satisfy both internal and external requirements.
- **Process improvement is needed**: Process mining can help identify bottlenecks and suggest potential improvements.

Chapter 2

Process Modeling with Petri Nets

2.1 Introduction to Petri Nets

Petri nets are a formal modeling language for the description of distributed systems. They were introduced by Carl Adam Petri in his doctoral dissertation in 1962. Petri nets are particularly well-suited for modeling concurrent and asynchronous systems.

2.1.1 Why Study Petri Nets?

For process mining and business process management, Petri nets offer several advantages:

- **Formal semantics**: The behavior of Petri nets is mathematically defined.
- **Graphical notation**: Petri nets have a graphical representation that is easy to understand.
- **Expressiveness**: Petri nets can express a wide range of process patterns.
- **Analysis techniques**: There are many techniques to analyze Petri nets for properties like deadlock freedom, boundedness, etc.
- **Vendor independence**: Petri nets are not tied to any vendor-specific notation.

While many business process modeling languages exist (e.g., BPMN, EPC, UML activity diagrams), they tend to come and go. Many lack formal semantics, making it difficult to analyze processes rigorously. Petri nets, on

the other hand, provide a foundation for understanding the core semantics of most process modeling languages.

2.1.2 Basic Concepts of Petri Nets

A Petri net is a directed bipartite graph with two types of nodes:

Definition 2.1 (Petri Net) *A Petri net is a tuple $PN = (P, T, F)$ where:*

- *P is a finite set of places,*
- *T is a finite set of transitions, with $P \cap T = \emptyset$,*
- *$F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation).*

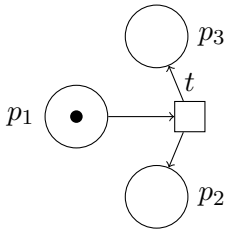


Figure 2.1: A simple Petri net

- **Places** (represented by circles) can contain tokens. Places represent conditions or states.
- **Transitions** (represented by rectangles) represent events or actions that may occur.
- **Arcs** connect places to transitions and transitions to places. They represent flow relations.
- **Tokens** (represented by black dots inside places) represent objects or resources. The state of a Petri net is determined by the distribution of tokens over places, called a marking.

2.1.3 Petri Net Dynamics: The Token Game

The dynamics of a Petri net are governed by the "token game":

Definition 2.2 (Marking) *The state of a Petri net is represented by a marking $M : P \to \mathbb{N}$, which assigns to each place a non-negative integer (the number of tokens in that place).*

Definition 2.3 (Enabled Transition) *A transition $t \in T$ is enabled at marking M if for all places $p \in {}^\bullet t$ (input places of t), $M(p) > 0$.*

Definition 2.4 (Firing Rule) *When an enabled transition t fires:*

1. *For each input place $p \in \bullet t$, one token is removed.*
2. *For each output place $p \in t\bullet$ (output places of t), one token is added.*

The firing of t at marking M leads to a new marking M', denoted as $M \xrightarrow{t} M'$.

Example 2.1 *Consider the Petri net in Figure 2.1. Initially, there is one token in place p_1, so transition t is enabled. When t fires, the token from p_1 is consumed, and one token is produced in each of p_2 and p_3.*

2.1.4 Important Petri Net Concepts

Definition 2.5 (Preset and Postset) *For a node $x \in P \cup T$:*

- *The preset $\bullet x = \{y \in P \cup T | (y, x) \in F\}$ is the set of input nodes.*
- *The postset $x\bullet = \{y \in P \cup T | (x, y) \in F\}$ is the set of output nodes.*

Definition 2.6 (Reachability) *A marking M' is reachable from marking M if there exists a sequence of transitions $\sigma = t_1 t_2 \dots t_n$ such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} M'$. We write $M \xrightarrow{\sigma} M'$.*

Definition 2.7 (Reachability Graph) *The reachability graph of a Petri net with initial marking M_0 is a labeled directed graph where:*

- *Nodes represent reachable markings,*
- *Edges represent transitions between markings,*
- *The initial node corresponds to M_0.*

2.1.5 Arc Weights in Petri Nets

In standard Petri nets, each arc has a weight of 1. However, we can extend Petri nets with arc weights:

Definition 2.8 (Arc Weight) *An arc weight function $W : F \to \mathbb{N}^+$ assigns a positive integer to each arc.*

With arc weights, the firing rule changes: when a transition t fires, it removes $W(p, t)$ tokens from each input place p and adds $W(t, p')$ tokens to each output place p'.

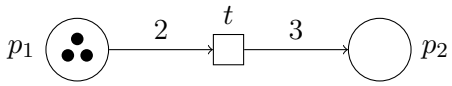


Figure 2.2: Petri net with arc weights

In Figure 2.2, transition t is enabled because place p_1 has at least 2 tokens. When t fires, 2 tokens are removed from p_1 and 3 tokens are added to p_2.

2.2 Workflow Nets

Workflow nets are a special class of Petri nets tailored for modeling business processes:

Definition 2.9 (Workflow Net) *A Petri net $PN = (P, T, F)$ is a workflow net (WF-net) if:*

1. *It has a single source place $i \in P$ such that $\bullet i = \emptyset$.*
2. *It has a single sink place $o \in P$ such that $o\bullet = \emptyset$.*
3. *Every node $x \in P \cup T$ is on a path from i to o.*

Workflow nets model processes that start in an initial state i and aim to reach a final state o. The third condition ensures that there are no "dangling" nodes that cannot contribute to the completion of a process instance.

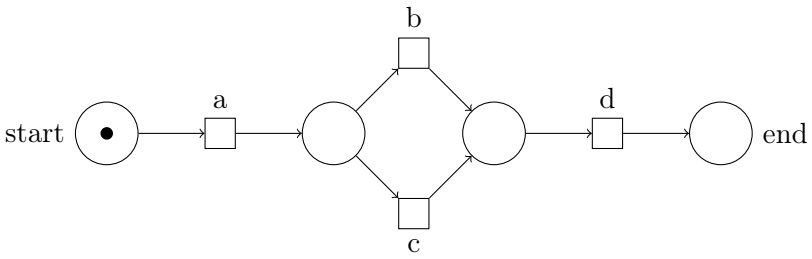


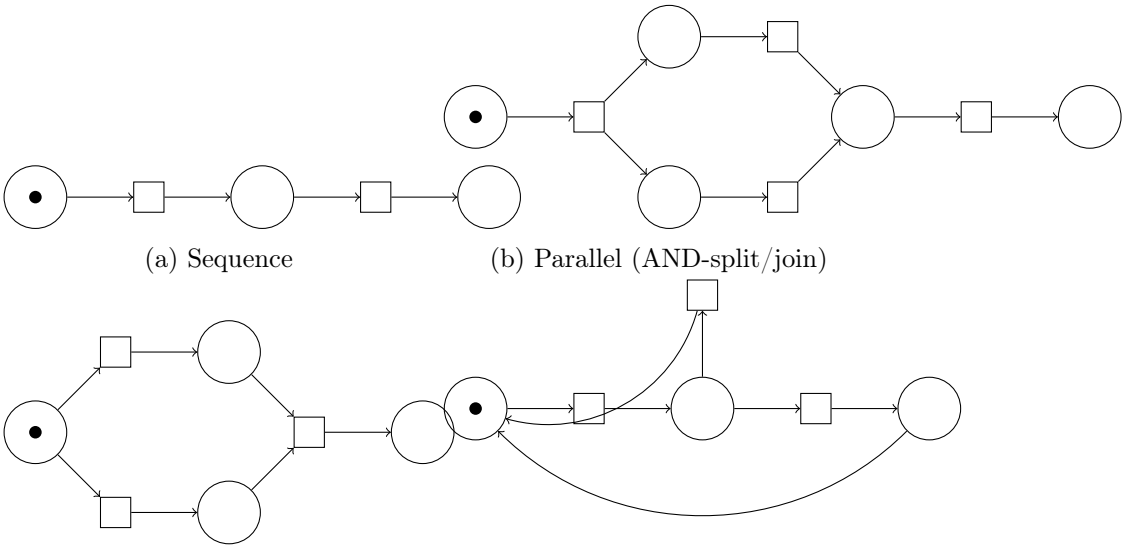
Figure 2.3: A workflow net

2.2.1 Control-Flow Patterns in Workflow Nets

Workflow nets can represent common control-flow patterns:

1. **Sequence**: Activities that follow each other.
2. **Parallel split (AND-split)**: A point where a single thread of control splits into multiple threads that can execute concurrently.

3. **Synchronization (AND-join)**: A point where multiple parallel threads converge into a single thread.
4. **Exclusive choice (XOR-split)**: A point where one of several branches is chosen based on a decision.
5. **Simple merge (XOR-join)**: A point where two or more alternative branches come together without synchronization.
6. **Multi-choice (OR-split)**: A point where one or more branches are chosen from many alternatives.
7. **Synchronizing merge (OR-join)**: A point where multiple threads that may execute converge, with synchronization taking place for active threads.
8. **Iteration**: Activities that may be repeated.



(a) Sequence (b) Parallel (AND-split/join)

(c) Exclusive choice (XOR-split/join) (d) Iteration

Figure 2.4: Control-flow patterns in workflow nets

2.3 Modeling Business Processes with Petri Nets

When modeling business processes with Petri nets, the following guidelines are useful:

1. Activities correspond to transitions.
2. States correspond to places.

3. A case (process instance) corresponds to a token moving through the net.
4. Use workflow nets to ensure a clear start and end.
5. Use patterns to model typical control-flow constructs.

Example 2.2 (Modeling a Loan Application Process) *Consider a loan application process:*

1. *The process starts when a customer submits a loan application.*
2. *The application is registered in the system.*
3. *Two activities are performed in parallel: checking the credit history and verifying the income.*
4. *Based on the results, the application is either approved or rejected.*
5. *If approved, the loan is disbursed.*
6. *The process ends.*

Figure 2.5 shows a Petri net model of this process.

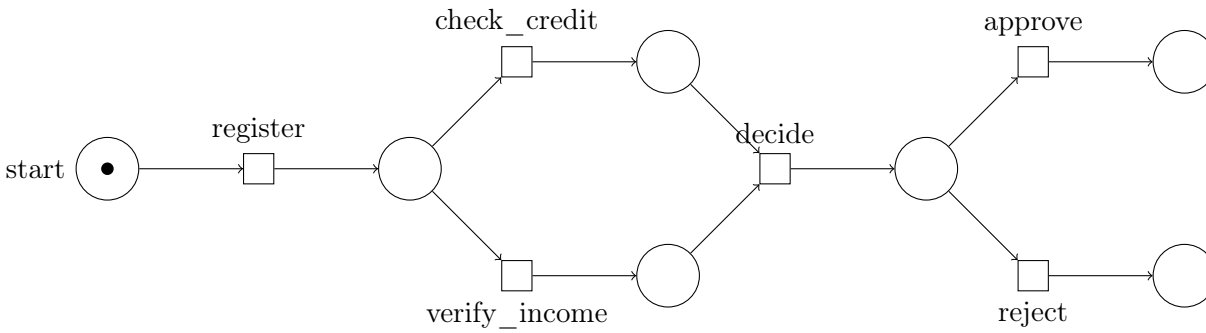


Figure 2.5: Petri net model of a loan application process

Chapter 3

Soundness Analysis of Process Models

A key property of business process models is "soundness", which ensures that a process can always complete and does not have design flaws like deadlocks or unreachable activities.

3.1 Soundness of Workflow Nets

Definition 3.1 (Soundness) *A workflow net $PN = (P, T, F)$ with an initial place i and a final place o is sound if:*

1. ***Option to complete**: For every marking M reachable from the initial marking $[i]$, there exists a firing sequence leading from M to the final marking $[o]$.*
2. ***Proper completion**: If marking M is reachable from the initial marking $[i]$ and $M \geq [o]$ (i.e., M marks the final place), then $M = [o]$.*
3. ***No dead transitions**: For every transition $t \in T$, there exists a marking M reachable from the initial marking $[i]$ such that t is enabled at M.*

Intuitively, soundness means:

- From any reachable state, it's possible to reach the end state.
- Once the end state is reached, all other places are empty.
- Every activity can be executed in at least one process instance.

