# On the Representational Bias in Process Mining

W.M.P. van der Aalst
Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
Email: w.m.p.v.d.aalst@tue.nl, WWW: vdaalst.com

***Abstract*—Process mining serves a bridge between data mining and business process modeling. The goal is to extract process-related knowledge from event data stored in information systems. One of the most challenging process mining tasks is *process discovery*, i.e., the automatic construction of process models from raw event logs. Today there are dozens of process discovery techniques generating process models using different notations (Petri nets, EPCs, BPMN, heuristic nets, etc.). This paper focuses on the *representational bias* used by these techniques. We will show that the choice of target model is very important for the discovery process itself. The representational bias should *not* be driven by the desired graphical representation but by the characteristics of the underlying processes and process discovery techniques. Therefore, we analyze the role of the representational bias in process mining.**

## I. Introduction

Process mining is an emerging discipline providing comprehensive sets of tools to provide fact-based insights and to support process improvements [1]. This new discipline builds on process model-driven approaches and data mining. However, process mining is much more than an amalgamation of existing approaches. For example, existing data mining techniques are too data-centric to provide a comprehensive understanding of the end-to-end processes in an organization. Business Intelligence (BI) tools tend to focus on simple dashboards and reporting rather than clear-cut business process insights. Business Process Management (BPM) suites heavily rely on experts modeling idealized to-be processes and do not help the stakeholders to understand the as-is processes based on factual data.

Process mining provides a new means to improve processes in a variety of application domains. There are two main drivers for this new technology. On the one hand, more and more events are being recorded thus providing detailed information about the history of processes. Some figures illustrating the growth of event data can be found in [2]. Storage space grew from 2.6 optimally compressed exabytes ($2.6 \times 10^{18}$ bytes) in 1986 to 295 compressed exabytes in 2007. Note that this includes paper, photos, hard-disks, CDs, etc. In 2007, 94 percent of all information storage capacity on Earth was digital. The other 6 percent resided in books, magazines and other non-digital formats. This is in stark contrast with 1986 when only 0.8 percent of all information storage capacity was digital. These numbers illustrate the exponential growth of data. In modern organizations many events are recorded and this will only increase further, thus enabling process mining techniques. On the other hand, organizations have problems dealing with the omnipresence of event data. Most organizations diagnose problems based on fiction (Powerpoint slides, Visio diagrams, etc.) rather than facts (event data). Therefore, it is vital to turn the massive amounts of event data into relevant knowledge and insights.

Event logs can be used to conduct three types of process mining: (a) discovery, (b) conformance, and (c) enhancement [1]. The goal of *discovery* is to extract models from raw event data in information systems (transaction logs, data bases, audit trails, etc.). A discovery technique takes an event log and produces a model without using any a-priori information. An example is the $\alpha$-algorithm [3] that takes an event log and produces a Petri net explaining the behavior recorded in the log. The second type of process mining is *conformance*. Here, an existing process model is compared with an event log of the same process. Conformance checking can be used to check whether reality, as recorded in the log, conforms to the model and vice versa. Techniques as presented in [4] may be used to detect, locate and explain deviations, and to measure the severity of these deviations. The third type of process mining is *enhancement*. Here, 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, e.g., adding a new perspective to the process model by cross-correlating it with the log. An example is the extension of a process model with performance data. For instance, by combining the timestamps in the event log with the discovered process model it is possible to show bottlenecks, service levels, throughput times, and frequencies.

Process mining is not restricted to the control-flow perspective and may include other perspectives such as the resource/organizational dimension, the time/performance dimension, and the object/data dimension. However, in this paper we focus on the most challenging process mining task: *process discovery*. Although this task highly depends on the *representational bias* chosen, lion's share of attention is devoted to possible mining algorithms rather than selecting a suitable target representation. This paper demonstrates that the representational bias plays a crucial role when discovering processes.

