Advanced Process Discovery: Heuristics and Regions
Overview

Part I: Preliminaries
Chapter 2 Process Modeling and Analysis
Chapter 3 Data Mining

Part II: From Event Logs to Process Models
Chapter 4 Getting the Data
Chapter 5 Process Discovery: An Introduction
Chapter 6 Advanced Process Discovery Techniques

Part III: Beyond Process Discovery
Chapter 7 Conformance Checking
Chapter 8 Mining Additional Perspectives
Chapter 9 Operational Support

Part IV: Putting Process Mining to Work
Chapter 10 Tool Support
Chapter 11 Analyzing "Lasagna Processes"
Chapter 12 Analyzing "Spaghetti Processes"

Part V: Reflection
Chapter 13 Cartography and Navigation
Chapter 14 Epilogue
Process discovery

- "world": people, machines, components, organizations
- records events, e.g., messages, transactions, etc.
- models, analyzes
- supports/controls
- specifies, configures, implements, analyzes
- discovers
- conformance
- enhancement
- software system
- event logs
- (process) model
- business processes

People, machines, and organizations interact within the "world". Event logs record activities, which are then analyzed and modeled to discover processes. These processes support and control aspects of the "world". The models are analyzed, specified, configured, and implemented to enhance and conform to the processes, ultimately leading to improved business processes.
Desire lines
Process model with perfect fitness
Different representational bias (triangles of fixed size)
Infrequent behavior (noise)
Overfitting?
Underfitting?
Balance four forces

- **Fitness**: ability to explain observed behavior
- **Precision**: avoiding overfitting
- **Generalization**: avoiding underfitting
- **Simplicity**: Occam’s Razor

Forces:
- **Lift**
- **Drag**
- **Thrust**
- **Gravity**
Challenges

What representation to use? The real process has no representation, it may only exist in the minds of people. (Even the later may not be the case: it may emerge from collaboration.)

We only have positive examples! These typically cover a fraction of all possible behaviors and there are no negatives examples.

Outside the laboratory we do not know it (at least not with 100% certainty).
representational bias
Anscombe's Quartet  (Francis Anscombe 1973)

Representational bias problem: simple linear regression may not be the best way to describe the relationship between two variables.

mean x = 9
var. x = 11
mean y = 7.5
var. y = 4.12
corr. = 0.816
same lin. regr.
BPM's Tower of Babel

endless varieties of process model notations
Representational bias
Search bias
YAWL

- **Start**
- **Register request**
- **Examine thoroughly**
  - **OR-split**
  - **OR-join**
- **Examine casually**
- **Check ticket**
- **Decide**
- **Pay compensation**
- **Reject request**
- **New information**
- **End**

Condition (like a place in a Petri net)
- **Task (i.e., an atomic activity)**
- **AND-split**
- **AND-join**
- **XOR-split**
- **XOR-join**
- **OR-split**
- **OR-join**
- **Start**
- **End**
- **Multiple instance task**
- **Composite task**
- **Cancelation region**
deferred choice pattern using the event-based XOR gateway

start

register request

examine casually

check ticket

reinitiate request

decline

casually examine

thoroughly examine

pay compensation

reject request

reinitiate request

end
Event-Driven Process Chains (EPCs)

start

register request

XOR

AND

e1

examine thoroughly

e2

examine casually

e3

check ticket

OR

AND

e4

decide

e5

pay compensation

end

intermediate event

end

start event

end event

AND-split connector

AND-join connector

XOR-split connector

XOR-join connector

OR-split connector

OR-join connector

function

intermediate event

begin
Vicious cycle paradox

- If one blocks, both should block (due to symmetry).
- If one is not blocked, both can potentially progress (due to symmetry).
- If both block, there will never be a second token. Hence, the choice to block was wrong.
- If both progress, there will be a second token. Hence, the choice to progress was wrong.
- Paradox: all choices are wrong.
Causal nets (C-nets)

- **a**: register request
- **b**: examine thoroughly
- **c**: examine casually
- **d**: check ticket
- **e**: decide
- **f**: reinitiate request
- **g**: pay compensation
- **h**: reject request
- **z**: end

**Additional Diagrams:**
- **XOR-split**
- **AND-split**
- **OR-split**
- **XOR-join**
- **AND-join**
- **OR-join**
Why C-nets?

- Similar to heuristic nets and representation used by genetic miners.
- Fits well with mainstream languages (BPMN, EPCs, YAWL, BPEL, etc.).
- Model XOR, AND, and OR, but no silent steps or duplicate activities needed.
- Loose interpretation (focus on replay semantics rather than execution semantics, i.e., moment of choice is not fixed).
C-nets
C-nets are used for heuristics mining

- Learn a dependency graph by counting frequencies
- Learn splits and joins
- Visualize (and convert if needed)

Settings, e.g., thresholds
Causal nets (C-nets): Notation

- XOR-split
- AND-split
- OR-split

- XOR-join
- AND-join
- OR-join
Example

- a: register request
- b: examine thoroughly
- c: examine casually
- d: check ticket
- e: decide
- f: reinitiate request
- g: pay compensation
- h: reject request
- z: end
Initially activity \( a \) can occur with an output binding enabling \( b \) and \( d \) or \( c \) and \( d \). 

