Do Petri Nets Provide the Right Representational Bias for Process Mining?

(short paper)

W.M.P. van der Aalst

Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, The Netherlands.
W.M.P.v.d.Aalst@tue.nl

Abstract. Process discovery is probably the most challenging process mining task. Given an event log, i.e., a set of example traces, it is difficult to automatically construct a process model explaining the behavior seen in the log. Many process discovery techniques use Petri nets as a language to describe the discovered model. This implies that the search space—often referred to as the *representational bias*—includes many inconsistent models (e.g., models with deadlocks and livelocks). Moreover, the low-level nature of Petri nets does not help in finding a proper balance between overfitting and underfitting. Therefore, we advocate a new representation more suitable for process discovery: *causal nets*. Causal nets are related to the representations used by several process discovery techniques (e.g., heuristic mining, fuzzy mining, and genetic mining). However, unlike existing approaches, C-nets use declarative semantics tailored towards process mining.

1 Challenges in Process Mining

Process mining is an emerging research area combining techniques from process modeling, model-based analysis, data mining, and machine learning. The goal is to extract knowledge about processes from event data stored in databases, transaction logs, message logs, etc. Process mining techniques are commonly classified into: (a) *discovery*, (b) *conformance*, and (c) *enhancement* [2]. In this paper, we restrict ourselves to control-flow discovery, i.e., learning a process model based on example traces.

A *trace* is a *sequence of events* for a particular *process instance* (also referred to as *case*). Events refer to some *activity*. For example, the trace $\langle a, b, c, d\rangle$ refers to a process execution starting with activity a and ending with activity d. An *event log* is a multiset of traces, e.g., $L = \{\langle a, b, c, d\rangle^{25}, \langle a, c, b, d\rangle^{35}, \langle a, e, d\rangle^{30}\}$ describes the execution sequences of 90 cases. There are dozens of process discovery techniques that are able to construct a process model from such an event log. Many of these techniques use Petri nets as a target representation [4,5,7,12,19,22,23]. Given event log L, these techniques have no problems discovering the Petri net in which, after a, there is a choice between doing b and

c concurrently or just e, followed by d. Note that this example is misleadingly simple as process discovery based on real-life event logs is extremely challenging.

Generally, we use four main quality dimensions for judging the quality of the discovered process model: *fitness* (the model should allow for the behavior observed), *simplicity* (the model should be as simple as possible), *precision* (the model should not allow for behavior that is very unlikely given the event log), and *generalization* (the model should not just represent the observed examples and also allow for behavior not yet observed but very similar to earlier behavior).

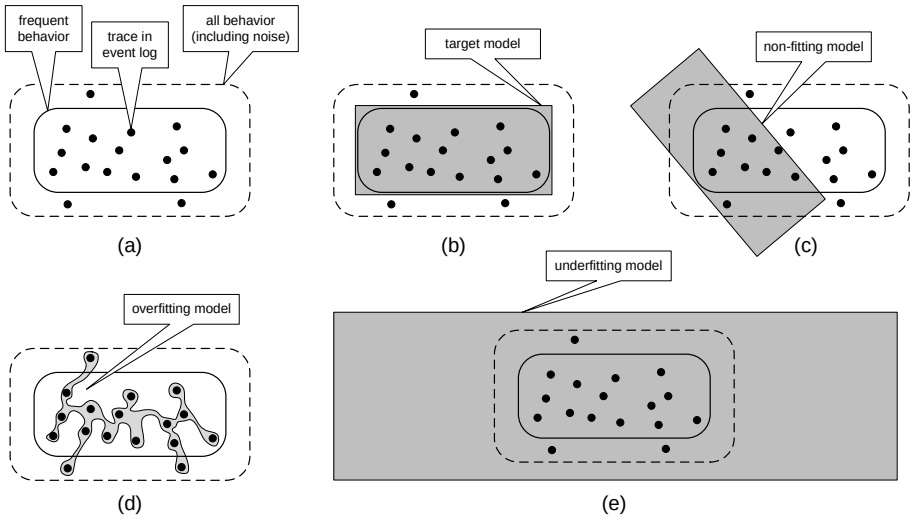

Fig. 1. Illustrating typical problems encountered when discovering process models from event logs: (c) a non-fitting model, (d) an overfitting model (poor generalization), and (e) an underfitting model (poor precision)