## II. Process Discovery: A Challenging Problem

In order to explain the role of the representational bias in process discovery, we start off with an example. The example

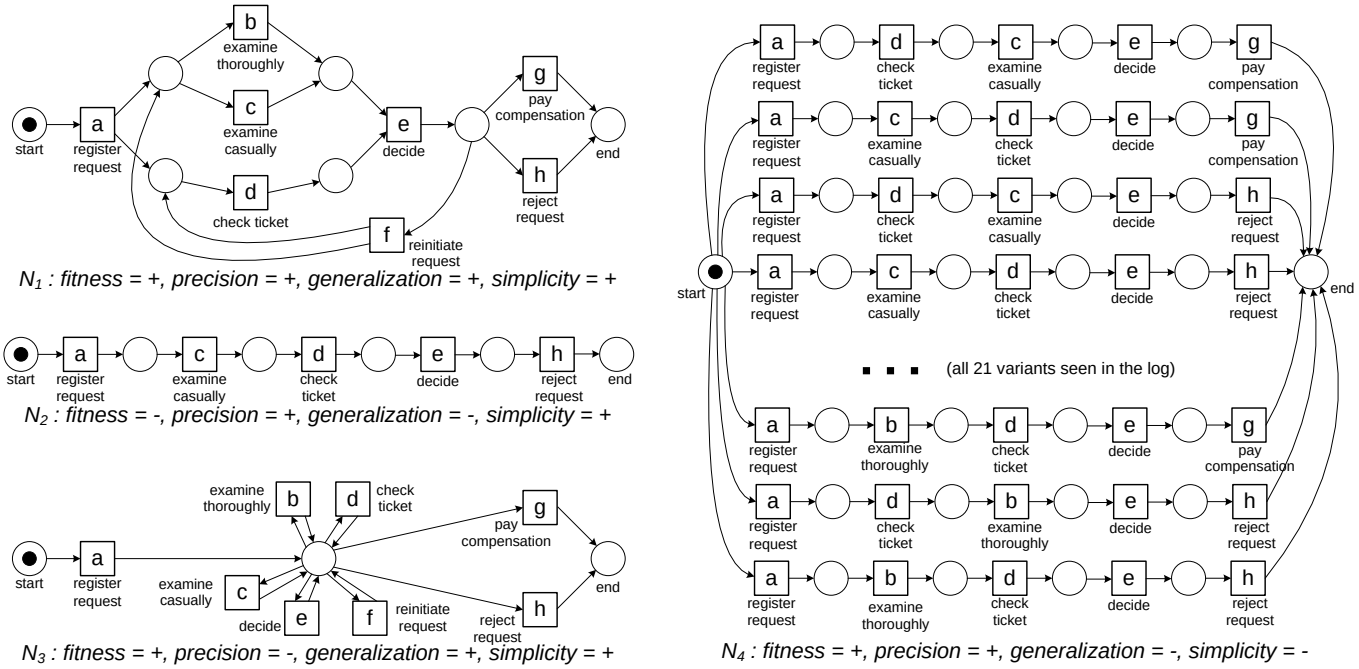


Fig. 1. Four alternative process models for the same log

is used to discuss quality criteria and challenges.

### A. Discovering Process Models: An Example

Table I shows an abstraction of some event log. The log contains information about 1391 cases (process instances). Each case corresponds to a request for compensation. We use single letters to describe activities, e.g., $a =$ *register request*. 455 cases followed the path $\langle a, c, d, e, h\rangle$, 191 cases followed the path $\langle a, b, d, e, g\rangle$, etc. The event log contains in total 7539 events. Note that Table I only shows activity names. In real-life event logs, events have timestamps, associated resources (e.g. the person executing the activity), transactional information (e.g., start, complete, or suspend), data attributes (e.g., amount or type of customer). Since we focus on control-flow discovery, we abstract from such additional information.

Figure 1 shows four models that could be discovered using existing process mining techniques. If we apply the $\alpha$-algorithm [3] to event log $L$, we obtain model $N_1$ shown in Fig. 1. $N_2$ is a model that only allows for cases having a trace $\langle a, c, d, e, h\rangle$, i.e., only the most frequent behavior is captured. $N_3$ shows a variant of the the so-called "flower model": any trace is allowed as long as it starts with $a$ and ends with $g$ or $h$. $N_4$ is the model that simply enumerates the 21 different traces seen in the event log.