\( a \) occurred with output binding \( \{b,d\} \).
Activity $b$ occurs removing obligation $(a,b)$ and creating obligation $(b,e)$.
Activity $d$ occurs removing obligation $(a,d)$ and creating obligation $(d,e)$
Activity e occurs removing obligations \((b,e)\) and \((d,e)\) and creating obligation \((e,g)\)
Activity $g$ occurs removing obligation $(e,g)$ and creating obligation $(g,z)$.
Activity z occurs removing obligation \((g,z)\) while leaving no other obligations.
Bindings
Valid binding sequences

- Start with start activity without any obligations and end with end activity without any pending obligations.
- Input bindings remove existing obligations and output bindings create obligations.
Semantics are declarative!

Moment of choice is irrelevant, i.e., bindings can be “reconsidered” until the end.

Only valid binding sequences are considered !!!
Relating WF-nets to C-nets

full firing sequence of sound WF-net
valid binding sequence of C-net
Relating C-nets to WF-nets

Typically the WF-net is not sound!!

WF-net may deadlock, leave tokens behind, etc.

Valid binding sequence of C-net  Valid firing sequence of WF-net
WF-net interpretation of C-nets
(only valid sequences!)

inability to terminate properly for non-conforming traces
C-nets are more expressive (due to declarative semantics)

There is no Petri net that has a set of full firing sequences corresponding to the valid binding sequences of this C-net.
Support in ProM 6

- C-nets can be discovered using heuristic miner and genetic miner (and other ones through conversion).
- Verification of C-nets (check relaxed soundness on corresponding WF-net).
- Replay functionality for C-nets: performance analysis, conformance checking.
heuristics mining approach
Approach

1. **Learn a dependency graph by counting frequencies**
   - **L**: event log
   - **Dependency graph**

2. **Learn splits and joins**
   - **C-net and frequencies**

3. **Visualize (and convert if needed)**
   - BPMN workflow nets
   - UML
   - EPC

4. **Settings**, e.g., thresholds
Running example

$L = [\langle a, e \rangle^5, \langle a, b, c, e \rangle^{10}, \langle a, c, b, e \rangle^{10}, \langle a, b, e \rangle^1, \langle a, c, e \rangle^1, \langle a, d, e \rangle^{10}, \langle a, d, d, e \rangle^2, \langle a, d, d, d, e \rangle^1]$
Example log: problems of the $\alpha$ algorithm

$L = [\langle a, e \rangle^5, \langle a, b, c, e \rangle^{10}, \langle a, c, b, e \rangle^{10}, \langle a, b, e \rangle^1, \langle a, c, e \rangle^1, \langle a, d, e \rangle^{10}, \langle a, d, d, e \rangle^2, \langle a, d, d, d, e \rangle^1]$
Taking into account frequencies

\[ L = [\langle a, e \rangle^5, \langle a, b, c, e \rangle^{10}, \langle a, c, b, e \rangle^{10}, \langle a, b, e \rangle^1, \langle a, c, e \rangle^1, \langle a, d, e \rangle^{10}, \langle a, d, d, e \rangle^2, \langle a, d, d, d, e \rangle^1] \]

\[ |a >_L b| = \sum_{\sigma \in L} L(\sigma) \times |\{1 \leq i < |\sigma| \mid \sigma(i) = a \land \sigma(i+1) = b\}| \]

<table>
<thead>
<tr>
<th>&gt;L</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Information loss when frequencies are ignored
Dependency measure: taking into account concurrency

\[ |a >_L b| = \sum_{\sigma \in L} L(\sigma) \times |\{1 \leq i < |\sigma| \mid \sigma(i) = a \land \sigma(i + 1) = b\}| \]

**dependency measure**

\[ |a \Rightarrow_L b| \text{ is the value of the dependency relation between } a \text{ and } b: \]

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

\[ |a \Rightarrow_L b| = 10 \]
\[ |a \Leftrightarrow_L b| = 10/11 \]
\[ |b \Rightarrow_L a| = 0 \]
\[ |b \Rightarrow_L a| = -10/11 \]

\[ |a \Rightarrow_L b| \] is the value of the dependency relation between \( a \) and \( b \):