The simplicity dimension refers to *Occam's Razor*; the simplest model that can explain the behavior seen in the log, is the best model. Figure 1 explains some of the main challenges related to the other three quality dimensions. Each black dot represents a trace (i.e., a sequence of activities) corresponding to one or more cases in the event log. (Recall that multiple cases may have the same corresponding trace.) An event log typically contains only a fraction of the possible behavior, i.e., the dots should only be seen as *samples* of a much larger set of possible behaviors. Moreover, one is typically primarily interested in frequent behavior and not in all possible behavior, i.e., one wants to abstract from *noise* (i.e., infrequent or exceptional behavior) and therefore not all dots need to be relevant for the process model to be constructed.

It is interesting to analyze such noisy behaviors. However, when constructing the overall process model, the inclusion of infrequent or exceptional behavior

leads to complex diagrams. Moreover, it is typically impossible to make reliable statements about noisy behavior given a relatively small set of observations. Figure 1(a) distinguishes between frequent behavior (solid rectangle with rounded corners) and all behavior (dashed rectangle), i.e., normal and noisy behavior. The difference between normal and noisy behavior is a matter of definition, e.g., normal behavior could be defined as the 80% most frequently occurring traces.

Let us assume that the two rectangles with rounded corners can be determined by observing the process infinitely long while the process is in steady-state (i.e., no concept drift [9]). Based on these assumptions, Fig. 1 sketches four discovered models depicted by shaded rectangles. These discovered models are based on the example traces in the log, i.e., the black dots. The "ideal process model" (Fig. 1(b)) allows for the behavior coinciding with the frequent behavior seen when the process would be observed ad infinitum. The "non-fitting model" in Fig. 1(c) is unable to characterize the process well as it is not even able to capture the examples in the event log used to learn the model. The "overfitting model" (Fig. 1d)) does not generalize and only says something about the examples in the current event log. New examples will most likely not fit into this model. The "underfitting model" (Fig. 1(e)) lacks precision and allows for behavior that would never be seen if the process would be observed ad infinitum.

Figure 1 illustrates the challenges that process discovery techniques need to address: *How to extract a simple target model that is not underfitting, overfitting, nor non-fitting?*

2 Petri nets as a Representational Bias for Process Mining

One can think of process mining as a search problem with a *search space* defined by the class of process models considered, i.e., the goal is to find a "best" process model in the collection of all permissible models. The observation that the target language defines the search space is often referred to as the *representational bias*.

Many process discovery techniques use Petri nets as a representational bias [4,5,7,12,19,22,23]. Examples of such techniques are the α-algorithm and its variants [5,22], state-based region techniques [4,19], and language-based region techniques [7,23]. Some of these techniques allow for labeled transitions, i.e., there may be invisible/silent steps (τ transitions not leaving a mark in the event log) or multiple transitions with the same label. However, all of the models have a clearly defined initial marking and one or more final markings. In fact, most techniques aim at discovering a so-called *workflow net* (WF-net) [1]. A WF-net has one source place (modeling the start of the process) and one sink place (modeling the end), and all nodes are on a path from source to sink. Ideally, such a discovered WF-net is *sound*. Soundness is a common correctness criterion for WF-nets requiring that from any reachable marking it is possible to reach the final marking (weak termination) and there are no dead transitions (i.e., there are no activities that can never happen).

In the remainder, we assume that the goal is to discover sound WF-nets from event logs. The particular soundness notion used is not very relevant. Moreover, the syntactical requirements imposed on WF-nets may be relaxed. However, a basic assumption of any process discovery algorithm is that all traces in the event log start in some initial state and ideally end in a well-defined end state.