Figure 1 illustrates that different process mining algorithms may produce different results. Each model is represented by a *workflow net* (WF-net). WF-nets are a subclass of Petri nets tailored towards the modeling of business processes. Each WF-net has a source place (*start*) and a sink place (*end*). Process instances "flow" from *start* to *end*. Intuitively, model

TABLE I
EVENT LOG $L$: $a =$ *register request*, $b =$ *examine thoroughly*, $c =$ *examine casually*, $d =$ *check ticket*, $e =$ *decide*, $f =$ *reinitiate request*, $g =$ *pay compensation*, AND $h =$ *reject request*

| frequency | reference | trace |
|---|---|---|
| 455 | $\sigma_1$ | $\langle a, c, d, e, h\rangle$ |
| 191 | $\sigma_2$ | $\langle a, b, d, e, g\rangle$ |
| 177 | $\sigma_3$ | $\langle a, d, c, e, h\rangle$ |
| 144 | $\sigma_4$ | $\langle a, b, d, e, h\rangle$ |
| 111 | $\sigma_5$ | $\langle a, c, d, e, g\rangle$ |
| 82 | $\sigma_6$ | $\langle a, d, c, e, g\rangle$ |
| 56 | $\sigma_7$ | $\langle a, d, b, e, h\rangle$ |
| 47 | $\sigma_8$ | $\langle a, c, d, e, f, d, b, e, h\rangle$ |
| 38 | $\sigma_9$ | $\langle a, d, b, e, g\rangle$ |
| 33 | $\sigma_{10}$ | $\langle a, c, d, e, f, b, d, e, h\rangle$ |
| 14 | $\sigma_{11}$ | $\langle a, c, d, e, f, b, d, e, g\rangle$ |
| 11 | $\sigma_{12}$ | $\langle a, c, d, e, f, d, b, e, g\rangle$ |
| 9 | $\sigma_{13}$ | $\langle a, d, c, e, f, c, d, e, h\rangle$ |
| 8 | $\sigma_{14}$ | $\langle a, d, c, e, f, d, b, e, h\rangle$ |
| 5 | $\sigma_{15}$ | $\langle a, d, c, e, f, b, d, e, g\rangle$ |
| 3 | $\sigma_{16}$ | $\langle a, c, d, e, f, b, d, e, f, d, b, e, g\rangle$ |
| 2 | $\sigma_{17}$ | $\langle a, d, c, e, f, d, b, e, g\rangle$ |
| 2 | $\sigma_{18}$ | $\langle a, d, c, e, f, b, d, e, f, b, d, e, g\rangle$ |
| 1 | $\sigma_{19}$ | $\langle a, d, c, e, f, d, b, e, f, b, d, e, h\rangle$ |
| 1 | $\sigma_{20}$ | $\langle a, d, b, e, f, b, d, e, f, d, b, e, g\rangle$ |
| 1 | $\sigma_{21}$ | $\langle a, d, c, e, f, d, b, e, f, c, d, e, f, d, b, e, g\rangle$ |

$N_1$ shown in Fig. 1 seems to capture the behavior seen in the event log in Table I well.

### B. Quality Criteria for Process Discovery

Determining the quality of a process mining result is difficult and is characterized by many dimensions. As discussed in [1], event logs may be *incomplete* and contain *noise*. Noise refers to rare and infrequent behavior not representative for the typical behavior of the process. Incompleteness refers to the

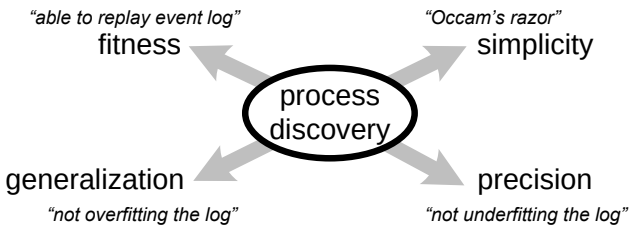


Fig. 2. Balancing the four quality dimensions: *fitness*, *simplicity*, *precision*, and *generalization*

problem that one typically sees only a fraction of all possible behaviors. Suppose that one would only have seen 1000 of the 1391 cases shown in Table I; it would be likely that some of the 21 traces would not appear in the event log. This does not mean that these trace are impossible. Typically, we only see positive examples and no negative examples.