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

Both \( (>_L \text{ and } \Rightarrow_L) \) need to be above a threshold to be approved.
XOR-split pattern

|a > L b| = 8
|a ⇒ L b| = 8/9
|a > L c| = 2
|a ⇒ L c| = 2/3
|b > L a| = 0
|b ⇒ L a| = -8/9
|b > L c| = 0
|b ⇒ L c| = 0/1
|c > L a| = 0
|c ⇒ L a| = -2/3
|c > L b| = 0
|c ⇒ L b| = 0/1

|a ⇒ L b| is the value of the dependency relation between a and b:

|a ⇒ L b| = \begin{cases} 
|a > L b| - |b > L a| & \text{if } a \neq b \\
|a > L b| + |b > L a| + 1 & \text{if } a = b
\end{cases}
XOR-join pattern

|a \geq_L b| = 0
|a \Rightarrow_L b| = 0/1
|a \geq_L c| = 8
|a \Rightarrow_L c| = 8/9
|b \geq_L a| = 0
|b \Rightarrow_L a| = 0/1
|b \geq_L c| = 2
|b \Rightarrow_L c| = 2/3
|c \geq_L a| = 0
|c \Rightarrow_L a| = -8/9
|c \geq_L b| = 0
|c \Rightarrow_L b| = -2/3

$a \Rightarrow_L a$ is the value of the dependency relation between $a$ and $b$:

$$|a \Rightarrow_L b| = \begin{cases} |a \geq_L b| - |b \geq_L a| & \text{if } a \neq b \\ |a \geq_L b| + |b \geq_L a| + 1 & \text{if } a = b \end{cases}$$
AND-split pattern

illustrates why $>_{\downarrow}$ is not enough and $\Rightarrow_{\downarrow}$ is needed

<table>
<thead>
<tr>
<th>$a &gt;_{\downarrow} b$</th>
<th>$= 5$</th>
<th>$b &gt;_{\downarrow} a$</th>
<th>$= 0$</th>
<th>$c &gt;_{\downarrow} a$</th>
<th>$= 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \Rightarrow_{\downarrow} b$</td>
<td>$= 5/6$</td>
<td>$b \Rightarrow_{\downarrow} a$</td>
<td>$= -5/6$</td>
<td>$c \Rightarrow_{\downarrow} a$</td>
<td>$= -5/6$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$a &gt;_{\downarrow} c$</th>
<th>$= 5$</th>
<th>$b &gt;_{\downarrow} c$</th>
<th>$= 5$</th>
<th>$c &gt;_{\downarrow} b$</th>
<th>$= 5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \Rightarrow_{\downarrow} c$</td>
<td>$= 5/6$</td>
<td>$b \Rightarrow_{\downarrow} c$</td>
<td>$= 0/11$</td>
<td>$c \Rightarrow_{\downarrow} b$</td>
<td>$= 0/11$</td>
</tr>
</tbody>
</table>

$a \Rightarrow_{\downarrow} b$ is the value of the dependency relation between $a$ and $b$:

$$a \Rightarrow_{\downarrow} b = \begin{cases} 
|a >_{\downarrow} b| - |b >_{\downarrow} a| & \text{if } a \neq b \\
|a >_{\downarrow} b| + |b >_{\downarrow} a| + 1 & \text{if } a = b
\end{cases}$$
AND-join pattern

\[ |a \Rightarrow_L b| = 5 \]
\[ |a \Rightarrow_L c| = 5/6 \]
\[ |a \Rightarrow_L b| = 0/11 \]
\[ |a \Rightarrow_L c| = 5/6 \]

\[ |b \Rightarrow_L a| = 5 \]
\[ |b \Rightarrow_L c| = 5/6 \]
\[ |b \Rightarrow_L a| = 0/11 \]
\[ |b \Rightarrow_L c| = 5/6 \]

\[ |c \Rightarrow_L a| = 0 \]
\[ |c \Rightarrow_L b| = 0 \]
\[ |c \Rightarrow_L a| = -5/6 \]
\[ |c \Rightarrow_L b| = 5/6 \]

\[ |a >_L b| = 5 \]
\[ |b >_L a| = 5 \]
\[ |c >_L a| = 0 \]

\[ |a >_L c| = 5 \]
\[ |b >_L c| = 5 \]

|a ⇒_L b| is the value of the dependency relation between a and b:

\[ |a ⇒_L b| = \begin{cases} |a >_L b| - |b >_L a| & \text{if } a \neq b \\ |a >_L b| + |b >_L a| + 1 & \text{if } a = b \end{cases} \]
Loop pattern

\[ |a \Rightarrow_L b| \] is the value of the dependency relation between \( a \) and \( b \):