3.2 Analysis of Petri Net Properties

To determine whether a workflow net is sound, we need to analyze several properties:

3.2.1 Boundedness

Definition 3.2 (Boundedness) *A place p in a Petri net is k-bounded if there is no reachable marking with more than k tokens in p. A Petri net is k-bounded if all of its places are k-bounded. A Petri net is bounded if it is k-bounded for some k. A 1-bounded Petri net is called safe.*

Boundedness is essential for soundness as it ensures that the number of tokens in each place remains finite, preventing unlimited growth of tokens that could indicate process errors like infinite loops.

3.2.2 Liveness

Definition 3.3 (Liveness) *A transition t in a Petri net is live if for every reachable marking M, there exists a marking M' reachable from M such that t is enabled at M'. A Petri net is live if all of its transitions are live.*

Liveness ensures that any transition can eventually be fired, meaning that no part of the process becomes "stuck" or permanently disabled.

3.2.3 Deadlock Freedom

Definition 3.4 (Deadlock) *A marking M is a deadlock if no transition is enabled at M. A Petri net is deadlock-free if no reachable marking is a deadlock, except possibly the final marking in the case of a workflow net.*

Deadlock freedom guarantees that the process cannot reach a state where no activity can be executed, except when the process is complete.

3.2.4 Home Markings

Definition 3.5 (Home Marking) *A marking M is a home marking if from any reachable marking, it is possible to reach M. A Petri net is reversible if the initial marking is a home marking.*

In workflow nets, the final marking $[o]$ should be a home marking, ensuring that from any state, the process can always be completed.

3.2.5 Dead Transitions

Definition 3.6 (Dead Transition) *A transition t is dead if there is no reachable marking that enables t.*

Dead transitions represent activities that can never be executed, which might indicate design flaws.

3.3 Verification Techniques

Various techniques can be used to verify the properties of Petri nets:

3.3.1 Reachability Graph Analysis

The reachability graph represents all possible states (markings) of a Petri net and the transitions between them. For bounded Petri nets, the reachability graph is finite and can be constructed algorithmically.

Algorithm 1 Reachability Graph Construction

1: Start with the initial marking M_0
2: $V \leftarrow \{M_0\}$ (set of visited markings)
3: $E \leftarrow \emptyset$ (set of edges)
4: $U \leftarrow \{M_0\}$ (set of unexplored markings)
5: **while** $U \neq \emptyset$ **do**
6: Select and remove a marking M from U
7: **for** each transition t enabled at M **do**
8: Compute the marking M' reached by firing t: $M \xrightarrow{t} M'$
9: **if** $M' \notin V$ **then**
10: $V \leftarrow V \cup \{M'\}$
11: $U \leftarrow U \cup \{M'\}$
12: **end if**
13: $E \leftarrow E \cup \{(M, t, M')\}$
14: **end for**
15: **end while**
16: **return** (V, E)

Once the reachability graph is constructed, we can check various properties:

- **Boundedness**: The Petri net is bounded if and only if the reachability graph is finite.
- **Liveness**: A transition t is live if and only if for every node M in the reachability graph, there exists a path from M to some marking M' where t is enabled.
- **Deadlock freedom**: There is no deadlock if and only if every node in the reachability graph has at least one outgoing edge (except possibly the final marking).
- **Home marking**: A marking M is a home marking if and only if it is reachable from every node in the reachability graph.
- **Dead transitions**: A transition t is dead if and only if it doesn't appear on any edge in the reachability graph.

3.3.2 State Space Explosion and Coverability Graphs

For unbounded Petri nets, the reachability graph is infinite. To analyze such nets, we use coverability graphs, which finitely represent the potentially infinite state space.

In a coverability graph, we use the special symbol ω to represent an unbounded number of tokens in a place.

Algorithm 2 Coverability Graph Construction

Start with the initial marking M_0
$V \leftarrow \{M_0\}$ (set of visited markings)
$E \leftarrow \emptyset$ (set of edges)
$U \leftarrow \{M_0\}$ (set of unexplored markings)
while $U \neq \emptyset$ **do**
    Select and remove a marking M from U
    for each transition t enabled at M **do**
        Compute the marking M' reached by firing t: $M \xrightarrow{t} M'$
        Check if there is a marking M'' on the path from M_0 to M such that $M'' < M'$ (i.e., $M''(p) \leq M'(p)$ for all p and $M'' \neq M'$)
        if such an M'' exists **then**
            for each place p with $M''(p) < M'(p)$ **do**
                $M'(p) \leftarrow \omega$ (mark the place as unbounded)
            end for
        end if
        if $M' \notin V$ **then**
            $V \leftarrow V \cup \{M'\}$
            $U \leftarrow U \cup \{M'\}$
        end if
        $E \leftarrow E \cup \{(M, t, M')\}$
    end for
end while
return (V, E)

The coverability graph allows us to check if a Petri net is bounded: the net is unbounded if and only if the coverability graph contains a marking with at least one ω symbol.

3.3.3 Structural Analysis

Some properties can be determined from the structure of the Petri net without exploring its state space:

Theorem 3.1 *Let $PN = (P, T, F)$ be a workflow net with initial place i and final place o. Let PN' be the "short-circuited" net obtained by adding*

a transition t^* connecting o back to i. Then PN is sound if and only if $(PN', [i])$ is live and bounded.

This theorem provides a way to check soundness using standard Petri net analysis techniques.

3.3.4 Tools for Petri Net Analysis

Several tools are available for analyzing Petri nets:

- **WoPeD** (Workflow Petri Net Designer): A tool for modeling, simulating, and analyzing workflow nets.
- **ProM**: A process mining framework with plugins for Petri net analysis.
- **CPN Tools**: A tool for editing, simulating, and analyzing Colored Petri Nets.
- **PIPE** (Platform Independent Petri Net Editor): A tool for creating and analyzing Petri nets.

Example 3.1 (Analyzing Soundness) *Consider the workflow net in Figure 3.1. Is this workflow net sound?*

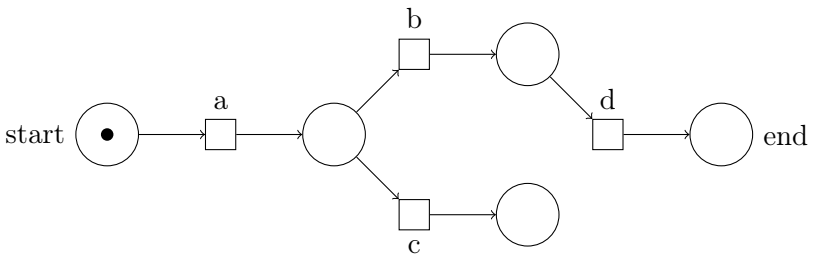


Figure 3.1: An unsound workflow net

By constructing the reachability graph, we can see that after firing transitions a and c, we reach a marking [p3] from which the final marking [end] is not reachable. This violates the "option to complete" property, so the workflow net is not sound.

The problem is that the token in place p3 gets stuck, as there is no transition that consumes from p3.

Chapter 4

Process Discovery

Process discovery techniques aim to automatically construct a process model from event logs, without the need for prior knowledge about the process. The goal is to discover a model that best represents the behavior observed in the event log.

4.1 The Process Discovery Challenge

Process discovery is a challenging task for several reasons:

- **Noise**: Event logs may contain outliers or exceptional behavior that should not be included in the model.
- **Incompleteness**: Event logs typically contain only a fraction of all possible process executions.
- **Balance of quality criteria**: A discovered model should strike a balance between four quality criteria: fitness, precision, generalization, and simplicity.

4.1.1 Quality Criteria for Process Discovery

Definition 4.1 (Fitness) *Fitness measures how well the discovered model can reproduce the behavior observed in the event log. A model with perfect fitness can replay all traces in the log from start to end.*

Definition 4.2 (Precision) *Precision measures how well the model prohibits behavior that is not observed in the event log. A model with perfect precision does not allow for more behavior than what is recorded in the log.*

Definition 4.3 (Generalization) *Generalization measures how well the model generalizes the behavior observed in the event log. A model with good generalization can reproduce behavior that is similar to but not exactly the same as what is recorded in the log.*

Definition 4.4 (Simplicity) *Simplicity measures the complexity of the model. A simple model is easier to understand and is less likely to overfit the data.*

These criteria often conflict with each other. For example, a model that perfectly fits the event log may have poor precision if it allows for much more behavior than observed. Similarly, a model with perfect precision may overfit the data and have poor generalization.

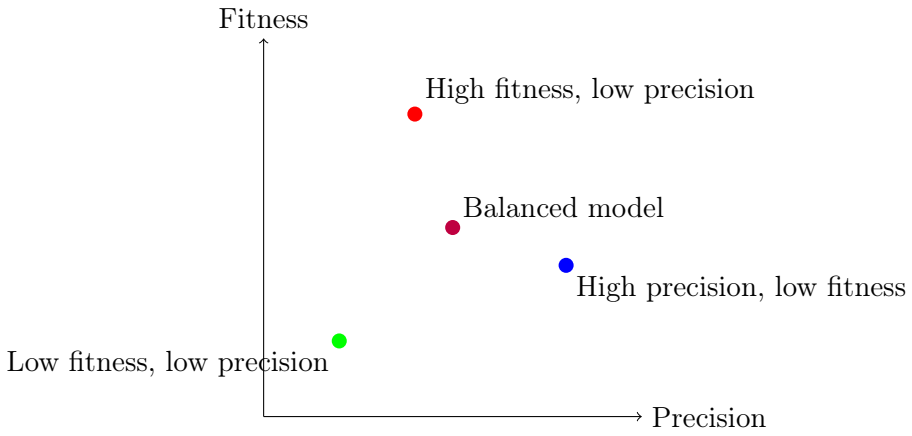


Figure 4.1: Trade-off between fitness and precision

4.2 The Alpha Algorithm

The Alpha algorithm is one of the first and most well-known process discovery algorithms. It constructs a Petri net model from an event log by analyzing the causal relationships between activities.

4.2.1 Basic Idea

The Alpha algorithm analyzes the directly-follows relation between activities in the event log and uses it to identify causal dependencies, parallelism, and choice.

Definition 4.5 (Directly-Follows Relation) *Let L be an event log. The directly-follows relation $>_L$ is defined as: $a >_L b$ if and only if there is a trace $\sigma = \langle e_1, e_2, \ldots, e_n \rangle$ in L such that $e_i = a$ and $e_{i+1} = b$ for some i.*

From the directly-follows relation, the Alpha algorithm derives four relations:

Definition 4.6 (Relations for the Alpha Algorithm) *Let L be an event log with activities A.*

- ***Causality***: $a \rightarrow_L b$ *if* $a >_L b$ *and not* $b >_L a$.
- ***Parallelism***: $a \parallel_L b$ *if* $a >_L b$ *and* $b >_L a$.
- ***Choice***: $a \#_L b$ *if not* $a >_L b$ *and not* $b >_L a$.

These relations are used to identify the basic patterns in the process:

- **Sequence**: If $a \rightarrow_L b$, then a is followed by b.
- **Parallel split (AND-split)**: If $a \rightarrow_L b$, $a \rightarrow_L c$, and $b \parallel_L c$, then b and c can be executed in parallel after a.
- **Synchronization (AND-join)**: If $a \rightarrow_L c$, $b \rightarrow_L c$, and $a \parallel_L b$, then c can only be executed after both a and b.
- **Exclusive choice (XOR-split)**: If $a \rightarrow_L b$, $a \rightarrow_L c$, and $b \#_L c$, then after a either b or c is executed.
- **Simple merge (XOR-join)**: If $a \rightarrow_L c$, $b \rightarrow_L c$, and $a \#_L b$, then c is executed after either a or b.