Process mining algorithms need to be able to deal with noise and incompleteness. Generally, we use four main quality dimensions for judging the quality of the discovered process model: *fitness*, *simplicity*, *precision*, and *generalization*. Figure 2 gives a high-level characterization of these four quality dimensions.

A model with good *fitness* allows for the behavior seen in the event log. A model has a perfect fitness if all traces in the log can be replayed by the model from beginning to end. There are various ways of defining fitness [1]. It can be defined at the case level, e.g., the fraction of traces in the log that can be fully replayed. It can also be defined at the event level, e.g., the fraction of events in the log that are indeed possible according to the model. WF-nets $N_1$, $N_3$, and $N_4$ have a good fitness, i.e., in each of these models it is possible to replay all of the 1391 cases shown in Table I. WF-net $N_2$ has a poor fitness (both at the case and event level), because most of the cases/events cannot be replayed.

The *simplicity* dimension refers to *Occam's Razor*; the simplest model that can explain the behavior seen in the log, is the best model. The complexity of the model could be defined by the number of nodes and arcs in the underlying graph. Also more sophisticated metrics can be used, e.g., metrics that take the "structuredness" or "entropy" of the model into account. Clearly, WF-nets $N_1$, $N_2$, and $N_3$ are simpler than WF-net $N_4$.

Fitness and simplicity alone are not adequate. This is illustrated by WF-net $N_3$. The "flower model" allows for any sequence starting with $a$ and ending with $g$ or $h$. The resulting model is simple and has a perfect fitness. Based on the first two quality dimensions this model is acceptable. This shows that the fitness and simplicity criteria are necessary, but not sufficient.

If the "flower model" $N_3$ is on one end of the spectrum, then the "enumerating model" $N_4$ is on the other end of the spectrum. The enumerating model of a log simply lists all the sequences possible, i.e., there is a separate sequential process fragment for each trace in the model. At the start there is one big XOR split selecting one of the sequences and at the end these sequences are joined using one big XOR join.

Extreme models such as the "flower model" (anything is possible) and the "enumerating model" (only the log is possible) show the need for two additional dimensions. A model is *precise* if it does not allow for "too much" behavior. Clearly, the "flower model" lacks precision. A model that is not precise is "underfitting". Underfitting is the problem that the model over-generalizes the example behavior in the log, i.e., the model allows for behaviors very different from what was seen in the log.

A model should *generalize* and not restrict behavior to the examples seen in the log (like the "enumerating model" $N_4$). A model that does not generalize is "overfitting". Overfitting is the problem that a very specific model is generated whereas it is obvious that the log only holds example behavior, i.e., the model explains the particular sample log, but a next sample log of the same process may produce a completely different process model.

Based on the four criteria it is obvious that WF-net $N_1$ is the best model for the event log in Table I

### C. What Makes Process Discovery Difficult?

Figure 3 illustrates the problem of balancing three of the four quality criteria: *fitness*, *precision*, and *generalization*. (The fourth criterion, *simplicity*, is not directly related to the alignment of traces and model.) 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 and therefore not all dots need to be relevant for the process model to be constructed.

Recall that we defined noise as infrequent or exceptional behavior. 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 the small set of observations. Figure 3 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). Based on these assumptions, Fig. 3 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" allows for the behavior coinciding with the frequent behavior seen when the process