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

illustrates why self loops are handled differently (otherwise 0)

\[ |a >_L a| = 0 \quad |b >_L b| = 3 \quad |c >_L c| = 0 \]
\[ |a \Rightarrow_L a| = 0/1 \quad |b \Rightarrow_L b| = -7/8 \quad |c \Rightarrow_L c| = -3/4 \]
\[ |a >_L b| = 7 \quad |b >_L b| = 3 \quad |c >_L b| = 0 \]
\[ |a \Rightarrow_L b| = 7/8 \quad |b \Rightarrow_L b| = 3/4 \quad |c \Rightarrow_L b| = -7/8 \]
\[ |a >_L c| = 3 \quad |b >_L c| = 7 \quad |c >_L c| = 0 \]
\[ |a \Rightarrow_L c| = 3/4 \quad |b \Rightarrow_L c| = 7/8 \quad |c \Rightarrow_L c| = 0/1 \]

assume
ac: 3 times
abc: 4 times
abbc: 3 times
Example revisited

\[ L = [\langle a, e \rangle^5, \langle a, b, c, e \rangle^{10}, \langle a, c, b, e \rangle^{10}, \langle a, b, e \rangle^1, \langle a, c, e \rangle^1, \\
\langle a, d, e \rangle^{10}, \langle a, d, d, e \rangle^2, \langle a, d, d, d, e \rangle^1] \]

<table>
<thead>
<tr>
<th>(&gt; L)</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>(b)</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>(c)</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>(d)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>(e)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Dependency measures computed for example

<table>
<thead>
<tr>
<th>( \Rightarrow_L )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( d )</th>
<th>( e )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( \frac{0}{0+1} = 0 )</td>
<td>( \frac{11-0}{11+0+1} = 0.92 )</td>
<td>( \frac{11-0}{11+0+1} = 0.92 )</td>
<td>( \frac{13-0}{13+0+1} = 0.93 )</td>
<td>( \frac{5-0}{5+0+1} = 0.83 )</td>
</tr>
<tr>
<td>( b )</td>
<td>( \frac{0-11}{0+11+1} = -0.92 )</td>
<td>( \frac{0}{0+1} = 0 )</td>
<td>( \frac{10-10}{10+10+1} = 0 )</td>
<td>( \frac{0-0}{0+0+1} = 0 )</td>
<td>( \frac{11-0}{11+0+1} = 0.92 )</td>
</tr>
<tr>
<td>( c )</td>
<td>( \frac{0-11}{0+11+1} = -0.92 )</td>
<td>( \frac{10-10}{10+10+1} = 0 )</td>
<td>( \frac{0}{0+1} = 0 )</td>
<td>( \frac{0-0}{0+0+1} = 0 )</td>
<td>( \frac{11-0}{11+0+1} = 0.92 )</td>
</tr>
<tr>
<td>( d )</td>
<td>( \frac{0-13}{0+13+1} = -0.93 )</td>
<td>( \frac{0-0}{0+0+1} = 0 )</td>
<td>( \frac{0-0}{0+0+1} = 0 )</td>
<td>( \frac{4}{4+1} = 0.80 )</td>
<td>( \frac{13-0}{13+0+1} = 0.93 )</td>
</tr>
<tr>
<td>( e )</td>
<td>( \frac{0-5}{0+5+1} = -0.83 )</td>
<td>( \frac{0-11}{0+11+1} = -0.92 )</td>
<td>( \frac{0-11}{0+11+1} = -0.92 )</td>
<td>( \frac{0-13}{0+13+1} = -0.93 )</td>
<td>( \frac{0}{0+1} = 0 )</td>
</tr>
</tbody>
</table>

\( |a \Rightarrow_L b| \) is the value of the dependency relation between \( a \) and \( b \):

\[
|a \Rightarrow_L b| = \frac{|a >_L b| - |b >_L a|}{|a >_L b| + |b >_L a| + 1}
\]

if \( a \neq b \)

\[
|a \Rightarrow_L b| = \frac{|a >_L b|}{|a >_L a| + 1}
\]

if \( a = b \)
Dependency graph using a lower threshold
(at least 2 direct successions and a dependency of at least 0.7)

- **Nodes:** a, b, c, d, e
- **Edges:**
  - a → b (11(0.92))
  - a → c (11(0.92))
  - a → d (11(0.92))
  - b → c (11(0.92))
  - c → e (11(0.92))
  - c → d (11(0.92))
  - d → e (11(0.92))
  - e → a (13(0.93))
  - e → d (13(0.93))

**Table:**

<table>
<thead>
<tr>
<th>( \Rightarrow ) ( &gt;_{L} )</th>
<th>a 0</th>
<th>b 11</th>
<th>c 11</th>
<th>d 13</th>
<th>e 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Dependency Matrix:**