4.2.2 The Alpha Algorithm

The Alpha algorithm consists of the following steps:

Algorithm 3 Alpha Algorithm

1: **Input:** Event log L
2: **Output:** Workflow net $\alpha(L) = (P, T, F)$
3: $T_L \leftarrow \{a \in A \mid \exists \sigma \in L : a \in \sigma\}$ (all activities in L)
4: $T_I \leftarrow \{a \in A \mid \exists \sigma \in L : \sigma[1] = a\}$ (start activities)
5: $T_O \leftarrow \{a \in A \mid \exists \sigma \in L : \sigma[|\sigma|] = a\}$ (end activities)
6: Compute the relations $>_L$, $\rightarrow_L$, $\parallel_L$, and $\#_L$
7: $X_L \leftarrow \{(A, B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge \forall a \in A, b \in B : a \rightarrow_L b \wedge \forall a_1, a_2 \in A : a_1 \#_L a_2 \wedge \forall b_1, b_2 \in B : b_1 \#_L b_2\}$
8: $Y_L \leftarrow \{(A, B) \in X_L \mid \forall (A', B') \in X_L : A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B')\}$ (maximal pairs in X_L)
9: $P_L \leftarrow \{p_{(A,B)} \mid (A, B) \in Y_L\} \cup \{i, o\}$ (places)
10: $F_L \leftarrow \{(a, p_{(A,B)}) \mid (A, B) \in Y_L \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_L \wedge b \in B\} \cup \{(i, t) \mid t \in T_I\} \cup \{(t, o) \mid t \in T_O\}$ (flow relation)
11: **return** (P_L, T_L, F_L)

The key idea is to identify pairs (A, B) where:

- A is a set of input activities and B is a set of output activities,
- Every activity in A has a causal relation with every activity in B,

- Activities within A are in a choice relation with each other,
- Activities within B are in a choice relation with each other.

Each such pair (A, B) corresponds to a place in the discovered Petri net.

Example 4.1 (Alpha Algorithm) *Consider the event log* $L = [\langle a, b, c, d\rangle^{10}, \langle a, c, b, d\rangle^{5}, \langle a, e, d\rangle$
Step 1: $T_L = \{a, b, c, d, e\}$
Step 2: $T_I = \{a\}$
Step 3: $T_O = \{d\}$
Step 4: *Compute the relations:*

- $>_L$: $a >_L b$, $a >_L c$, $a >_L e$, $b >_L c$, $b >_L d$, $c >_L b$, $c >_L d$, $e >_L d$
- $\rightarrow_L$: $a \rightarrow_L b$, $a \rightarrow_L c$, $a \rightarrow_L e$, $b \rightarrow_L d$, $c \rightarrow_L d$, $e \rightarrow_L d$
- $\|_L$: $b \|_L c$, $c \|_L b$
- $\#_L$: $a\#_L d$, $b\#_L e$, $c\#_L e$, $d\#_L a$, $d\#_L b$, $d\#_L c$, $d\#_L e$, $e\#_L b$, $e\#_L c$

Step 5: $X_L = \{(\{a\}, \{b, c, e\}), (\{b, c\}, \{d\}), (\{e\}, \{d\})\}$
Step 6: $Y_L = X_L$ *(all pairs in* X_L *are maximal)*
Step 7: $P_L = \{p_{(\{a\},\{b,c,e\})}, p_{(\{b,c\},\{d\})}, p_{(\{e\},\{d\})}, i, o\}$
Step 8: $F_L = \{(a, p_{(\{a\},\{b,c,e\})}), (p_{(\{a\},\{b,c,e\})}, b), (p_{(\{a\},\{b,c,e\})}, c), (p_{(\{a\},\{b,c,e\})}, e), (b, p_{(\{b,c\},\{d\})}$
The resulting Petri net is shown in Figure 4.2.

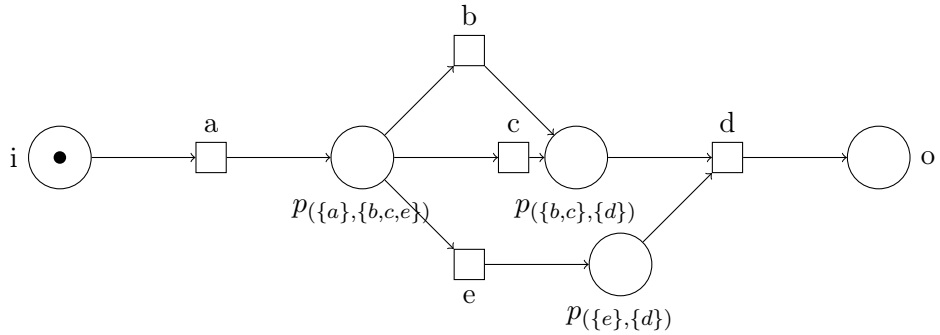


Figure 4.2: Petri net discovered by the Alpha algorithm

4.2.3 Limitations of the Alpha Algorithm

The Alpha algorithm has several limitations:

1. **Noise**: The Alpha algorithm does not handle noise well. A single outlier can significantly alter the discovered model.
2. **Infrequent behavior**: The Alpha algorithm treats all behavior equally, regardless of frequency.

3. **Incomplete logs**: The Alpha algorithm assumes that the event log contains all possible directly-follows relations.
4. **Loops of length one**: The Alpha algorithm cannot discover loops of length one ($a \rightarrow a$).
5. **Loops of length two**: The Alpha algorithm confuses loops of length two ($a \rightarrow b \rightarrow a$) with parallelism.
6. **Implicit places**: The Alpha algorithm may discover implicit places (places that do not constrain the behavior of the net).
7. **Non-local dependencies**: The Alpha algorithm cannot discover non-local dependencies.

Despite these limitations, the Alpha algorithm is a fundamental algorithm in process mining and has paved the way for more advanced process discovery techniques.

4.3 Advanced Process Discovery Algorithms

To overcome the limitations of the Alpha algorithm, several advanced process discovery algorithms have been developed:

4.3.1 Heuristic Miner

The Heuristic Miner is an extension of the Alpha algorithm that can handle noise and infrequent behavior by taking into account the frequency of directly-follows relations.

Dependency Measure

The key innovation of the Heuristic Miner is the introduction of a dependency measure:

Definition 4.7 (Dependency Measure) *Let L be an event log with activities A. The dependency measure between activities a and b is defined as:*

$$a \Rightarrow_L b = \begin{cases} \frac{|a>_L b| - |b>_L a|}{|a>_L b| + |b>_L a| + 1} & \text{if } a \neq b \\ \frac{|a>_L a|}{|a>_L a| + 1} & \text{if } a = b \end{cases}$$

The dependency measure ranges from -1 to 1:

- If $a \Rightarrow_L b$ is close to 1, there is a strong dependency from a to b.
- If $a \Rightarrow_L b$ is close to -1, there is a strong dependency from b to a.
- If $a \Rightarrow_L b$ is close to 0, there is no clear dependency between a and b.

Heuristic Miner Algorithm

The Heuristic Miner algorithm consists of the following steps:

1. Compute the directly-follows relation $>_L$ and count the frequencies.
2. Compute the dependency measure $a \Rightarrow_L b$ for all pairs of activities.
3. Build a dependency graph by including all pairs (a, b) with a dependency measure above a certain threshold.
4. Identify splits and joins in the dependency graph based on frequencies.
5. Convert the dependency graph to a Causal net (C-net) or a Petri net.

Unlike the Alpha algorithm, the Heuristic Miner can handle noise, infrequent behavior, and short loops by focusing on the most frequent and reliable directly-follows relations.

4.3.2 Region-Based Miner

The Region-Based Miner is a family of algorithms that use the theory of regions to discover process models. The basic idea is to first construct a transition system from the event log and then convert it to a Petri net using region theory.

Transition System Construction

A transition system is a directed graph where nodes represent states and edges represent transitions between states. To construct a transition system from an event log, we need to define what constitutes a state:

Definition 4.8 (State Representation) *Let $\sigma = \langle e_1, e_2, \ldots, e_n \rangle$ be a trace. A state in the transition system can be defined in various ways:*

- ***Sequence abstraction**: The state is represented by the sequence of activities observed so far.*
- ***Multiset abstraction**: The state is represented by the multiset of activities observed so far.*
- ***Set abstraction**: The state is represented by the set of activities observed so far.*

Additionally, we can consider different state encodings:

- **Past**: The state is determined by what has happened.
- **Future**: The state is determined by what will happen.

- **Past and future**: The state is determined by both what has happened and what will happen.

Example 4.2 (Transition System Construction) *Consider the event log* $L = [\langle a, b, c, d\rangle, \langle a, c, b, d\rangle]$.

Using the past with sequence abstraction, the states are:

- $[]$ *(empty trace)*
- $[a]$ *(after executing* a*)*
- $[a, b]$ *(after executing* a *and* b*)*
- $[a, c]$ *(after executing* a *and* c*)*
- $[a, b, c]$ *(after executing* a*,* b*, and* c*)*
- $[a, c, b]$ *(after executing* a*,* c*, and* b*)*
- $[a, b, c, d]$ *(after executing* a*,* b*,* c*, and* d*)*
- $[a, c, b, d]$ *(after executing* a*,* c*,* b*, and* d*)*

The resulting transition system is shown in Figure 4.3.

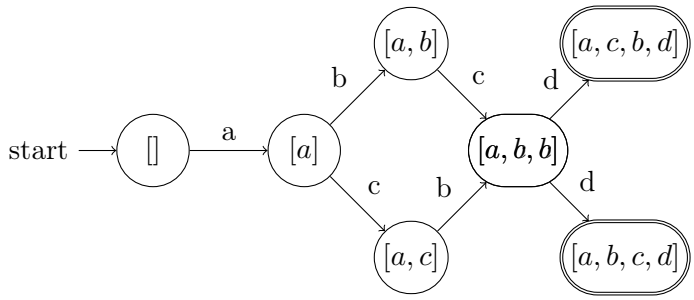


Figure 4.3: Transition system constructed from the event log

Region Theory

Once we have a transition system, we can use region theory to convert it to a Petri net. A region is a set of states in the transition system that behaves consistently with respect to the transitions:

Definition 4.9 (Region) *A set of states* R *in a transition system is a region if for every transition label* t*:*

- *Either all transitions labeled with* t *enter* R *(i.e., the source state is outside* R *and the target state is inside* R*),*

- *Or all transitions labeled with t exit R (i.e., the source state is inside R and the target state is outside R),*

- *Or all transitions labeled with t do not cross R (i.e., both the source and target states are inside R or both are outside R).*

Definition 4.10 (Minimal Region) *A region R is minimal if there is no proper subset of R that is also a region.*

Converting a Transition System to a Petri Net

The algorithm to convert a transition system to a Petri net using region theory is as follows:

1. Compute all minimal non-trivial regions in the transition system.

2. For each region, create a place in the Petri net.

3. For each transition label t that enters a region R, add an arc from the transition t to the corresponding place.

4. For each transition label t that exits a region R, add an arc from the corresponding place to the transition t.

5. Place a token in each place corresponding to a region that contains the initial state.

The resulting Petri net should exhibit the same behavior as the transition system.

Example 4.3 (Region-Based Mining) *Consider the transition system in Figure 4.3. Some of the minimal regions are:*

- $R_1 = \{[], [a, b, c, d], [a, c, b, d]\}$

- $R_2 = \{[a]\}$

- $R_3 = \{[a, b], [a, c]\}$

- $R_4 = \{[a, b, c], [a, c, b]\}$

The corresponding Petri net is shown in Figure 4.4.