Petri nets allow for a wide variety of analysis techniques and provide a simple, yet powerful, graphical representation. This is the reason why they were chosen as a target language for dozens of process discovery techniques described in literature [4,5,7,12,19,22,23]. Nevertheless, in this paper, we pose the question *"Are Petri nets a suitable representational bias for process discovery?"*.

In our view, there are several problems associated to using Petri nets as a representational bias.

- *The search space is too large (including mostly "incorrect" models).* When randomly generating a Petri net, the model is most likely not sound. The fraction of sound process models is small. As a result, most of the process discovery techniques tend to create incorrect process models. For example, the α-algorithm can generate models that have deadlocks and livelocks. Region-based techniques may also suffer from such problems; they can replay the event log but also exhibit deadlocks and livelocks.
- *Petri nets cannot capture important process patterns in a direct manner.* Process modeling languages used by end-users tend to support higher-level constructs, often referred to as workflow patterns [3]. Examples are the OR-split (Multi-Choice pattern) and OR-join (Synchronizing Merge pattern). Many of these patterns can be expressed in terms of Petri nets, i.e., the higher-level construct is mapped onto a small network. This is no problem for model-based analysis (e.g., verification). However, the discovered process model needs to be interpreted by the end-user. Whereas it is relatively easy to translate higher-level constructs to Petri nets, it is difficult to translate lower-level constructs to languages such as BPMN, EPCs, UML, YAWL, etc.
- *It is difficult to "invent" modeling elements.* If all transitions need to have a unique visible label, then the only task of a process discovery algorithm is to "invent" places. If two transitions can have the same visible label, then the process discovery algorithm may also need to duplicate transitions. If transitions can be silent, e.g., to skip an activity, then the process discovery algorithm needs to "invent" such silent transitions (if needed). Places, duplicate transitions, and silent transitions cannot be coupled directly to observations in the event log. The fact that such modeling elements need to be "invented" makes the search space larger (often infinite) and the relation between event log and model more indirect.
- *The representational bias does not help in finding a proper balance between overfitting and underfitting.* Because of the low-level nature of Petri nets, there are no natural patterns to support generalization. Algorithms tend to overfit or underfit the event log. One of the reasons is that the representational bias does not help in guiding the discovery algorithm towards a desirable model. Note that some of the more advanced region-based algo-

rithms allow for the formulation of additional constraints (e.g., the target model should be free-choice and the number of input and output arcs per node is bounded) [23].

Note that the above problems are not specific for Petri nets. Most of the current representations suffer from a subset of these problems. Consider for example BPMN; the fraction of sound BPMN models is small and the mining algorithm needs to "invent" process fragments consisting of gateways and events to capture behavior adequately. However, compared to Petri nets, BPMN can capture more patterns directly.

3 Towards a Better Representational Bias: Causal Nets

The goal of this paper is *not* to provide a solution for all of the problems induced by using Petri nets as a representational bias for process mining. Instead, we would like to discuss potential notations that provide a *more suitable representational bias*. We do not propose new discovery techniques. Instead, we note that most of the existing process discovery techniques can be modified to support a more refined representational bias.

To trigger this discussion, we advocate a new representation more suitable for process discovery: *causal nets* (C-nets) [2]. On the one hand, C-nets are related to the representations used by several process discovery techniques (e.g., heuristic mining [17,21], fuzzy mining [17], and genetic mining [18]). Moreover, in [6] a similar representation is used for conformance checking. On the other hand, C-nets use declarative semantics not based on a local firing rule. This way a larger fraction of models (if not all) is considered to be correct.