- a → b: \( 0 / 0 + 1 = 0 \)
- a → c: \( 11 / 10 + 1 = 0.92 \)
- a → d: \( 11 / 10 + 1 = 0.92 \)
- a → e: \( 13 / 10 + 1 = 0.93 \)
- b → c: \( 11 / 10 + 1 = 0.92 \)
- b → d: \( 11 / 10 + 1 = 0.92 \)
- b → e: \( 13 / 10 + 1 = 0.93 \)
- c → d: \( 11 / 10 + 1 = 0.92 \)
- c → e: \( 13 / 10 + 1 = 0.93 \)
- d → e: \( 11 / 10 + 1 = 0.92 \)

<table>
<thead>
<tr>
<th>( \Rightarrow ) ( &gt;_{L} )</th>
<th>a 0</th>
<th>b 11</th>
<th>c 11</th>
<th>d 13</th>
<th>e 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Dependency graph using a higher threshold
(at least 5 direct successions and a dependency of at least 0.9)

<table>
<thead>
<tr>
<th>&gt;L</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>11</td>
<td>11</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Next step: learn splits and joins

1. Learn a dependency graph by counting frequencies
2. Learn splits and joins
3. Visualize (and convert if needed)

Diagram:
- Event log L
- Dependency graph
- C-net and frequencies
- BPMN
- UML
- EPC

Settings, e.g., thresholds
Learning splits and joins: example output

\[ L = [\langle a, e \rangle^5, \langle a, b, c, e \rangle^{10}, \langle a, c, b, e \rangle^{10}, \langle a, b, e \rangle^1, \langle a, c, e \rangle^1, \langle a, d, e \rangle^{10}, \langle a, d, d, e \rangle^2, \langle a, d, d, d, e \rangle^1] \]
There are various approaches to learn splits and joins

- The heuristic miner counts how often sets of inputs and outputs appear, i.e., given windows before and after the activity (many variants possible) it counts which sets of activities appear before and after and picks one.
  - This way one can count the frequency of bindings and select the most frequent ones.
  - Heuristic miner does not consider end-to-end traces, i.e., it is only a best guess based on local information.
Alternative approaches

• All approaches are based on the idea that ideally a trace can be replayed from the initial state to the final state.
• This can be checked precisely using various replay approaches (will be discussed later).
• An activity "a" with two input arcs and two output arcs has 3 possible input and output bindings. Each input/output arc needs to be involved in at least one binding, so there are \(|\{\{i1\},\{i2\}\}, \{\{i1\},\{i1,i2\}\}, \{\{i2\},\{i1,i2\}\}, \{\{i1\},\{i2\},\{i1,i2\}\}\)| x \(|\{\{o1\},\{o2\}\}, \{\{o1\},\{o1,o2\}\}, \{\{o2\},\{o1,o2\}\}, \{\{o1\},\{o2\},\{o1,o2\}\}\}| = 4 x 4 = 16 possible "a" activities.
• Hence, one can use approaches that simply "try bindings": exhaustively, randomly, or evolutionary (see genetic miners)
Visualization using frequencies

Originally only in ProM's fuzzy miner, now in almost all commercial process mining products.

\[ L = [\langle a, e \rangle^5, \langle a, b, c, e \rangle^{10}, \langle a, c, b, e \rangle^{10}, \langle a, b, e \rangle^1, \langle a, c, e \rangle^1, \langle a, d, e \rangle^{10}, \langle a, d, d, e \rangle^2, \langle a, d, d, d, e \rangle^1] \]
ProM's heuristic miner (multiple variants)

\[ L = \langle a, e \rangle^5, \langle a, b, c, e \rangle^{10}, \langle a, c, e \rangle^5, \langle a, d, e \rangle^5, \langle a, d, d, e \rangle^2, \langle a, c, d, e \rangle^2, \langle a, c, d, c, e \rangle \]
Standard settings
Lower thresholds

Heuristics Miner Parameters

Thresholds
- Relative-to-best: 5
- Dependency: 40
- Length-one-loops: 40
- Length-two-loops: 90
- Long distance: 90

Options
- All tasks connected:
- Long distance dependencies:
- Unique start and end tasks:
Different visualizations
Corresponding Petri net

Same or not?
No, also the two unique traces are allowed (depends on thresholds)

\[ L = [\langle a, e \rangle^5, \langle a, b, c, e \rangle^{10}, \langle a, c, b, e \rangle^{10}, \langle a, b, e \rangle^1, \langle a, c, e \rangle^1, \langle a, d, e \rangle^{10}, \langle a, d, d, e \rangle^2, \langle a, d, d, d, e \rangle^1] \]
Summary: characteristics of heuristic mining

- Can deal with noise and therefore quite robust.
- Improved representational bias.
- Split and join rules are only considered locally (therefore many discovered model are not sound and require repair actions).
two phase approach
Overview of the approach

1. Learn a transition system using a state abstraction.
2. Transform the transition systems into an equivalent Petri net.
3. Visualize (and convert if needed).