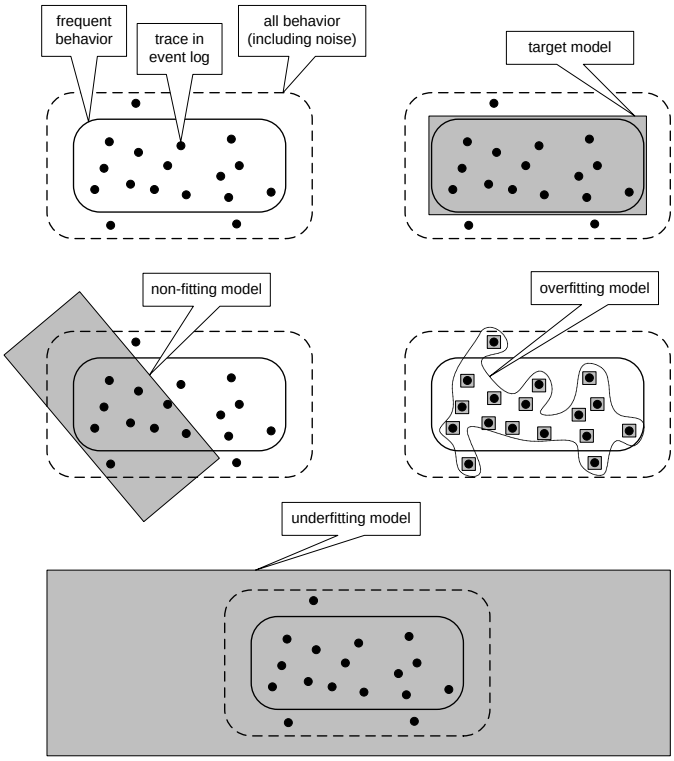


Fig. 3. Overview of the challenges that process discovery techniques need to address

would be observed ad infinitum. The "non-fitting model" in Fig. 3 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" 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" lacks precision and allows for behavior that would never be seen if the process would be observed ad infinitum.

Figure 3 illustrates the challenges process discovery techniques need to address: How to extract a simple target model that is not underfitting, overfitting, nor non-fitting?

## III. Representational Bias

Figure 1 shows four example models that may be discovered based on the event log in Table I. Note that these are only examples, e.g., the $\alpha$-algorithm will generate WF-net $N_1$. The $\alpha$-algorithm [3] *assumes that the underlying process can be adequately described by a WF-net*. Any discovery technique requires such a *representational bias*. The notion of a representational bias can be (metaphorically) illustrated using Fig. 3. If we assume upfront that the target model is a "circle" or a "triangle" while the frequent behavior forms a "rectangle", then it would be difficult to find a suitable process model. This section will show that this representational bias is of crucial importance. Whereas most people focus on understandability of the representation, we emphasize the importance of the implicit search space implied by the representational bias.

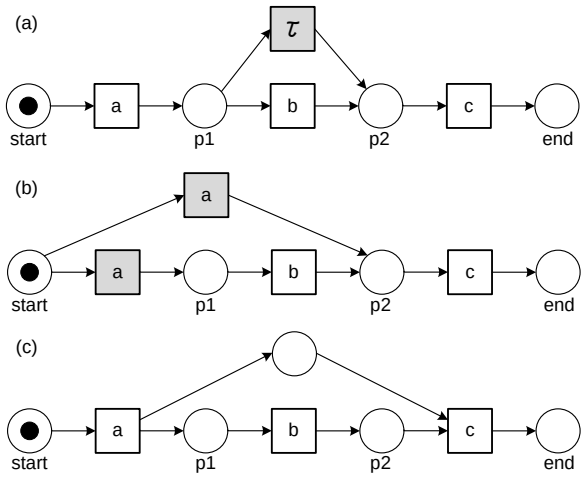


Fig. 4. Three WF-nets for the event log $L_1 = [\langle a, b, c\rangle^{20}, \langle a, c\rangle^{30}]$

### A. Example: The Representational Bias of the $\alpha$-Algorithm

The $\alpha$-algorithm assumes that the underlying process can be described by a WF-net where each transition bears a unique and visible label. In such a WF-net it is not possible to have two transitions with the same label or transitions whose occurrences remain invisible (i.e., it is not possible to have a so-called "silent transition" $\tau$). These assumptions may seem harmless, but, as shown next, have a noticeable effect on the class of process models that can be discovered.

Let us consider, for example, event log $L_1 = [\langle a, b, c\rangle^{20}, \langle a, c\rangle^{30}]$. Figure 4(a) describes the underlying process well: activity $b$ can be skipped by executing the $\tau$ transition. (Note that the $\tau$ transition corresponds to a so-called "silent" step of the process, i.e., it is not recorded.) Figure 4(b) shows an alternative WF-net using two $a$ transitions and no $\tau$ transition. These two models are trace equivalent. However, it is not possible to construct a WF-net without duplicate and $\tau$ labels that is trace equivalent to these two models. Figure 4(c) shows the model produced by the $\alpha$-algorithm; because of the representational bias, the algorithm is destined to fail for this log. The WF-net in Fig. 4(c) can only reproduce trace $\langle a, b, c\rangle$ and not $\langle a, c\rangle$.