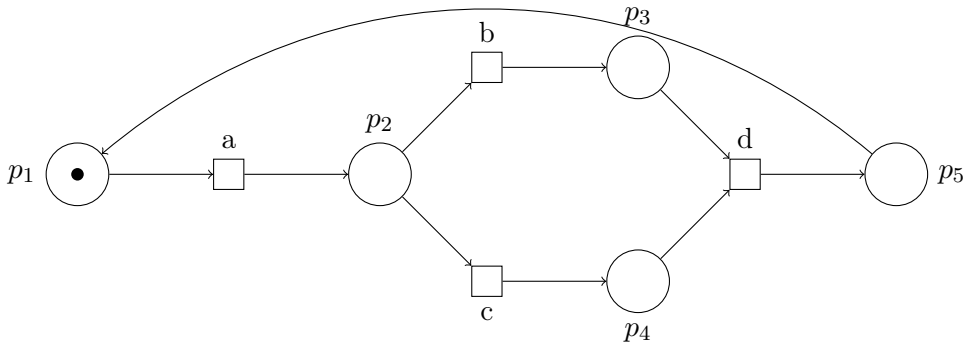


Figure 4.4: Petri net discovered using region theory

4.3.3 Inductive Miner

The Inductive Miner is a process discovery algorithm that recursively splits the event log based on detected patterns and constructs a process tree.

Process Trees

A process tree is a hierarchical representation of a process model where internal nodes represent operators (sequence, choice, parallel, loop) and leaf nodes represent activities.

Definition 4.11 (Process Tree) *A process tree is a tree where:*

- *Leaf nodes represent activities or the silent activity τ.*
- *Internal nodes represent operators: $\rightarrow$ (sequence), $\times$ (exclusive choice), $\wedge$ (parallel), $\circlearrowleft$ (loop).*

Process trees are a sound by construction, meaning that they always correspond to sound workflow nets.

Inductive Miner Algorithm

The basic idea of the Inductive Miner is to recursively split the event log and construct a process tree:

1. If the event log contains a single activity or empty traces, create a leaf node.
2. Otherwise, try to detect a cut, which is a pattern that splits the log:
    - **Sequence cut**: Split the log based on a sequence pattern.
    - **Exclusive choice cut**: Split the log based on a choice pattern.

- **Parallel cut**: Split the log based on a parallel pattern.
- **Loop cut**: Split the log based on a loop pattern.

3. If a cut is found, create an internal node with the corresponding operator and recursively apply the algorithm to the sublogs.

4. If no cut is found, use a fall-back option like creating a flower model or applying a different algorithm.

The Inductive Miner has several variants, including the Inductive Miner - Infrequent (IMf), which can handle noise by filtering out infrequent behavior.

4.4 Evaluating Process Discovery Results

Once a process model has been discovered, it is important to evaluate its quality. This can be done using the four quality criteria mentioned earlier: fitness, precision, generalization, and simplicity.

4.4.1 Fitness

Fitness measures how well the model can reproduce the behavior observed in the event log. A common way to calculate fitness is using token-based replay:

Definition 4.12 (Token-Based Fitness) *Given an event log L and a Petri net N, the token-based fitness is calculated as:*

$$fitness(L, N) = 1 - \frac{m + r}{c + p}$$

where:

- *m is the number of missing tokens (tokens that had to be artificially added during replay),*
- *r is the number of remaining tokens (tokens left in the net after replay),*
- *c is the number of consumed tokens,*
- *p is the number of produced tokens.*

4.4.2 Precision

Precision measures how well the model prohibits behavior that is not observed in the event log. One way to calculate precision is using the ETC (Escaping Transitions Count) method:

Definition 4.13 (ETC Precision) *Given an event log L and a Petri net N, the ETC precision is calculated as:*

$$precision(L, N) = 1 - \frac{\sum_{s \in S}(|av(s)| - |ex(s)|)}{\sum_{s \in S} |av(s)|}$$

where:

- *S is the set of states encountered during replay,*
- *$av(s)$ is the set of activities available in state s according to the model,*
- *$ex(s)$ is the set of activities actually executed in state s according to the log.*

4.4.3 Generalization

Generalization measures how well the model generalizes the behavior observed in the event log. One approach to estimate generalization is based on the frequency with which transitions are executed during replay:

Definition 4.14 (Frequency-Based Generalization) *Given an event log L and a Petri net N, the frequency-based generalization is calculated as:*

$$generalization(L, N) = 1 - \frac{\sum_{t \in T} \frac{1}{\sqrt{freq(t)+1}}}{|T|}$$

where:

- *T is the set of transitions in the Petri net,*
- *$freq(t)$ is the frequency with which transition t is executed during replay.*

4.4.4 Simplicity

Simplicity measures the complexity of the model. A simple approach is to count the number of nodes and arcs in the model:

Definition 4.15 (Size-Based Simplicity) *Given a Petri net $N = (P, T, F)$, the size-based simplicity is calculated as:*

$$simplicity(N) = 1 - \frac{|P| + |T| + |F|}{(|P| + |T|)^2}$$

4.4.5 Balancing the Quality Criteria

The four quality criteria often conflict with each other. For example, a model with perfect fitness might have poor precision, and a model with perfect precision might have poor generalization. The challenge is to find a balance that best suits the specific needs of the analysis.

4.5 Process Discovery in Practice

Process discovery in practice often involves the following steps:

1. **Data extraction and preprocessing**: Extract event data from information systems and preprocess it to create a clean event log.
2. **Filtering**: Filter the event log to focus on specific aspects of the process or to remove outliers.
3. **Discovery**: Apply process discovery algorithms to discover a process model.
4. **Evaluation**: Evaluate the quality of the discovered model using the four quality criteria.
5. **Enhancement**: Enhance the model with additional perspectives like time, resources, and data.
6. **Interpretation**: Interpret the results and identify potential process improvements.

Several tools are available for process discovery:

- **ProM**: An open-source framework with numerous plugins for process mining, including various process discovery algorithms.
- **Disco**: A commercial tool with a user-friendly interface for process mining.
- **Celonis**: A commercial platform for process mining with advanced features for process discovery and analysis.
- **PM4Py**: An open-source Python library for process mining.

Chapter 5

Conformance Checking

Conformance checking is the process of verifying whether the behavior observed in an event log conforms to a process model and vice versa. It is used to identify deviations, check compliance, and evaluate process discovery results.

5.1 Introduction to Conformance Checking

Conformance checking addresses questions like:

- Does the actual process execution, as recorded in the event log, conform to the process model?
- Are there deviations between the model and the recorded behavior? If so, where and why?
- How well does a discovered process model represent the behavior observed in the event log?

There are various reasons for performing conformance checking:

- **Auditing and compliance**: Checking whether process executions adhere to regulations, standards, or internal policies.
- **Model validation**: Validating whether a (discovered or manually created) process model accurately represents the real process.
- **Deviation detection**: Identifying exceptional behavior or violations that may indicate fraud, inefficiencies, or other issues.
- **Model comparison**: Comparing process models discovered using different algorithms or from different time periods.

5.2 Token-Based Replay

One approach to conformance checking is token-based replay, which involves replaying the traces from the event log on the process model and counting produced, consumed, missing, and remaining tokens.

5.2.1 Basic Idea

The basic idea of token-based replay is to simulate the execution of each trace in the event log on the process model (a Petri net) and track the tokens:

- Start with a token in the initial place.
- For each event in the trace, try to fire the corresponding transition.
- If the transition is not enabled, add missing tokens to its input places.
- After replaying the entire trace, count the remaining tokens (tokens left in places other than the final place).

5.2.2 Fitness Calculation

Based on the token-based replay, we can calculate a fitness value that measures how well the model can reproduce the behavior in the event log:

Definition 5.1 (Token-Based Fitness) *Given an event log L and a Petri net N, the token-based fitness is calculated as:*

$$fitness(L, N) = 1 - \frac{m + r}{c + p}$$

where:

- *m is the number of missing tokens (tokens that had to be artificially added during replay),*
- *r is the number of remaining tokens (tokens left in the net after replay),*
- *c is the number of consumed tokens,*
- *p is the number of produced tokens.*

Example 5.1 (Token-Based Replay) *Consider the Petri net model in Figure 5.1 and the trace $\sigma = \langle a, c, d, e \rangle$.*

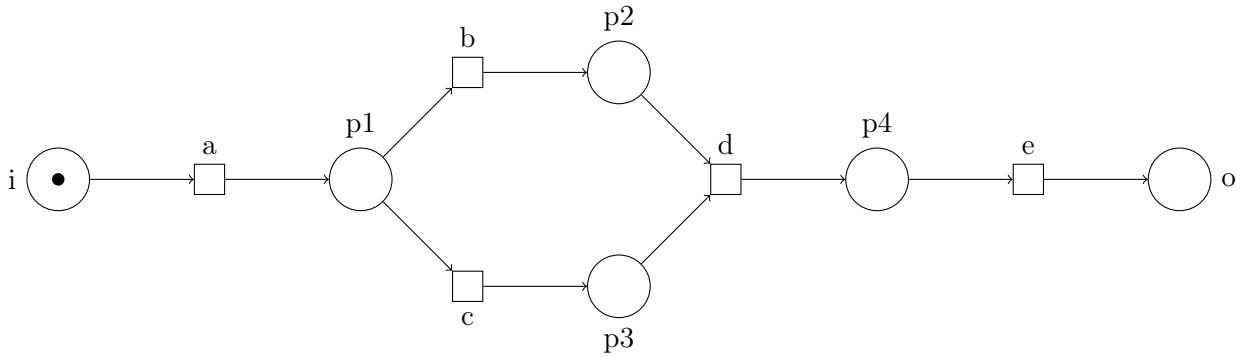


Figure 5.1: Petri net model for token-based replay

Step 1: *Initialize counters:* $p = 1$ *(initial token in place* i*),* $c = 0$*,* $m = 0$*,* $r = 0$*.*

Step 2: *Replay event* a*:*

- *Transition* a *is enabled, so it can fire.*
- *After firing, place* i *loses one token and place* $p1$ *gains one token.*
- *Update counters:* $p = 2$*,* $c = 1$*,* $m = 0$*,* $r = 0$*.*

Step 3: *Replay event* c*:*

- *Transition* c *is enabled, so it can fire.*
- *After firing, place* $p1$ *loses one token and place* $p3$ *gains one token.*
- *Update counters:* $p = 3$*,* $c = 2$*,* $m = 0$*,* $r = 0$*.*

Step 4: *Replay event* d*:*

- *Transition* d *requires tokens in both* $p2$ *and* $p3$*, but only* $p3$ *has a token. So,* d *is not enabled.*
- *Add a missing token to place* $p2$*.*
- *Now, transition* d *can fire, consuming tokens from* $p2$ *and* $p3$ *and producing a token in* $p4$*.*
- *Update counters:* $p = 4$*,* $c = 4$*,* $m = 1$*,* $r = 0$*.*

Step 5: *Replay event* e*:*

- *Transition* e *is enabled, so it can fire.*
- *After firing, place* $p4$ *loses one token and place* o *gains one token.*
- *Update counters:* $p = 5$*,* $c = 5$*,* $m = 1$*,* $r = 0$*.*

Step 6: *End of replay:*

- *All tokens are in the expected places (just one token in the final place o).*
- *Final counters:* $p = 5$, $c = 5$, $m = 1$, $r = 0$.
- *Calculate fitness:* $fitness(\sigma, N) = 1 - \frac{m+r}{c+p} = 1 - \frac{1+0}{5+5} = 1 - \frac{1}{10} = 0.9$.

5.3 Alignment-Based Conformance Checking

While token-based replay is intuitive, it has limitations: it cannot handle duplicate activities, invisible transitions, or non-local dependencies well. Alignment-based conformance checking addresses these limitations by aligning traces from the event log with the most similar model executions.