Settings, e.g., thresholds.
Example: Process Discovery Using State-Based Regions

Event log:

```
01011001101101001
0111110110100011
0110011110111000
0110110100110001
011011010011001
```

Diagram:

- Start state: `a`
- Transition: `a` to `b`
- Transition: `b` to `[a,b]`
- Transition: `[a,b]` to `[a,b,c]`
- Transition: `[a,b,c]` to `[a,b,c,d]`
- Transition: `[a,b,c,d]` to `d`
- Transition: `d` to `end`

- Transition: `a` to `c`
- Transition: `c` to `[a,c]`
- Transition: `[a,c]` to `[a,e]`
- Transition: `[a,e]` to `[a,d,e]`

- Transition: `b` to `p1`
- Transition: `p1` to `b`
- Transition: `e` to `p3`
- Transition: `p3` to `d`
- Transition: `c` to `p2`
- Transition: `p2` to `c`
Example of State-Based Region

- enter: b,e
- leave: d
- do-not-cross: a,c
constructing transition systems from event logs
Learning a Transition System

- past, future, past+future
- sequence, multiset, set abstraction
- limited horizon to abstract further
- filtering e.g. based on transaction type, names, etc.
- labels based on activity name or other features

trace: a b c d c d c d e f a g h h h i

current state

downward arrow pointing to:

past

past and future

future
Past Without Abstraction (Full Sequence)

Sometimes called the "prefix automaton"

\[ L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle] \]
Future Without Abstraction

$L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle]$
Past with Multiset Abstraction

\[ L_1 = \langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle \]
Only Last Event Matters For State

\[ L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle] \]
Only Next Event Matters For State

\[ L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle] \]
Only Last Event Matters

\[ L_2 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle] \]
Only Set of Last Two Events Matters

\[ L_2 = \left[ \langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^4, \langle a, b, c, e, f, b, c, d \rangle^2, \langle a, b, c, e, f, c, b, d \rangle, \langle a, c, b, e, f, b, c, d \rangle^2, \langle a, c, b, e, f, b, c, e, f, c, b, d \rangle \right] \]
Using ProM
Inspect Event Log

\[ L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle] \]
Inspect Traces
Run Plugin
Select (scroll or by name)
Start Plugin "Mine Transition System"
Start Window

**past**

**future**

**attributes**
Abstraction

list, multiset, or set

all, or only last k events
Which events to filter?
Which labels need to be visible?
Any repair actions?

- Remove self loops
- Improve diamond structure
- Merge states with identical inflow

Diagrams showing:
- Remove self loops
- Improve diamond structure
- Merge states with identical inflow
Check configuration

![ProM UTopia window with TS Miner and Check configuration panel]

- Key classifiers:
  - Event Name backwards
    - a
    - b
    - c
    - d
    - e
- Collection type:
  - List
- Collection size:
  - No limit
- Transition label filter:
  - e=complete
  - e=complete
  - e=complete
  - e=complete
- Post-mining conversions:
  - None

[Arrow pointing to the Finish button]
Resulting transition system
Convert transition system to Petri net **Using Regions**
Resulting Petri net
$L_1 = [\langle a, b, c, d \rangle^3, \langle a, c, b, d \rangle^2, \langle a, e, d \rangle]$
state-based regions
What is a (state-based) region?

- a = enter
- b = enter
- c = exit
- d = exit
- e = do not cross
- f = do not cross
Starting point: A Transition System

- We assume that there is only one initial state (otherwise preprocessing needed).
- It is convenient to also have just one final state that can always be reached (not strictly necessary)
- All states need to be reachable!
Definition

- A **region** $r$ is a set of states, such that for all transitions $(s_0, e, s_0')$, $(s_1, e, s_1')$ in the transition system holds that:
  1) $s_0 \in r$ and $s_0' \notin r$ implies that $s_1 \in r$ and $s_1' \notin r$
  2) $s_0 \notin r$ and $s_0' \in r$ implies that $s_1 \notin r$ and $s_1' \in r$

- In words: A region is a set of states, such that, if a transition **exits** the region, then all equally labeled transitions **exit** the region, and if a transition **enters** the region, then all equally labeled transitions **enter** the region. All events not entering or exiting the region do **not cross** the region.
Example of a region

- a enters
- b exits
- c does not cross
- d does not cross
- e does not cross
Example of a region

- a does not cross
- b enters
- c does not cross
- d does not cross
- e exits
Example of a region

- a exits
- b does not cross
- c does not cross
- d does not cross
- e does not cross

Places corresponding to regions containing the initial state are initially marked.
Example of a region

- a enters
- b does not cross
- c does not cross
- d exits
- e does not cross
• **a** enters
• **b** does not cross and exits
• **c** does not cross and exits
• **d** does not cross and exits
• **e** does not cross
• a does not cross
• b does not cross and exits
• c does not cross and exits
• d does not cross and exits
• e does not cross
Multiple regions

Etc.
Selectively chosen regions ...
Problem: many regions

How many regions does the reachability graph of this Petri net have?