Event logs $L_1$ illustrates the effect a representational bias can have. From the viewpoint of the $\alpha$-algorithm, the choice to not consider duplicate labels and $\tau$ transitions is sensible. $\tau$ transitions are not recorded in the log and hence any algorithm will have problems reconstructing their behavior. Multiple transitions with the same label are undistinguishable in the event log. Therefore, any algorithm will have problems associating the corresponding events to one of these transitions.

Figure 5 shows another example illustrating the effect a representational bias can have. The WF-net in Fig. 5(a) has a so-called "non-free-choice construct". The concept of *free-choice nets* is well-defined in the Petri net domain [5]. A Petri net is free choice if any two transitions sharing an input place have identical input sets. Transitions $d$ and $e$ share an input

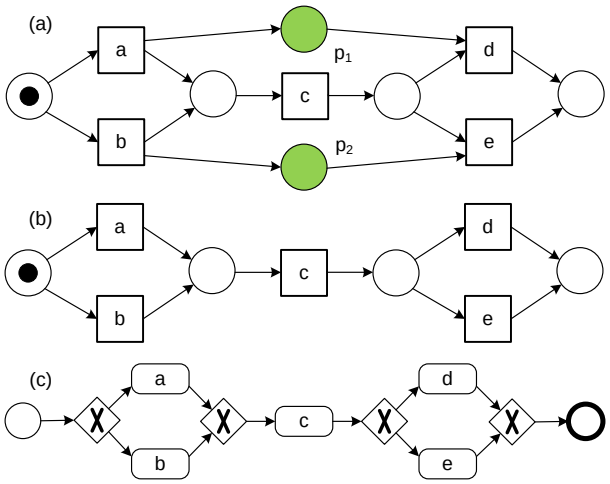


Fig. 5. Two WF-nets and one BPMN model for the event logs $L_2 = [\langle a, c, d\rangle^{20}, \langle b, c, e\rangle^{30}]$ and $L_3 = [\langle a, c, d\rangle^{20}, \langle b, c, e\rangle^{30}, \langle a, c, e\rangle^{1}, \langle b, c, d\rangle^{2}]$

place, but have different input sets. For example, place $p_1$ is an input place of $d$, but not of $e$. Places $p_1$ and $p_2$ "control" the choice following $c$. Therefore, the WF-net in Fig. 5(a) allows for only two possible traces: $\langle a, c, d\rangle$ and $\langle b, c, e\rangle$.

The WF-net in Fig. 5(b) is free-choice net because the choice between $d$ and $e$ is no longer controlled by $p_1$ and $p_2$. Now there are four possible traces. For example, $\langle a, c, e\rangle$ is also possible. Fig. 5(c) shows the corresponding BPMN notation. The BPMN notation does not allow for the "non-free-choice construct" shown in the first WF-net. Most process mining algorithms do *not* allow for "non-free-choice constructs" because of their representational bias.

Now consider two event logs: $L_2 = [\langle a, c, d\rangle^{20}, \langle b, c, e\rangle^{30}]$ and $L_3 = [\langle a, c, d\rangle^{20}, \langle b, c, e\rangle^{30}, \langle a, c, e\rangle^{1}, \langle b, c, d\rangle^{2}]$.

The WF-net in Fig. 5(b) and the BPMN model in Fig. 5(c) can replay both logs, i.e., fitness is good with respect to $L_2$ and $L_3$. The WF-net in Fig. 5(a) can replay $L_2$ but not $L_3$. However, the fitness with respect to $L_3$ is reasonable as 50 out of 53 cases can be replayed. One could argue that the WF-net in Fig. 5(b) and the BPMN model in Fig. 5(c) are underfitting both logs. In fact, for $L_2$, the non-free-choice WF-net in Fig. 5(a) is clearly the best model. However, many process modeling languages are inherently free-choice, thus making it impossible to discover $p_1$ and $p_2$.