A C-net is a graph where nodes represent *activities* and arcs represent *causal dependencies*. Each activity has a set of possible *input bindings* and a set of possible *output bindings*. Consider, for example, the causal net shown in Fig. 2. Activity a has only an empty input binding as this is the start activity. There are two possible output bindings: $\{b, d\}$ and $\{c, d\}$. This means that a is followed by either b and d, or c and d. Activity e has two possible input bindings ($\{b, d\}$ and $\{c, d\}$) and three possible output bindings ($\{g\}$, $\{h\}$, and $\{f\}$). Hence, e is preceded by either b and d, or c and d, and is succeeded by just g, h or f. Activity z is the end activity having two input bindings and one output binding (the empty binding). This activity has been added to create a unique end point. All executions commence with start activity a and finish with end activity z. Note that unlike, Petri nets, there are no places in the causal net; the routing logic is solely represented by the possible input and output bindings.

Definition 1 (Causal net [2]). *A* Causal net *(C-net) is a tuple* $C = (A, a_i, a_o, D, I, O)$ *where:*

- A *is a finite set of* activities*;*
- $a_i \in A$ *is the* start activity*;*
- $a_o \in A$ *is the* end activity*;*

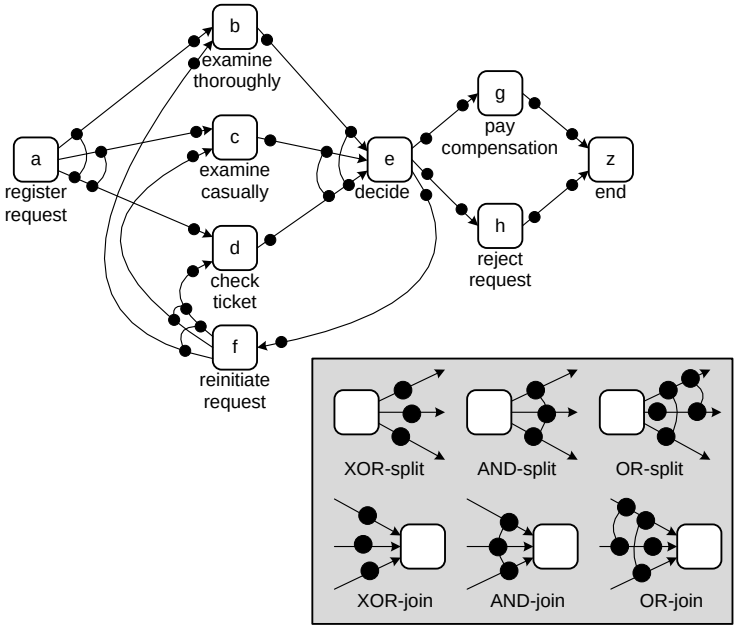

Fig. 2. Example of a C-net and some of the typical input and output bindings present in conventional business process modeling languages [2]

- $D \subseteq A \times A$ *is the* dependency relation,
- $AS = \{X \subseteq \mathcal{P}(A) \mid X = \{\emptyset\} \ \vee \ \emptyset \notin X\}$;[1]
- $I \in A \rightarrow AS$ *defines the set of possible* input bindings *per activity; and*
- $O \in A \rightarrow AS$ *defines the set of possible* output bindings *per activity,*

such that

- $D = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup_{as \in I(a_2)} as\}$;
- $D = \{(a_1, a_2) \in A \times A \mid a_2 \in \bigcup_{as \in O(a_1)} as\}$;
- $\{a_i\} = \{a \in A \mid I(a) = \{\emptyset\}\}$;
- $\{a_o\} = \{a \in A \mid O(a) = \{\emptyset\}\}$; *and*
- *all activities in the graph* (A, D) *are on a path from* a_i *to* a_o.

An *activity binding* is a tuple (a, as^I, as^O) denoting the occurrence of activity a with input binding as^I and output binding as^O. For example, $(e, \{b, d\}, \{f\})$ denotes the occurrence of activity e in Fig. 2 while being preceded by b and d, and succeeded by f. A *binding sequence* σ is a sequence of activity bindings. A possible binding sequence for the C-net of Fig. 2 is $\sigma_{ex} = \langle (a, \emptyset, \{b, d\}), (b, \{a\}, \{e\}), (d, \{a\}, \{e\}), (e, \{b, d\}, \{g\}), (g, \{e\}, \{z\}), (z, \{g\}, \emptyset)\rangle$.