At most 1 transition label can be crossing (or none). So there are \((n+1)\) cases. Each case splits the set of states in two, so \(2(n+1)\) regions. (not too bad …)
Problem: many regions

How many regions does the reachability graph of this Petri net have?

Every subset of states forms a region. Hence, there are $2^{(n+1)}$ regions.
Problem: many regions

For a sequence of 10 activities we have $2^{(10+1)} = 2048$ regions.

Idea: let us just consider minimal non-trivial regions!

These extra regions do not influence the behavior.
Regions – Region Properties

• Let \( S \) be the set of all states of a transition system.

• Trivial Regions: Both \( S \) and \( \emptyset \) are called the trivial regions,

• Complements: If \( r \) is a region, then \( S \setminus r \) is a region,

• Pre-/Post-regions: If event \( e \) exits (enters) a region \( r \), then \( r \) is a pre- (post-)region of \( e \),

• Minimal regions: If \( r_0 \) and \( r_1 \) are regions, and \( r_0 \) is a subset of \( r_1 \), then \( r_1 \setminus r_0 \) is a region.

• The latter implies the existence of (non-trivial) *minimal regions*. 
Trivial regions: \( S \) and \( \emptyset \) are trivial regions

\[s1 \rightarrow a\]
\[s2 \rightarrow c, d\]
\[s3 \rightarrow d\]
\[s4 \rightarrow d\]
\[s5 \rightarrow a\]
\[s6 \rightarrow c\]
\[s7 \rightarrow b\]
\[s8 \rightarrow b\]
\[s9 \rightarrow e\]
\[s10 \rightarrow e\]

\[s1 \rightarrow a\]
\[s2 \rightarrow c, d\]
\[s3 \rightarrow d\]
\[s4 \rightarrow d\]
\[s5 \rightarrow a\]
\[s6 \rightarrow c\]
\[s7 \rightarrow b\]
\[s8 \rightarrow b\]
\[s9 \rightarrow e\]
\[s10 \rightarrow e\]

\[a, b, c, d, e \text{ do not cross}\]
Complement:
If $r$ is a region, then $S \setminus r$ is a region

"exits" and "enters" are swapped
If \( r_0 \) and \( r_1 \) are regions, and \( r_0 \) is a subset of \( r_1 \), then \( r_1 \setminus r_0 \) is a region.
Not minimal yet …
Example: 8 minimal regions
Pre and post regions

- If event $e$ enters a region $r$, then $r$ is a post-region of $e$.
  - $r$ is post-region of $a$
  - $r$ is post-region of $b$

- If event $e$ exits a region $r$, then $r$ is a pre-region of $e$.
  - $r$ is pre-region of $c$
  - $r$ is pre-region of $d$

- $\text{pre}(e)$ is the set of all (minimal) pre-regions of $e$.
- $\text{pre}(e)$ is the set of all (minimal) pre-regions of $e$.
- Both are sets of sets!
Basic algorithm to construct a Petri net

• For each event in the transition system, a transition is generated in the Petri net.
• Compute the minimal non-trivial regions.
• For each minimal non-trivial in the transition system, a place is generated in the Petri net.
• Add corresponding arcs (post-regions are output places and pre-regions are input places).
• A token is added to each place that corresponds to a region containing the initial state.

The resulting Petri net is called the minimal saturated net.
Applying the algorithm yields the expected result
Load Petri net with 10 parallel activities
Construct reachability graph

![ProM interface](image)
Reachability graph (1+2^{10}+1 = 1026 states)
Apply state-based regions to fold state space
Discovered Petri net

• Petri net is rediscovered!
• Odd example, normally the transition system is constructed from an event log.
40.825 states, 221.618 transitions

26 transitions, 28 places, 1 token
But ....
Consider an event log containing just $<a,a>$ traces.

**Only trivial regions:** $\emptyset$ and $\{s1,s2,s3\}$

**Petri net**

Also allows for:
- a
- aaaa
- aaaaaaaaaa
Consider an event log containing traces \(<a,c>\), \(<a,b,c>\), \(<a,b,b,c>\), \(<a,b,b,b,c>\), …

transition system able to generate log

Regions:
- \(\{s1\}\) (a exits, b and c do not cross)
- \(\{s2\}\) (a enters, b does not cross, c exits)
- \(\{s3\}\) (a and b does not cross, c enters)

Petri net

Also allows for:
- \(bbac\)
- \(acbbbb\)
- \(babcb\)
Consider an event log containing traces \(<a,b>, <b>\)

transition system able to generate log

Regions:
- \{s1,s2\} (a does not cross, b exits)
- \{s3,s4\} (a does not cross, b enters)
- \{s1,s3\} (a exits and b does not cross)
- \{s2,s4\} (a enters and b does not cross)
Petri net

Also allows for trace <b,a>!
All underfitting … but allow for original behavior
Refinement: label splitting

If certain conditions do not hold, then the conflicting labels are temporarily relabeled. As a result bisimilarity (strong equivalence notation) is guaranteed.
Using ProM
(uses label splitting to solve problem)
Using ProM
(addresses self-loop problem)
Using ProM
(uses label splitting to solve problem)

two "b" transitions
The basic algorithm produces a Petri net that can reproduce the traces of the original model (and often more).