5.3.1 Alignments

An alignment is a sequence of moves that shows how a trace in the event log can be related to a valid execution of the process model. There are three types of moves:

Definition 5.2 (Types of Moves) *Given a trace* σ *and a process model* M*:*

- ***Synchronous move****: Both the log and the model execute the same activity.*
- ***Move on log only (log move)****: The log executes an activity that is not matched in the model.*
- ***Move on model only (model move)****: The model executes an activity that is not matched in the log.*

Definition 5.3 (Alignment) *An alignment between a trace* σ *and a process model* M *is a sequence of moves such that:*

- *The projection of the alignment on the log equals* σ *(ignoring "moves on model only").*
- *The projection of the alignment on the model represents a valid execution of* M *(ignoring "moves on log only").*

5.3.2 Optimal Alignments

Not all alignments are equally good. We want to find an alignment with the minimum number of deviations:

Definition 5.4 (Cost Function) *A cost function assigns a cost to each type of move:*

- $cost(a, a) = 0$ *for a synchronous move.*
- $cost(a, \gg) > 0$ *for a log move.*
- $cost(\gg, a) > 0$ *for a model move.*

Definition 5.5 (Optimal Alignment) *An optimal alignment is an alignment with the minimum total cost.*

Finding an optimal alignment can be formulated as a shortest path problem:

1. Construct a state space where states represent partial alignments.
2. Assign costs to transitions between states based on the cost function.
3. Find the shortest path from the initial state (empty alignment) to a final state (complete alignment).

Example 5.2 (Alignment) *Consider the trace* $\sigma = \langle a, c, d, e \rangle$ *and the Petri net model in Figure 5.1.*

An optimal alignment might be:

$$\gamma = \begin{pmatrix} a & c & \gg & d & e \\ a & c & b & d & e \end{pmatrix}$$

Where:

- *The top row represents the trace (with* $\gg$ *for model moves).*
- *The bottom row represents the model execution (with* $\gg$ *for log moves).*

In this alignment, there is one model move $(c, \gg)$*, meaning that the model executed activity* b *that was not in the log.*

5.3.3 Fitness based on Alignments

Once we have computed an optimal alignment for each trace in the event log, we can calculate a fitness value:

Definition 5.6 (Alignment-Based Fitness) *Given an event log L, a process model M, and optimal alignments γ_σ for each trace $\sigma \in L$, the alignment-based fitness is calculated as:*

$$fitness(L, M) = 1 - \frac{\sum_{\sigma \in L} cost(\gamma_\sigma)}{\sum_{\sigma \in L} worst\_cost(\sigma, M)}$$

where:

- *$cost(\gamma_\sigma)$ is the cost of the optimal alignment γ_σ.*
- *$worst\_cost(\sigma, M)$ is the cost of the worst possible alignment, typically calculated as the sum of the costs of moving the entire trace on log only and moving the shortest model execution on model only.*

5.4 Precision and Generalization

Besides fitness, other important quality criteria are precision and generalization.

5.4.1 Precision

Precision measures how well the model prohibits behavior that is not observed in the event log. A model with low precision allows for much more behavior than observed, which might indicate that the model is too general or "underfitting" the data.

Escaping Edges

One approach to measuring precision is based on "escaping edges":

Definition 5.7 (Escaping Edge) *An escaping edge is a transition that is allowed by the model after a specific prefix but is never observed in the event log after that prefix.*

Precision Calculation

Using the concept of escaping edges, we can calculate precision as follows:

Definition 5.8 (ETC Precision) *Given an event log L and a process model M, the ETC (Escaping Transitions Count) precision is calculated as:*

$$precision(L, M) = 1 - \frac{\sum_{s \in S}(|av(s)| - |ex(s)|)}{\sum_{s \in S} |av(s)|}$$

where:

- *S is the set of states encountered during replay,*
- *$av(s)$ is the set of activities available in state s according to the model,*
- *$ex(s)$ is the set of activities actually executed in state s according to the log.*

Example 5.3 (Precision Calculation) *Consider a process model that allows for sequences $\langle a, b, c\rangle$, $\langle a, b, d\rangle$, $\langle a, c, b\rangle$, and $\langle a, c, d\rangle$. Now imagine an event log with only two traces: $\langle a, b, c\rangle$ and $\langle a, c, b\rangle$.*

After observing prefix $\langle a, b\rangle$ in the log, the only activity ever executed next is c. However, the model allows for both c and d. This means that there is one escaping edge after prefix $\langle a, b\rangle$.

Similarly, after observing prefix $\langle a, c\rangle$ in the log, the only activity ever executed next is b. However, the model allows for both b and d. This means that there is one escaping edge after prefix $\langle a, c\rangle$.

If we compute the precision, we get:

$$precision(L, M) = 1 - \frac{(2-1) + (2-1)}{2+2} = 1 - \frac{2}{4} = 0.5$$

This indicates that the model allows for twice as much behavior as observed in the log.

5.4.2 Generalization

Generalization measures how well the model generalizes the behavior observed in the event log. A model with low generalization might be "overfitting" the data, meaning that it is too specific to the observed behavior and does not capture the underlying process well.

Measuring generalization is challenging, but one approach is based on the frequency with which transitions are executed during replay:

Definition 5.9 (Frequency-Based Generalization) *Given an event log L and a Petri net N, the frequency-based generalization is calculated as:*

$$generalization(L, N) = 1 - \frac{\sum_{t \in T} \frac{1}{\sqrt{freq(t)+1}}}{|T|}$$

where:

- *T is the set of transitions in the Petri net,*
- *$freq(t)$ is the frequency with which transition t is executed during replay.*

The intuition is that transitions that are executed more frequently are more likely to be correctly generalized. Transitions that are executed rarely might be artifacts of the specific event log rather than representative of the underlying process.

5.5 Conformance Checking in Practice

Conformance checking in practice often involves the following steps:

1. **Model selection**: Choose an appropriate process model to check against. This can be a discovered model, a reference model, or a compliance model.
2. **Log preparation**: Prepare the event log by filtering out irrelevant events, handling missing data, and ensuring correct ordering.
3. **Conformance analysis**: Apply conformance checking techniques to identify deviations and measure conformance.
4. **Result interpretation**: Interpret the results, focusing on significant deviations and their root causes.
5. **Process improvement**: Use the insights gained to improve the process or the model.

Several tools support conformance checking:

- **ProM**: Offers various plugins for conformance checking, including token-based replay and alignment-based approaches.
- **Disco**: Provides features for filtering and analyzing event logs to identify deviations.
- **Celonis**: Includes conformance checking capabilities and visualizations to identify and analyze deviations.
- **PM4Py**: Supports alignment-based conformance checking and other techniques.

Chapter 6

Mining Additional Perspectives

So far, we have focused on the control-flow perspective of processes. However, processes have multiple perspectives, including organizational (who does what), time (when things happen), and data (what information is processed). Mining these additional perspectives can provide valuable insights for process analysis and improvement.

6.1 Organizational Mining

Organizational mining focuses on the organizational perspective of processes, analyzing how resources (people, roles, departments) are involved in the execution of activities.

6.1.1 Resource-Activity Matrix

The resource-activity matrix is a basic tool for organizational mining. It shows which resources perform which activities and how frequently:

Definition 6.1 (Resource-Activity Matrix) *A resource-activity matrix M is a matrix where:*

- *Rows represent resources,*
- *Columns represent activities,*
- *Each cell $M_{r,a}$ contains the number of times resource r performed activity a, often normalized by the number of cases.*

Example 6.1 (Resource-Activity Matrix) *Consider an event log with activities a, b, c, d, and e, and resources Pete,Mike, Ellen, Sue, Sean, and Sara. The resource-activity matrix might look like this:*

Resource	a	b	c	d	e
Pete	0.3	0.1	0	0.2	0
Mike	0.5	0.4	0	0.3	0
Ellen	0.2	0.5	0	0.5	0
Sue	0	0	0.5	0	0
Sean	0	0	0.5	0	0
Sara	0	0	0	0	1.0

Table 6.1: Example of a resource-activity matrix

From this matrix, we can observe that:

- *Activity a is performed by Pete (30%), Mike (50%), and Ellen (20%).*
- *Activity b is performed by Pete (10%), Mike (40%), and Ellen (50%).*
- *Activity c is performed by Sue (50%) and Sean (50%).*
- *Activity d is performed by Pete (20%), Mike (30%), and Ellen (50%).*
- *Activity e is performed exclusively by Sara (100%).*

6.1.2 Role Discovery

One of the key goals of organizational mining is to discover roles, which are groups of resources that perform similar activities:

Definition 6.2 (Role) *A role is a group of resources that perform similar activities, often with similar frequencies.*

Roles can be discovered by clustering resources based on their profiles in the resource-activity matrix. Various clustering algorithms can be used, such as k-means, hierarchical clustering, or spectral clustering.

Example 6.2 (Role Discovery) *Based on the resource-activity matrix in Table 6.1, we might discover the following roles:*

- ***Role 1 (Frontline Workers)**: Pete, Mike, and Ellen, who mainly perform activities a, b, and d.*
- ***Role 2 (Specialists)**: Sue and Sean, who exclusively perform activity c.*
- ***Role 3 (Manager)**: Sara, who exclusively performs activity e.*

6.1.3 Social Network Analysis

Another aspect of organizational mining is social network analysis, which studies the interactions between resources.

Definition 6.3 (Social Network) *A social network in the context of process mining is a graph where:*

- *Nodes represent resources,*
- *Edges represent relationships or interactions between resources,*
- *Edge weights represent the strength or frequency of the interactions.*

Handover of Work

A common type of relationship in social network analysis is the handover of work, which occurs when one resource completes an activity and another resource starts the next activity:

Definition 6.4 (Handover of Work) *A handover of work from resource r_1 to resource r_2 occurs when:*

- *Resource r_1 completes an activity a,*
- *Resource r_2 later performs an activity b,*
- *Activity a is directly or indirectly followed by activity b in the process.*

Example 6.3 (Handover of Work) *Consider an event log with the following trace:*

$$\sigma = \langle (a, Pete), (b, Mike), (c, Sue), (d, Ellen), (e, Sara) \rangle$$

The handovers of work in this trace are:

- *Pete to Mike (from activity a to b),*
- *Mike to Sue (from activity b to c),*
- *Sue to Ellen (from activity c to d),*
- *Ellen to Sara (from activity d to e).*

By analyzing multiple traces, we can build a handover of work matrix and visualize it as a social network.

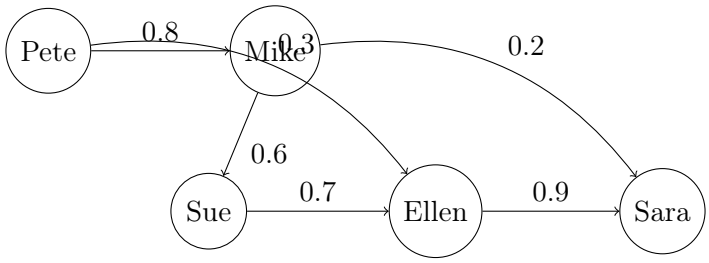


Figure 6.1: Example of a social network based on handover of work

6.1.4 Organizational Mining in Practice

Organizational mining can provide valuable insights for process improvement and organizational design:

- **Role analysis**: Identifying roles and their responsibilities can help in better role design and workload distribution.
- **Resource utilization**: Analyzing how resources are utilized can help identify bottlenecks and optimize resource allocation.
- **Collaboration patterns**: Understanding how resources collaborate can improve team organization and communication.
- **Process handovers**: Analyzing handovers of work can identify inefficient transfer of work between resources or departments.

Tools like ProM provide plugins for organizational mining, including resource-activity matrix analysis, role discovery, and social network analysis.

6.2 Data-Perspective Mining

Data-perspective mining focuses on how data affects the process flow, particularly at decision points where the process can take different paths based on data conditions.