[1] $\mathcal{P}(A) = \{A' \mid A' \subseteq A\}$ is the powerset of A. Hence, elements of AS are *sets of sets* of activities.

A binding sequence is *valid* if a predecessor activity and successor activity always "agree" on their bindings. For a predecessor activity x and successor activity y we need to see the following "pattern": $\langle \ldots, (x, \{\ldots\}, \{y, \ldots\}), \ldots, (y, \{x, \ldots\}, \{\ldots\}), \ldots \rangle$, i.e., the occurrence of activity x with y in its output binding needs to be followed by the occurrence of activity y and the occurrence of activity y with x in its input binding needs to be preceded by the occurrence of activity x. σ_{ex} is an example of a valid sequence.

For technical details regarding these notions we refer to [2]. It is important to note that *the behavior of C-nets is limited to valid binding sequences*. C-nets are not driven by local firing rules (like a Petri net), but by the more declarative notion of valid binding sequences in which activities always "agree" on their bindings.

It can be shown that *C-nets are more expressive than Petri nets*. For any sound WF-net one can construct a C-net such that any full firing sequence of the WF-net corresponds to a valid binding sequence of the C-net and vice versa. Note that at first sight, C-nets seem to be related to *zero-safe nets* [10]. The places in a zero-safe net are partitioned into stable places and zero places. Observable markings only mark stable places, i.e., zero places need to be empty. In-between observable markings zero places may be temporarily marked. In [15] an approach is described to synthesize zero-safe nets. However, zero places cannot be seen as bindings because the "agreement" between two activities may be non-local, i.e., an output binding may create the obligation to execute an activity occurring much later in the process.

For process discovery, the representational bias provided by C-nets is more suitable than the representational bias provided by Petri nets.

- Since the behavior of C-nets is limited to valid binding sequences, any C-net is in principle correct. Therefore, we do not need to consider a search space in which most models are internally inconsistent (deadlocks, etc.)
- C-nets can capture important process patterns in a direct manner. For example, OR-splits (Multi-Choice pattern) and OR-joins (Synchronizing Merge pattern) can be modeled directly. Moreover, there is no need to introduce silent transitions or multiple transitions with the same label to discover a suitable model for event logs such as $L = [\langle a, b, c \rangle^{20}, \langle a, c \rangle^{30}]$. C-nets are closely connected to languages such as BPMN, EPCs, UML, YAWL, etc. However, the interpretation is different as we only consider valid binding sequences. Models may be "cleaned up" as a post optimization.
- There is no need to "invent" modeling elements such as places and silent transitions. We only need to find the set of possible input and output bindings per activity. Note that input and output bindings have a more direct connection to the event log than routing elements encountered in conventional languages such as Petri nets (places and silent/duplicate transitions), BPMN (gateways, events, etc.), and EPCs (connectors and events).
- The representational bias of C-nets is tailored towards finding a proper balance between overfitting and underfitting. It is easy to define the types of input and output bindings that are preferred, e.g., an AND-split or XOR-

split is preferred over an OR-split. For example, it is possible to associate thresholds to extending the set possible bindings.

4 Conclusion

This short paper does *not* aim to provide a new process discovery algorithm. Instead, its purpose is to trigger a discussion on the representational bias used by existing process mining algorithms. We showed that Petri nets are less suitable as a target language. We introduced C-nets as an alternative representational bias. C-nets are able to express behavioral patterns in a more direct manner. Moreover, by limiting the behavior of C-nets to valid binding sequences, we obtain a more suitable search space. We believe that our formalization sheds new light on the representations used in [6,17,18,20,21].