A range of state abstractions can be used to create a transition system from an event log.

Many refinements exist (for example label splitting).
conclusion and outlook
Beyond regions and heuristics

- there are many more process discovery techniques
- process discovery is just one of several process mining tasks
- next we focus on the event data
extra assignments
(if regions are not clear)
Exercise 1

• Download event logs from "book page" on www.processmining.org.
• Load L1.xes into ProM.
• Apply Alpha algorithm.
• Build various transition systems using various abstractions (past/future/both, set/multiset/sequence, all/k-last events, etc.).
• Construct a Petri net using state-based regions for at least three transition systems.

\[ L_1 = [\langle a, b, c, d \rangle^3 , \langle a, c, b, d \rangle^2 , \langle a, e, d \rangle] \]
Result Alpha Miner
Odd abstraction

Select collection type
- List
- Multiset
- Set

Select collection size limit
- No limit
- Limit: 

Petri net synthesized from PTS (mined from L1 mxml)
Horizon of 1

Label splitting!!!
Exercise 2

$L_4 = \left[ \langle a, c, d \rangle^{45}, \langle b, c, d \rangle^{42}, \langle a, c, e \rangle^{38}, \langle b, c, e \rangle^{22} \right]$

- Construct two transition systems for event log $L_4$
  - One based on past with no abstractions (prefix automaton)
  - One based on the last event only.
- Compute all minimal non-trivial regions for both transition systems.
- Show the corresponding Petri nets.
- Check results using ProM.
Prefix automaton

\[ L_4 = [\langle a, c, d \rangle^{45}, \langle b, c, d \rangle^{42}, \langle a, c, e \rangle^{38}, \langle b, c, e \rangle^{22}] \]
Regions

```
Regions

©Wil van der Aalst TU/e (use only with permission & acknowledgements)
```
Transition system based on last event only

![Transition system diagram]

\[ L_4 = [\langle a, c, d \rangle^{45}, \langle b, c, d \rangle^{42}, \langle a, c, e \rangle^{38}, \langle b, c, e \rangle^{22}] \]
Regions and Petri net
Past with no abstractions
(note optimization wrt to merging final states)
Based on the last event only
Exercise 3

$L_9 = [\langle a, c, d \rangle^{45}, \langle b, c, e \rangle^{42}]$

• Construct two transition systems for event log $L_9$
  – One based on past with no abstractions (prefix automaton)
  – One based on the last event only.
• Compute all minimal non-trivial regions for both transition systems.
• Show the corresponding Petri nets.
• Check results using ProM.
Prefix automaton

$L_9 = [\langle a, c, d \rangle^{45}, \langle b, c, e \rangle^{42}]$
Regions
Transition system based on last event only

\[ L_9 = \left[ \langle a, c, d \rangle^{45}, \langle b, c, e \rangle^{42} \right] \]
Regions and Petri net

```
s1  \{a\}  \{b\}  
s3        c  
\{\}  b

s2  \{a\}  \{c\}  
s5                \{d\}  
s8                \{e\}

\begin{align*}
& a \\
& b \\
& c
\end{align*}

\begin{align*}
& d \\
& e \\
& d
\end{align*}
```
Past with no abstractions
(note optimization wrt to merging final states)
Based on the last event only
Exercise 4

\[ L_6 = \left[ \langle a,c,e,g \rangle^2, \langle a,e,c,g \rangle^3, \langle b,d,f,g \rangle^2, \langle b,f,d,g \rangle^4 \right] \]

- Construct a transition system based on past with no abstractions (prefix automaton) for event log \( L_6 \).
- Compute all minimal non-trivial regions.
- Show the corresponding Petri net.
- Check result using ProM (also explore other abstractions).
\[ L_6 = [\langle a,c,e,g \rangle^2, \langle a,e,c,g \rangle^3, \langle b,d,f,g \rangle^2, \langle b,f,d,g \rangle^4] \]
Some regions
Additional non-trivial minimal regions
Petri net

Note redundant places!
Even more minimal regions?

- These would be added when using the basic algorithm.
- However, they can be discarded because they have no exiting arcs and are therefore not limiting the behavior.
- ProM does not show these places.
Prefix automaton
Set abstraction
Last event only
(label splitting!)