6.2.1 Decision Mining

Decision mining aims to discover the rules that govern the choices at decision points in a process:

Definition 6.5 (Decision Point) *A decision point in a process is a point where the process can take different paths, typically represented by an XOR-split in the process model.*

Definition 6.6 (Decision Rule) *A decision rule at a decision point specifies the conditions under which a specific path is chosen.*

Decision mining uses machine learning techniques, particularly decision trees, to discover these rules from event data.

Decision Tree Learning

Decision tree learning is a common technique for decision mining:

1. Identify decision points in the process model (XOR-splits).
2. For each decision point, extract cases from the event log and their attributes at the time of the decision.
3. Apply decision tree learning to discover the rules that best predict the chosen path.

Example 6.4 (Decision Mining) *Consider a loan application process with a decision point after the "Assess Application" activity, where the process can either take the "Approve Loan" path or the "Reject Loan" path.*

By analyzing the event data, we might discover the following decision rule:

- *If the applicant's credit score is greater than 700 and the loan amount is less than $50,000, take the "Approve Loan" path.*
- *Otherwise, take the "Reject Loan" path.*

This rule can be represented as a decision tree, as shown in Figure 6.2.

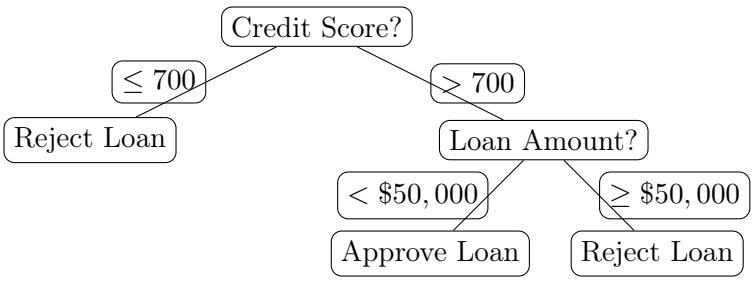


Figure 6.2: Example of a decision tree for loan approval

6.2.2 Guard Discovery

In process models with data, guards are conditions that determine when a transition can fire:

Definition 6.7 (Guard) *A guard is a condition attached to a transition or an arc in a process model that must be satisfied for the transition to be enabled.*

Guard discovery is the process of learning these conditions from event data:

1. For each transition or arc, identify the cases where it fired and the cases where it was enabled but didn't fire.
2. Extract the data attributes at the time of enablement.
3. Apply machine learning techniques to discover the conditions that best discriminate between the "fired" and "not fired" cases.

Example 6.5 (Guard Discovery) *In a Petri net model of a loan application process, there might be transitions for "Standard Assessment" and "Detailed Assessment". By analyzing the event data, we might discover the following guards:*

- *Guard for "Standard Assessment": Loan amount $<$ \$10,000*
- *Guard for "Detailed Assessment": Loan amount $\geq$ \$10,000*

6.2.3 Data-Aware Process Discovery

Data-aware process discovery aims to discover process models that incorporate data elements and their influence on the process flow. These models can include:

- **Data objects**: Entities processed by the process, such as applications, invoices, or orders.
- **Data attributes**: Properties of data objects, such as amount, status, or category.
- **Data flows**: The flow of data between activities.
- **Data dependencies**: How data affects the process flow.

Example 6.6 (Data-Aware Process Model) *Figure 6.3 shows a data-aware process model for a loan application process. The model includes data objects (Application, Assessment, Decision), data attributes (Amount, Status), and guards on transitions.*

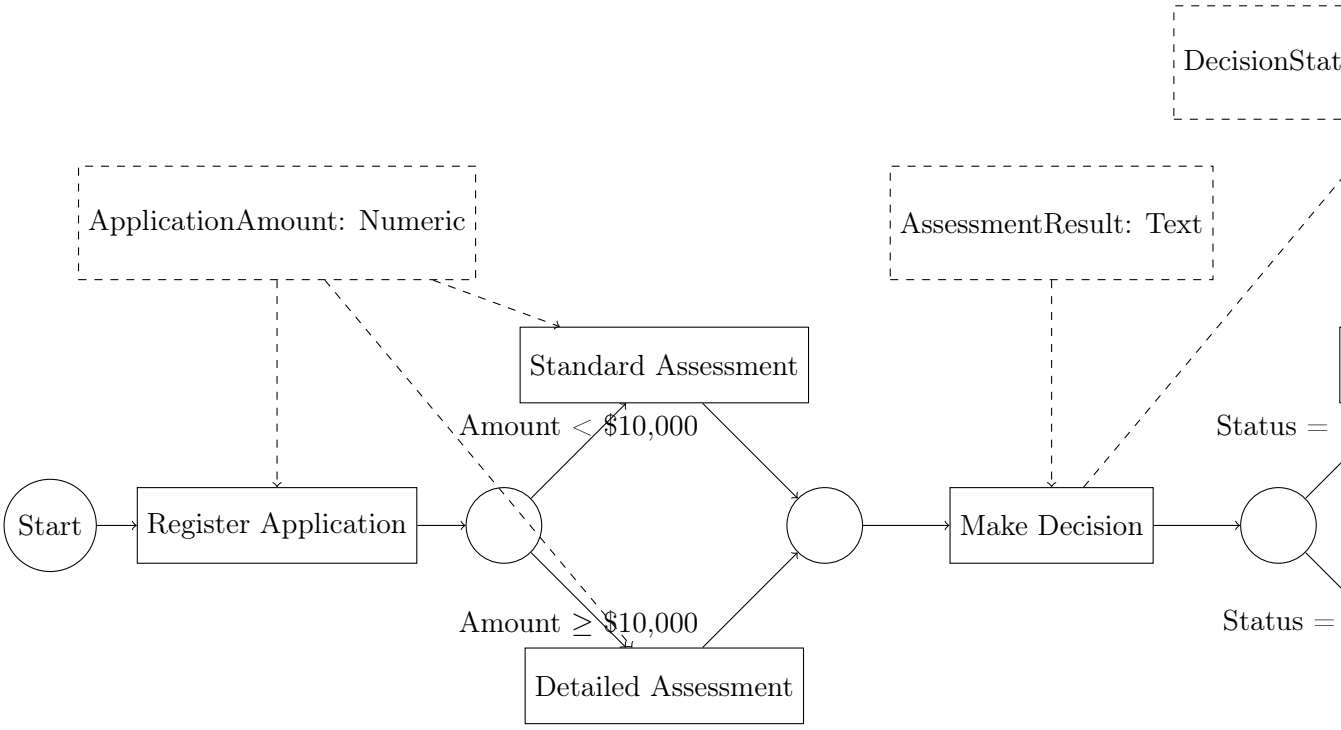


Figure 6.3: Data-aware process model for a loan application process

6.3 Time-Perspective Mining

Time-perspective mining focuses on the temporal aspects of processes, including activity durations, waiting times, and process cycle times.

6.3.1 Performance Analysis

Performance analysis aims to understand the time-related aspects of process execution:

Definition 6.8 (Time Measures) *Common time measures in process mining include:*

- ***Activity duration**: The time it takes to execute an activity, from the moment it starts until it completes.*
- ***Waiting time**: The time between the completion of one activity and the start of the next activity.*
- ***Synchronization time**: The time spent waiting for other parallel branches to complete.*

- ***Cycle time**: The total time from the start to the end of a process instance.*
- ***Service time**: The total time spent executing activities (sum of activity durations).*
- ***Sojourn time**: The time between the completion of one activity and the completion of the next activity.*

Example 6.7 (Performance Analysis) *Consider an event log with the following trace:*

$\sigma = \langle(a, start, 10:00), (a, complete, 10:30), (b, start, 11:00), (b, complete, 11:45), (c, start, 13:0$

From this trace, we can compute:

- *Activity duration of a: 10:30 - 10:00 = 30 minutes*
- *Activity duration of b: 11:45 - 11:00 = 45 minutes*
- *Activity duration of c: 14:15 - 13:00 = 75 minutes*
- *Waiting time between a and b: 11:00 - 10:30 = 30 minutes*
- *Waiting time between b and c: 13:00 - 11:45 = 75 minutes*
- *Cycle time: 14:15 - 10:00 = 255 minutes*
- *Service time: 30 + 45 + 75 = 150 minutes*

By analyzing multiple traces, we can compute statistics for these measures, such as average, minimum, maximum, and standard deviation.

6.3.2 Bottleneck Analysis

Bottleneck analysis aims to identify activities or paths in the process that take a disproportionate amount of time:

Definition 6.9 (Bottleneck) *A bottleneck is an activity or path in a process that significantly slows down the overall process execution.*

Bottlenecks can be identified by:

- Analyzing activity durations and waiting times,
- Identifying activities with high execution times or high variability,
- Analyzing resource utilization,
- Identifying paths with long synchronization times.

Example 6.8 (Bottleneck Analysis) *Consider a process with average activity durations and waiting times as shown in Figure 6.4.*

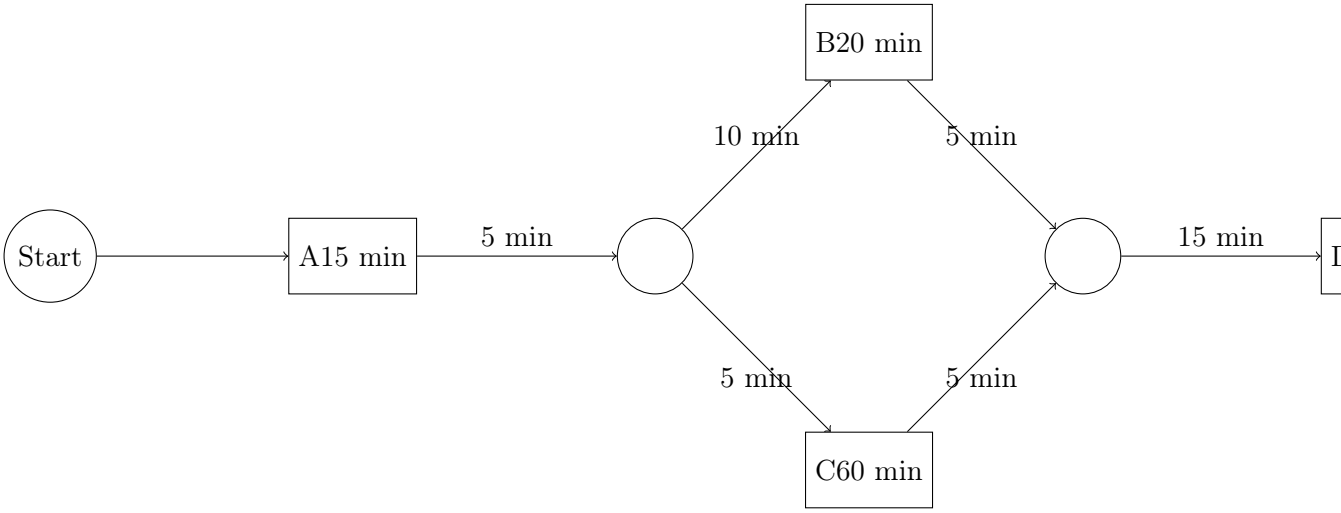


Figure 6.4: Process model with activity durations and waiting times

From this analysis, we can identify activity C as a bottleneck because:

- *It has a much longer duration (60 minutes) compared to other activities (15-30 minutes).*
- *It causes a long synchronization time at merge point p2, as activity B completes much earlier.*

6.3.3 Time-Based Process Discovery

Time-based process discovery aims to discover process models that incorporate time aspects:

- **Performance-annotated process models**: Process models annotated with time information such as average durations, waiting times, and frequencies.
- **Performance spectrum**: A visualization that shows the distribution of process executions over time, highlighting performance patterns and anomalies.
- **Time prediction models**: Models that predict the remaining time until process completion based on the current state and historical data.