The non-free-choice construct is just one of many constructs that existing process mining algorithms have problems with. Other examples are arbitrary nested loops, cancelation, unbalanced splits and joins, and partial synchronization. In this context it is important to observe that *process discovery is, by definition, restricted by the expressive power of the target language*, i.e., the representational bias.

### B. Typical representational limitations

The well-know *workflow patterns* [6], [7] serve as a good basis for discussing the limitations imposed by the representational bias of a process mining algorithm. The *Workflow Patterns Initiative* was established with the aim of delineating the fundamental requirements that arise during business process modeling on a recurring basis and describe them in an imperative way. The patterns developed in this context help to discuss and identify the representational bias of a language. Here, we do not discuss the more than 40 control-flow patterns [7]. Instead, we mention some typical representational limitations imposed by process discovery algorithms:

- **Inability to represent concurrency.** Low-level models, such as Markov models, flow charts, and transition systems, do not allow for the modeling of concurrency other than enumerating all possible interleavings. To model a process with 10 parallel activities, a low-level model will need to enumerate all $2^{10} = 1024$ states and $10 \times 2^{10-1} = 5120$ transitions. Higher level models (like Petri nets and BPMN) only need to depict 10 activities and $2 \times 10 = 20$ "local" states (states before and after each activity).
- **Inability to represent silent actions.** In some notations, it is impossible to model silent actions like the skipping of an activity. Although such events are not explicitly recorded in the event log, they need to be reflected in the model (as illustrated by Fig. 4).
- **Inability to represent duplicate actions.** In many notations there cannot be two activities having the same label. If the same activity appears in different parts of the process, but these different instances of the same activity cannot be distinguished in the event log, then most algorithms will assume a single activity thus creating causal dependencies (e.g., non-existing loops) that do not exist in the actual process.
- **Inability to model OR-splits/joins.** Higher level notations such as YAWL, BPMN, EPCs, causal nets, etc. allow for the modeling of OR-splits and OR-joins. If the representational bias of a discovery algorithm does not allow for OR-splits and OR-joins, then the discovered model may be more complex or the algorithm is unable to find a suitable model.
- **Inability to represent non-free-choice behavior.** Most algorithms do not allow for non-free-choice constructs, i.e., constructs where concurrency and choice meet. Non-free-choice constructs can be used to represent non-local dependencies as is illustrated by the WF-net in Fig. 5(a). Many notations do not allow for such constructs.
- **Inability to represent hierarchy.** Most process discovery algorithms work on "flat" models. A notable exception is the Fuzzy Miner [8] that extracts hierarchical models. Activities that have a lower frequency but that are closely related to other low frequent activities are grouped into subprocesses. The representational bias determines whether, in principle, hierarchical models can be discovered or not.

### C. Improving the Representational Bias

The representational bias helps limiting the search space of possible candidate models. This can make discovery algo-

rithms more efficient. Moreover, it can also be used to give preference to particular types of models.

It seems that existing approaches can benefit from selecting a more suitable representational bias. Most process discovery algorithms allow for models that have all kinds of obvious problems, e.g., deadlocks, livelocks, inability to terminate, improper termination, and dead activities. The *soundness property* [9] defined for WF-nets and other notations is a domain-independent requirement. *It is desirable to have a representational bias that limits the search space to only sound models (i.e., free of deadlocks and other anomalies).* Unfortunately, this is not the case for most of the existing approaches. For instance, the $\alpha$-algorithm may yield models that have deadlocks or livelocks. Genetic process mining algorithms tend continuously explore "bad candidates" [1].

Therefore, one would like to have a representational bias enforcing soundness. Unfortunately, currently, this can typically only be achieved by severely limiting the expressiveness of the modeling language or by using more time-consuming analysis techniques. Consider, for example, the so-called *block-structured* process models. A model is block-structured if it satisfies a number of syntactical requirements such that soundness is guaranteed by these requirements. See [10]–[12] for pointers to various definitions. Most of these definitions require a one-to-one correspondence between splits and joins, e.g., concurrent paths created by an AND-split need to be synchronized by the corresponding AND-join. Since many real-life processes are not block structured, one should be careful to not limit the expressiveness too much. Note that techniques that turn unstructured models into block-structured process models tend to introduce many duplicate or silent activities. Therefore, such transformations do not alleviate the core problems.

