Conformance Checking

What are we really doing?

dr.ir. B.F. van Dongen
Assistant Professor
Eindhoven University of Technology
b.f.v.dongen@tue.nl
Process Mining

- Discovering processes
  - How do people behave?

- Compliance oriented
  - Where and why do people deviate from standards / rules / regulations?

- Performance oriented
  - Where are bottlenecks in my processes?
Conformance checking
Aligning models to Observed Behavior

• Starting point for conformance checking is a process model and a trace in a log

• In the process model, tasks are labeled with activity names

• In the trace, events are labeled with activity names

• Conformance checking:

  What is the most likely execution of the model, corresponding to the trace observed in the log?
Introduction: Alignments

- Alignments are used for conformance checking

- Alignments are computed over a trace and a model:
  - A trace is a (partial) order of activities
  - A model is a labeled Petri net or a labeled Process Tree, labeled with activities

- An alignment explains exactly where deviations occur:
  - A synchronous move mean that an activity is in the log and a corresponding transition was enabled in the model
  - A log move means that no corresponding activity is found in the model
  - A model move means that no corresponding activity appeared in the log
Example

model:

log
Example

model:

```
model:

A --> p2 --> B
A --> p3 --> C
D --> p3 --> C
D --> p5 --> E
B --> p4 --> E
C --> p5 --> E
```

log

calignment
Align: Synchronous Move on A

model:

log

alignment
Align: Synchronous Move on B

model:

log

alignment
Align: Log Move on F

model:

log

alignment
Align: Log Move on B

model:

log

alignment
Align: Model Move on C

model:

log

alignment

ABFBDE

...
Align: Synchronous Move on D

model:

ABFBDE

log

alignment
Align: Model Move on C

model:

```
A --- p1 --- B
    /         \
   /           /
  C --- p3 --- D
     |         |  \
     |         /
  E --- p5 --- p6
```

log:

```
ABFBDE...
```

alignment:

```
A | B | C | D | C
```
Align: Synchronous Move on E

model:

log

alignment
Complete Alignment

Compute efficiently using a cost-based optimization technique.

Model:

```
A  B  C  E
p1  B  p4
p2  p5
p3  C
D
p6
```

Firing sequence:

```
ABFBDE
```

Alignment:

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>F</td>
<td>B</td>
<td>D</td>
</tr>
</tbody>
</table>
```

Log:

```
ABFBDE...
```
Computing Alignments

- To compute alignments, an A-star algorithm is used

```
Initialize PriorityQueue q
While (head(q) is not target t)
    VisitedNode n = head(q)
    For each edge in the graph from node(n) to m
        If m was considered before, or is in the queue
            with lower cost, continue
        If m is in the queue with higher cost, update and
            reposition it
    If m is new,
        compute an estimate for the remaining distance to t
        V = new VisitedNode(m)
        set n a predecessor for v
        add v to the priority queue
Return head(q)
```
Computing Alignments

- The search space is a “product” of the statespace of the model and the trace

- Each node is a combination of a state in the model and the executed events in the trace

- Each arc is a **move on model**, **move on log** or a **synchronous move**

- A heuristic function estimates the remaining distance to the final node (i.e. model in a final state and all events executed)
Alignments: Use

- **Alignments are used in many applications:**
  - Fitness computations (how well does my model fit my log?)
  - Precision computations (how precise is my model given the log?)
  - Performance analysis
  - Data-conformance analysis
  - Model repair
  - ...

- **Genetic process discovery algorithms exist that use alignments as the basis for “fitness” functions**

**Computing alignments should be done efficiently!**
Estimating the remaining distance

- Underestimating the remaining distance is essential for the performance of the A-star algorithm.

1. Estimating “0” is always correct → Dijkstra’s shortest path

2. Using the structure of the model can improve the estimator, but comes at a cost of computation time

3. A hybrid solution works best
Estimators for Petri nets

- The Marking Equation for Petri nets provides an underestimate of the remaining distance in the search graph, since:
  - Every trace in the model is a solution to the marking equation, but
  - Not every solution to the marking equation is a trace in the model

- The form of the equation is the following:

  \[
  \text{Minimize } c \cdot x \\
  \text{Where } A \cdot x = r
  \]

  - \(c\) and \(A\) are the same for all traces
  - \(r\) depends on the current state
  - \(x\) is an integer vector
A Closer Look: Estimation

• If we store the vector $x$, we can derive one solution from the other, since $x$ represents the various moves.
  • if $x_i \geq 1$ and we execute move $i$, then $c \cdot x - c \cdot 1_i$ is optimal and the $x - 1_i$ is a new optimal solution for the problem

Minimize $c \cdot x$
Where $A \cdot x = r - A \cdot 1_i$

• Otherwise, if we execute move $i$, then $c \cdot x - c \cdot 1_i$ is a lower bound for $c \cdot x$

• So, we get the problem:

Minimize $c \cdot x$
Where $A \cdot x = r$
$c \cdot x \geq v$
A Closer Look: Estimation

• Technically, the vector $x$ is an integer vector, since you cannot do half a model move or half a synchronous move

• Hence, our LP (polynomial complexity) is an ILP (exponential complexity)

• Fortunately, the LP provides a lower bound on the ILP (and we need an underestimate!)
  • ILP is more precise, but requires more CPU time
  • LP is less precise, but faster

• A hybrid solution allows 1 second to be spend on the ILP computation, before reverting to the LP relaxed solution (or the given lower bound)
Lazy A-Star: Delaying the estimates

- Since there is a quick lower bound on the estimate, delay the real estimation as long as possible

```
Initialize PriorityQueue q
While (head(q) is not target t)
    VisitedNode n = head(q)
    If n.estimate is delayed
        compute the real estimate and re-queue n
    else
        For each edge in the graph from node(n) to m
            If m was considered before, or is in the queue with lower cost, continue
            If m is in the queue with higher cost, update and reposition it
            If m is new,
                compute a quick estimate for the remaining distance to t
                V = new VisitedNode(m)
                set n a predecessor for v
                add v to the priority queue
    Return head(q)
```
Implementation Performance

Time per trace per model size for different noise levels

- Dijkstra’s estimate
- Hybrid ILP
- LP
- Delayed Hybrid ILP
Implementation Performance

Time per trace per noise level for different model sizes

- Dijkstra’s estimate
- Hybrid ILP
- LP
- Delayed Hybrid ILP
Conclusions

• Conformance checking requires the alignment of observed behavior to modeled behavior

• Computing alignments is a cpu-intensive task

• Alignments can be used for:
  • Fitness computations (how well does my model fit my log?)
  • Precision computations (how precise is my model given the log?)
  • Performance analysis
  • Data-conformance analysis
  • Model repair
  • ...

• All of this is implemented in ProM, for Petri nets and Process Trees (demo this afternoon)