Example 6.9 (Performance-Annotated Process Model) *Figure 6.5 shows a performance-annotated process model with average activity durations, waiting times, and frequencies.*

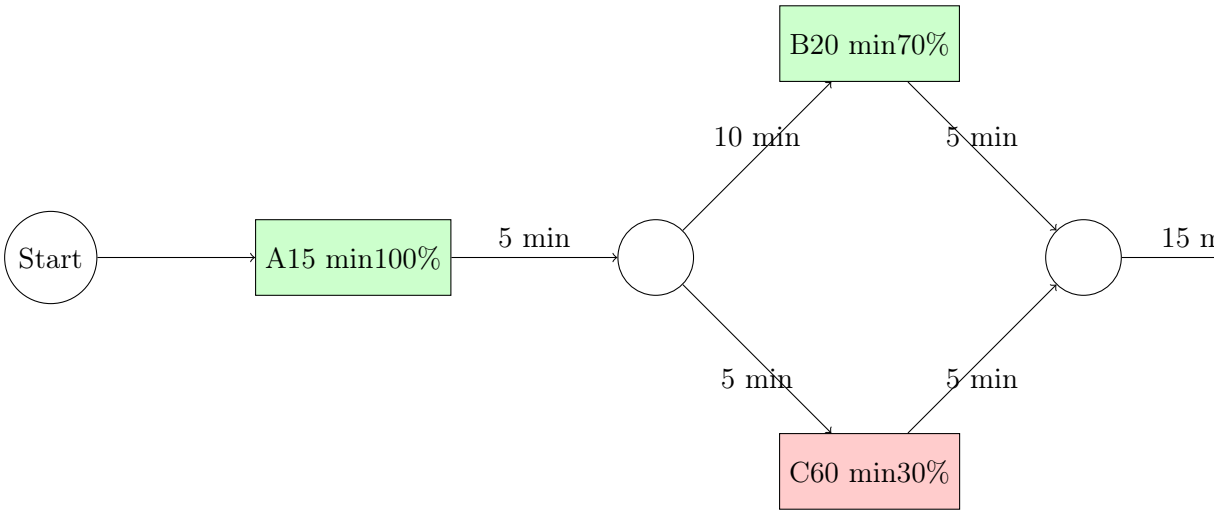


Figure 6.5: Performance-annotated process model

In this model:

- *Activities are colored based on their durations (green: fast, yellow: medium, red: slow).*
- *Each activity is annotated with its average duration and execution frequency.*
- *Arcs are annotated with average waiting times.*

Chapter 7

Business Process Modeling and Simulation with BPMN

Business Process Model and Notation (BPMN) is a widely used standard for modeling business processes. It provides a graphical notation that is easy to understand by both business and technical users. In addition to modeling, BPMN models can be used for process simulation, which allows for performance analysis and what-if scenarios.

7.1 Introduction to BPMN

BPMN was developed by the Object Management Group (OMG) and is now a widely adopted standard for business process modeling.

Definition 7.1 (BPMN) *Business Process Model and Notation (BPMN) is a graphical representation for specifying business processes in a business process model.*

7.1.1 Basic BPMN Elements

BPMN includes four basic categories of elements:

1. **Flow Objects**: Events, Activities, and Gateways.
2. **Connecting Objects**: Sequence Flows, Message Flows, and Associations.
3. **Swimlanes**: Pools and Lanes.
4. **Artifacts**: Data Objects, Groups, and Annotations.

Flow Objects

- **Events**: Something that happens during the course of a process. Events can start, interrupt, or end a process flow.
- **Activities**: Work that is performed within a process. Activities can be atomic (tasks) or compound (sub-processes).
- **Gateways**: Control points that determine the flow of the process based on conditions.

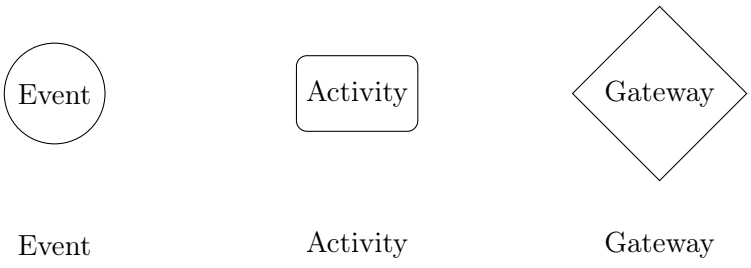


Figure 7.1: Basic BPMN flow objects

Connecting Objects

- **Sequence Flow**: Shows the order of activities in a process.
- **Message Flow**: Shows the flow of messages between participants.
- **Association**: Links artifacts to flow objects.

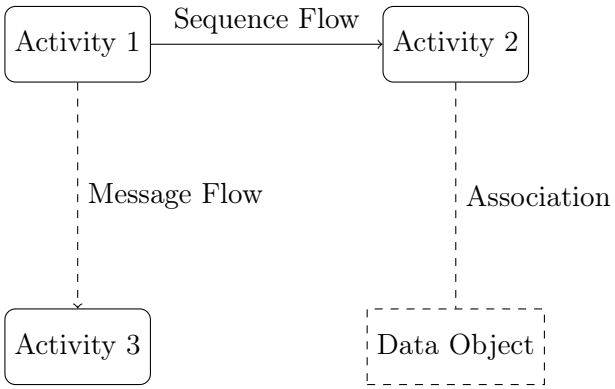


Figure 7.2: BPMN connecting objects

Swimlanes

- **Pool**: Represents a participant in a process, such as an organization or a system.

- **Lane**: A sub-partition within a pool, often representing roles or departments.

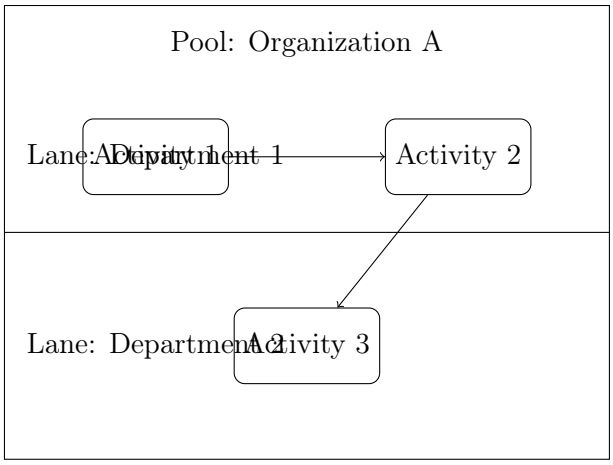


Figure 7.3: BPMN swimlanes

Artifacts

- **Data Objects**: Represent information flowing through the process.

- **Groups**: Group elements together for documentation or analysis purposes.

- **Annotations**: Provide additional information for readers of the BPMN diagram.

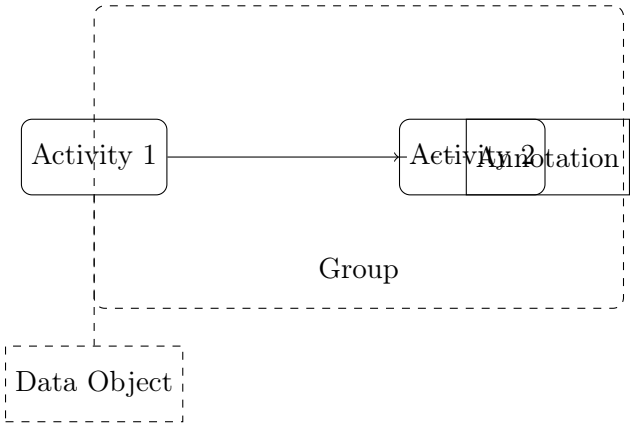


Figure 7.4: BPMN artifacts

7.1.2 BPMN vs. Petri Nets

BPMN and Petri nets are both used for process modeling, but they have different strengths and weaknesses:

- **Expressiveness**: BPMN is more expressive, with a richer set of modeling constructs.
- **Formality**: Petri nets have a formal semantics, which enables rigorous analysis.
- **Accessibility**: BPMN is generally more accessible to business users.
- **Tool support**: BPMN has wider tool support in industry.

Aspect	**BPMN**	**Petri Nets**
Modeling constructs	Rich set of constructs, including events, activities, gateways, and artifacts	Basic set of constructs: places, transitions, and arcs
Semantics	Semi-formal semantics, with some ambiguities	Formal semantics, with clear token game rules
Analysis	Limited formal analysis, focus on simulation	Rich set of formal analysis techniques
Target audience	Business analysts and technical developers	Primarily technical users and academics
Tool support	Wide range of commercial and open-source tools	More limited tool support, primarily in academic context

Table 7.1: Comparison of BPMN and Petri nets

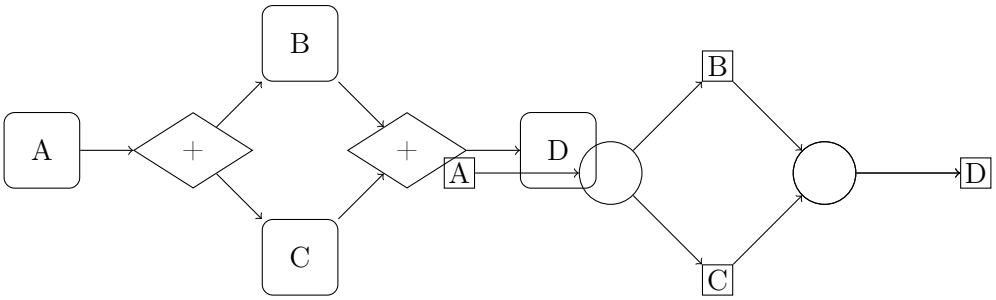
7.2 Mapping BPMN to Petri Nets

To leverage the formal analysis techniques available for Petri nets, BPMN models can be mapped to Petri nets. This mapping is not always straightforward due to the richer semantics of BPMN, but it can be done for many common patterns.

7.2.1 Basic Mapping Patterns

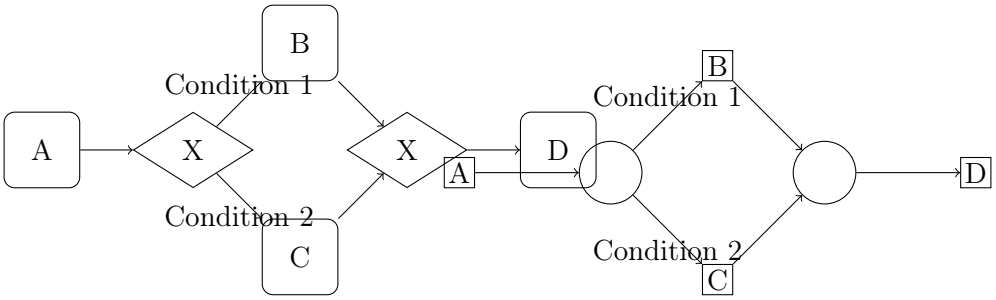
(a) BPMN Sequence (b) Petri Net Equivalent

Figure 7.5: Mapping of sequence pattern



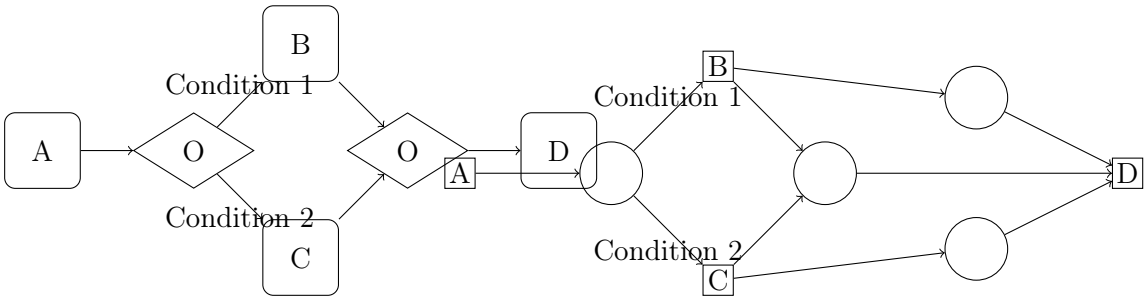
(a) BPMN Parallel Gateway (b) Petri Net Equivalent

Figure 7.6: Mapping of parallel gateway pattern



(a) BPMN Exclusive Gateway (b) Petri Net Equivalent

Figure 7.7: Mapping of exclusive gateway pattern



(a) BPMN Inclusive Gateway (b) Petri Net Equivalent

Figure 7.8: Mapping of inclusive gateway pattern

7.2.2 Challenges in Mapping BPMN to Petri Nets

While basic patterns can be mapped relatively easily, there are challenges in mapping more complex BPMN constructs to Petri nets:

- **Events**: BPMN has a rich set of event types, many of which don't have direct equivalents in Petri nets.
- **Data**: BPMN can model data objects and data flows, which are not directly supported in basic Petri nets.
- **Resources**: BPMN pools and lanes represent organizational resources, which are not part of basic Petri net models.
- **Exceptions and Compensations**: BPMN has specific constructs for exceptions and compensations, which are challenging to map to Petri nets.