Soundness is difficult to incorporate because it is related to behavior rather than structure. Structural requirements can be incorporated more easily. As an example, we refer to *region-based process mining techniques* [13]–[16]. State-based regions can be used to construct a Petri net from a transition system [14]. The transition system can be extracted from an event log using different abstraction mechanisms (see [13] for an overview). Language-based regions can be used to construct a Petri net from a prefix-closed language. Synthesis approaches using language-based regions can be applied directly to event logs [15], [16].

In [14] it is shown how additional requirements can be imposed on the Petri net constructed based on the transition system. For example, one can make sure that the resulting model is free-choice or without self-loops. The technique described in [14] uses label-splitting and in [13] it is shown how this can be used in the context of process mining.

As shown in [16], similar requirements can be imposed on the resulting models when using language-based regions. The representational bias can be modified to enforce certain structural properties, such as marked graphs, state machines, pure nets, elementary nets, and free-choice nets. Moreover, proper termination and other desirable properties can be encoded in the ILP formulation of the problem [16].

## IV. Conclusion

In this paper, we emphasized the importance of selecting the right representational bias when discovering process models from event logs. The representational bias should be based on essential properties of a process model (e.g., soundness [9]), and not driven by the desired graphical presentation. Improving the representational bias will improve both the quality of the results and the efficiency of the algorithms.


## Acknowledgment

The author would like to thank the members of the IEEE Task Force on Process Mining (www.win.tue.nl/ieeetfpm/) and all that contributed to the development of ProM (www.processmining.org).


## References


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

[2] M. Hilbert and P. Lopez, "The World's Technological Capacity to Store, Communicate, and Compute Information," *Science*, 2011.

[3] W. van der Aalst, A. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[4] A. Rozinat and W. van der Aalst, "Conformance Checking of Processes Based on Monitoring Real Behavior," *Information Systems*, vol. 33, no. 1, pp. 64–95, 2008.

[5] J. Desel and J. Esparza, *Free Choice Petri Nets*, Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, 1995, vol. 40.

[6] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[7] "Workflow Patterns Home Page," http://www.workflowpatterns.com.

[8] C. Günther and W. van der Aalst, "Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics," in *International Conference on Business Process Management (BPM 2007)*, Lecture Notes in Computer Science, G. Alonso, P. Dadam, and M. Rosemann, Eds., vol. 4714. Springer-Verlag, Berlin, 2007, pp. 328–343.

[9] W. van der Aalst, K. van Hee, A. ter Hofstede, N. Sidorova, H. Verbeek, M. Voorhoeve, and M. Wynn, "Soundness of Workflow Nets: Classification, Decidability, and Analysis," *Formal Aspects of Computing*, 2011, dx.doi.org/10.1007/s00165-010-0161-4.

[10] M. Dumas, W. van der Aalst, and A. ter Hofstede, *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley & Sons, 2005.

[11] A. ter Hofstede, W. van der Aalst, M. Adams, and N. Russell, *Modern Business Process Automation: YAWL and its Support Environment*. Springer-Verlag, Berlin, 2010.

[12] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, Berlin, 2007.

[13] W. van der Aalst, V. Rubin, H. Verbeek, B. van Dongen, E. Kindler, and C. Günther, "Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting," *Software and Systems Modeling*, vol. 9, no. 1, pp. 87–111, 2010.

[14] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Deriving Petri Nets from Finite Transition Systems," *IEEE Transactions on Computers*, vol. 47, no. 8, pp. 859–882, Aug. 1998.

[15] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Process Mining Based on Regions of Languages," in *International Conference on Business Process Management (BPM 2007)*, Lecture Notes in Computer Science, G. Alonso, P. Dadam, and M. Rosemann, Eds., vol. 4714. Springer-Verlag, Berlin, 2007, pp. 375–383.

[16] J. van der Werf, B. van Dongen, C. Hurkens, and A. Serebrenik, "Process Discovery using Integer Linear Programming," *Fundamenta Informaticae*, vol. 94, pp. 387–412, 2010.