It is interesting to investigate how classical region theory [8,11,13,14,16] can be applied to the synthesis of C-nets. For example, it seems possible to adapt region-based mining approaches as described in [23] to C-nets. However, the straightforward encoding of the synthesis problem into an Integer Linear Programming (ILP) problem makes discovery intractable for realistic examples. Moreover, existing region-based mining techniques have problems dealing with noise and incompleteness. As a result, the discovered models typically do not provide a good balance between overfitting and underfitting.

C-nets are very suitable for genetic process mining [18]. It is also possible to use a mixture of heuristic mining [20,21] and genetic mining [18]. For example, one can first discover the dependency relation D using heuristics and then optimize the input bindings I and output bindings O using genetic algorithms. Genetic operators such as crossover and mutation can be defined on C-nets in a straightforward manner. The fitness function can be based on replay, e.g., the fraction of events and process instances in the log that fit into the model. Here we suggest using the technique described in [6]. Moreover, we also suggest to incorporate the complexity of the model in the fitness function.

Currently, ProM already provides basic support for C-nets (ProM 6 can be downloaded from www.processmining.org). In the future, we aim to add more plug-ins working directly on C-nets.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.
3. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
4. W.M.P. van der Aalst, V. Rubin, H.M.W. Verbeek, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010.

5. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
6. A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Towards Robust Conformance Checking. In M. zur Muehlen and J. Su, editors, *BPM 2010 Workshops, Proceedings of the Sixth Workshop on Business Process Intelligence (BPI2010)*, volume 66 of *Lecture Notes in Business Information Processing*, pages 122–133. Springer-Verlag, Berlin, 2011.
7. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007.
8. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Synthesis of Petri Nets from Finite Partial Languages. *Fundamenta Informaticae*, 88(4):437–468, 2008.
9. R.P. Jagadeesh Chandra Bose, W.M.P. van der Aalst, I.Zliobaite, and M. Pechenizkiy. Handling Concept Drift in Process Mining. In H. Mouratidis and C. Rolland, editors, *International Conference on Advanced Information Systems Engineering (Caise 2011)*, volume 6741 of *Lecture Notes in Computer Science*, pages 391–405. Springer-Verlag, Berlin, 2011.
10. R. Bruni and U. Montanari. Zero-Safe Nets: Comparing the Collective and Individual Token Approaches. *Information and Computation*, 156(1-2):46–89, 2000.
11. M.P. Cabasino, A. Giua, and C. Seatzu. Identification of Petri Nets from Knowledge of Their Language. *Discrete Event Dynamic Systems*, 17(4):447–474, 2007.
12. J. Carmona and J. Cortadella. Process Mining Meets Abstract Interpretation. In J.L. Balcazar, editor, *ECML/PKDD 210*, volume 6321 of *Lecture Notes in Artificial Intelligence*, pages 184–199. Springer-Verlag, Berlin, 2010.
13. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.
14. P. Darondeau. Unbounded Petri Net Synthesis. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 413–438. Springer-Verlag, Berlin, 2004.
15. P. Darondeau. On the Synthesis of Zero-Safe Nets. In *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 364–378. Springer-Verlag, Berlin, 2008.
16. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368, 1989.
17. C.W. Günther and W.M.P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer-Verlag, Berlin, 2007.
18. A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.
19. M. Sole and J. Carmona. Process Mining from a Basis of Regions. In J. Lilius and W. Penczek, editors, *Applications and Theory of Petri Nets 2010*, volume 6128 of *Lecture Notes in Computer Science*, pages 226–245. Springer-Verlag, Berlin, 2010.

20. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.
21. A.J.M.M. Weijters and J.T.S. Ribeiro. Flexible Heuristics Miner (FHM). BETA Working Paper Series, WP 334, Eindhoven University of Technology, Eindhoven, 2010.
22. L. Wen, W.M.P. van der Aalst, J. Wang, and J. Sun. Mining Process Models with Non-Free-Choice Constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180, 2007.
23. J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 94:387–412, 2010.