Various extensions of Petri nets, such as Colored Petri Nets (CPN) and Workflow Nets, can address some of these challenges, but a complete and precise mapping remains a challenge.

7.3 Business Process Simulation

Business Process Simulation (BPS) is a technique for analyzing business processes by simulating their execution over time.

Definition 7.2 (Business Process Simulation) *Business Process Simulation is the act of imitating the operation of a real-world business process over time.*

7.3.1 Why Simulate Business Processes?

Simulation offers several benefits for business process analysis:

- **Performance analysis**: Understand the performance of processes without disrupting real operations.
- **What-if analysis**: Evaluate potential changes before implementing them.
- **Bottleneck identification**: Identify bottlenecks and resource constraints.
- **Resource planning**: Determine optimal resource allocation.
- **Process optimization**: Identify opportunities for process improvement.
- **Cost-benefit analysis**: Evaluate the costs and benefits of process changes.

7.3.2 Elements of a Simulation Model

A business process simulation model includes:

- **Process model**: The structure of the process, typically in BPMN.
- **Process instances**: Cases or transactions that flow through the process.
- **Activity parameters**: Processing times, costs, and resource requirements for activities.
- **Resources**: Available resources, their skills, and schedules.
- **Routing rules**: Rules that determine how process instances flow through the process.
- **Arrival pattern**: The pattern and rate at which new process instances arrive.

7.3.3 Processing Times

In simulation, processing times can be modeled using various distributions:

- **Fixed time**: A constant time for all process instances.
- **Normal distribution**: Processing times are normally distributed around a mean.

- **Exponential distribution**: Processing times follow an exponential distribution, often used for service times.
- **Uniform distribution**: Processing times are uniformly distributed between a minimum and maximum.
- **Empirical distribution**: Processing times are based on observed data.

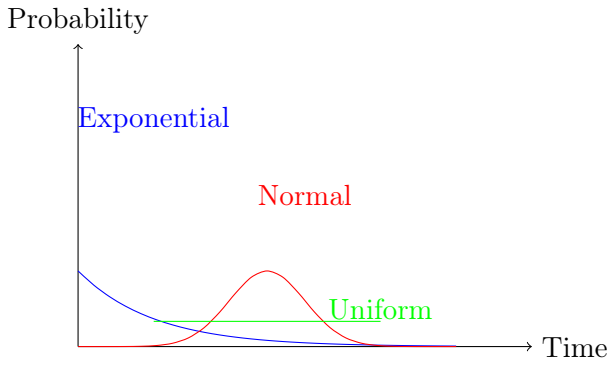


Figure 7.9: Common probability distributions for processing times

7.3.4 Arrival Patterns

The arrival of new process instances can follow various patterns:

- **Constant rate**: Process instances arrive at a constant rate.
- **Poisson process**: Arrivals follow a Poisson process, with exponentially distributed inter-arrival times.
- **Time-dependent**: Arrival rates vary over time, e.g., higher during business hours.
- **Calendar-based**: Arrivals follow a calendar pattern, e.g., weekdays vs. weekends.
- **Empirical**: Arrival patterns are based on observed data.

7.3.5 Resource Allocation

Resources in a simulation model can be allocated in various ways:

- **Role-based**: Activities require resources with specific roles, and resources are allocated based on their roles.
- **Priority-based**: Resources are allocated based on priorities.

- **Skill-based**: Resources are allocated based on their skills.
- **Availability-based**: Resources are allocated based on their availability, considering work schedules and other commitments.
- **Pooled**: Resources are pooled together and allocated as needed.

7.3.6 Simulation Output Analysis

The output of a business process simulation can include:

- **Process performance metrics**: Cycle time, service time, waiting time, and throughput.
- **Resource utilization**: The percentage of time resources are busy.
- **Queue statistics**: Queue lengths, waiting times, and waiting time distributions.
- **Cost metrics**: Process costs, activity costs, and resource costs.
- **Bottleneck analysis**: Identification of bottlenecks and their causes.

Example 7.1 (Simulation Output Analysis) *Figure 7.10 shows an example of simulation output analysis, including process performance metrics and resource utilization.*

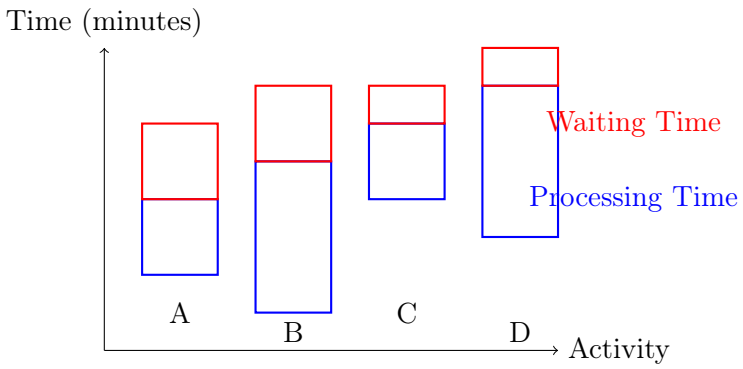


Figure 7.10: Example of simulation output analysis

7.3.7 What-If Analysis

What-if analysis is a key application of business process simulation:

Definition 7.3 (What-If Analysis) *What-if analysis is the process of changing input parameters in a simulation model to see how those changes affect the output.*

Common what-if scenarios include:

- **Resource changes**: Adding or removing resources, changing resource skills or schedules.
- **Process changes**: Changing the process structure, adding or removing activities, changing routing rules.
- **Workload changes**: Changing the arrival rate or pattern of process instances.
- **Priority changes**: Changing the priority of activities or process instances.
- **Technology changes**: Introducing automation or new technologies.

Example 7.2 (What-If Analysis) *A company wants to reduce the cycle time of its customer service process. Through simulation, they test the following scenarios:*

1. *Adding one more customer service representative.*
2. *Automating the initial customer data collection.*
3. *Changing the routing rules to prioritize urgent requests.*

The simulation results might show that automating the initial data collection provides the most significant reduction in cycle time with the lowest cost.

7.3.8 Business Process Simulation Tools

Several tools support business process simulation:

- **Bizagi Modeler**: A BPMN modeling tool with simulation capabilities.
- **ARIS Simulation**: Part of the ARIS platform, supporting detailed business process simulation.
- **Arena Simulation**: A general-purpose simulation tool with capabilities for business process simulation.
- **Signavio Process Intelligence**: A process mining and simulation tool.
- **Simul8**: A simulation tool with a focus on business process simulation.
- **BIMP**: A free online BPMN simulator.

Chapter 8

Conclusion and Future Directions

Throughout this course, we have explored the field of process mining, covering its main components: process discovery, conformance checking, and enhancement. We have also delved into related areas such as process modeling with Petri nets and BPMN, and business process simulation.

8.1 Summary of Process Mining

Process mining bridges the gap between process science and data science, using event data to discover, monitor, and improve real processes. It provides a data-driven approach to process analysis, complementing traditional model-based approaches.

The main types of process mining are:

- **Process Discovery**: Automatically constructing process models from event logs.
- **Conformance Checking**: Comparing event logs with process models to identify deviations.
- **Process Enhancement**: Extending process models with additional perspectives such as time, resources, and data.

These techniques can be applied in various domains, including healthcare, manufacturing, financial services, and public administration.

8.2 Integration with Other Technologies

Process mining can be integrated with other technologies and approaches:

- **Business Intelligence (BI)**: Process mining can complement BI by providing process-centric insights.
- **Artificial Intelligence (AI)**: AI techniques can enhance process mining, e.g., for predictive process monitoring.
- **Robotic Process Automation (RPA)**: Process mining can identify opportunities for automation and monitor RPA bots.
- **Business Process Management (BPM)**: Process mining can support the BPM lifecycle, from process design to implementation and evaluation.
- **Enterprise Architecture (EA)**: Process mining can provide insights into the actual operation of an organization's processes, informing EA decisions.

8.3 Challenges and Future Directions

Despite its benefits, process mining faces several challenges:

- **Data quality**: Event logs may be incomplete, contain noise, or have inconsistencies.
- **Complex processes**: Real-world processes can be complex, with many variants and exceptions.
- **Privacy and security**: Process mining may involve sensitive data, raising privacy and security concerns.
- **Integration with existing systems**: Integrating process mining with existing IT systems can be challenging.
- **Interpretation of results**: Process mining results can be complex and require domain knowledge to interpret.

Future directions in process mining include:

- **Real-time process mining**: Applying process mining techniques to streaming event data.
- **Predictive process mining**: Using historical process data to predict future process behavior.
- **Prescriptive process mining**: Recommending actions based on process mining insights.

- **Process mining with unstructured data**: Incorporating unstructured data such as text and images into process mining.
- **Process mining in IoT and edge computing**: Applying process mining to Internet of Things (IoT) and edge computing scenarios.

8.4 Conclusion

Process mining is a powerful approach for discovering, monitoring, and improving processes based on event data. By combining data science techniques with process science, it provides a unique perspective on how processes actually operate. As organizations continue to digitize their operations, the importance of process mining is likely to grow, enabling data-driven process improvement and operational excellence.

This course has provided a foundation in process mining concepts, techniques, and tools. The knowledge and skills acquired can be applied in various domains to improve processes, reduce costs, enhance compliance, and increase customer satisfaction.

Bibliography

[1] van der Aalst, W. M. P. (2016). *Process Mining: Data Science in Action*. Berlin: Springer-Verlag.

[2] van der Aalst, W. M. P. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Berlin: Springer-Verlag.

[3] van der Aalst, W. M. P., & Stahl, C. (2004). *Modeling Business Processes: A Petri Net-Oriented Approach*. Cambridge, MA: MIT Press.

[4] Dumas, M., La Rosa, M., Mendling, J., & Reijers, H. A. (2018). *Fundamentals of Business Process Management*. Berlin: Springer-Verlag.

[5] Weijters, A. J. M. M., van der Aalst, W. M. P., & de Medeiros, A. K. A. (2006). Process Mining with the Heuristics Miner-algorithm. *BETA Working Paper Series*, 166, 1-34.

[6] de Leoni, M., & van der Aalst, W. M. P. (2014). Data-aware Process Mining: Discovering Decisions in Processes Using Alignments. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 1454-1461.

[7] Rozinat, A., & van der Aalst, W. M. P. (2008). Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1), 64-95.

[8] Adriansyah, A., van Dongen, B. F., & van der Aalst, W. M. P. (2011). Conformance Checking Using Cost-Based Fitness Analysis. *IEEE International Enterprise Distributed Object Computing Conference*, 55-64.

[9] Song, M., & van der Aalst, W. M. P. (2008). Towards Comprehensive Support for Organizational Mining. *Decision Support Systems*, 46(1), 300-317.