# Towards an Evaluation Framework for Process Mining Algorithms

A. Rozinat, A.K. Alves de Medeiros, C.W. Günther, A.J.M.M. Weijters, and W.M.P. van der Aalst

Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{a.rozinat,a.k.medeiros,c.w.gunther,a.j.m.m.weijters,
w.m.p.v.d.aalst}@tue.nl

**Abstract.** Although there has been a lot of progress in developing process mining algorithms in recent years, no effort has been put in developing a common means of assessing the quality of the models discovered by these algorithms. In this paper, we outline elements of an evaluation framework that is intended to enable (a) process mining researchers to compare the performance of their algorithms, and (b) end users to evaluate the validity of their process mining results. Furthermore, we describe two possible approaches to evaluate a discovered model (i) using existing comparison metrics that have been developed by the process mining research community, and (ii) based on the so-called k-fold-cross validation known from the machine learning community. To illustrate the application of these two approaches, we compared a set of models discovered by different algorithms based on a simple example log.

## 1 Introduction

Process mining has proven to be a valuable approach that provides new and objective insights into the way business processes are actually conducted within organizations. Taking a set of real executions (the so-called "event log") as the starting point, these techniques attempt to extract non-trivial and useful process information from various perspectives, such as control flow, data flow, organizational structures, and performance characteristics. A common *mining XML* (MXML) log format was defined in [5] to enable researchers and practitioners to share their logs in a standardized way. However, while process mining has reached a certain level of maturity and has been used in a variety of real-life case studies (see [3, 25] for two examples), *a common framework to evaluate process mining results is still lacking*. We believe that there is the need for a concrete framework that enables (a) process mining researchers to compare the performance of their algorithms, and (b) end users to evaluate the validity of their process mining results. This paper is a first step into this direction.

The driving element in the process mining domain is some operational process, for example a business process such as an insurance claim handling procedure in an insurance company, or the booking process of a travel agency. Nowadays, many business processes are supported by information systems that help

coordinating the steps that need to be performed in the course of the process. Workflow systems, for example, assign work items to employees according to their roles and the status of the process. Typically, these systems record *events* related to the activities that are performed, e.g., in audit trails or transaction logs [5].[1] These event logs form the input for process mining algorithms.

In this paper we focus on providing a means of comparison for algorithms that discover the *control-flow perspective* of a process (which we simply refer to as process discovery algorithms from now on). In particular, we focus on *validation techniques* for these process discovery algorithms. We argue that this evaluation can take place in different dimensions, and we describe two different validation approaches: one based on existing validation metrics, and another based on the so-called k-fold cross validation technique known from the machine learning domain. To directly support the evaluation and comparison of different mining results (both for researchers and end users), we have implemented an extensible `Control Flow Benchmark` plug-in in the context of the ProM framework[2].

The remainder of this paper is organized as follows. Section 2 motivates the need for an evaluation framework. Then, Section 3 outlines first steps towards such a common framework. Subsequently, two different evaluation approaches are described and illustrated based on the running example in Section 4 and in Section 5. Finally, Section 6 presents the `Control Flow Benchmark` plug-in in ProM, and the paper concludes. Related work is reviewed throughout the paper and, therefore, not provided in a separate section.

## 2 Process Discovery: Which Model is the "Best"?

The goal of a process discovery algorithm is to construct a process model which reflects the behavior that has been observed in the event log. Different process modeling languages[3] can be used to capture the causal relationships of the steps, or activities, in the process. The idea of applying process mining in the context of workflow management was first introduced in [8]. Over the last decade many process mining approaches have been proposed [10, 18]. For more information on process mining we refer to a special issue of Computers in Industry on process mining [6] and a survey paper [5]. While all these approaches aim at the discovery of a "good" process model, often targeting particular challenges (e.g., the mining of loops, or duplicate tasks), they have their limitations and many different event logs and quality measurements are used. Hence, no standard measure is available.

[1] It is important to note that information systems that do not enforce users to follow a particular process often still provide detailed event logs, e.g., hospital information systems, ERP systems etc.

[2] ProM offers a wide range of tools related to process mining and process analysis. Both documentation and software (including the source code) can be downloaded from *http://www.processmining.org*.

[3] In the remainder of this paper we will use Petri nets, motivated by their formal semantics. Note that in our tool ProM there exist translations from process modeling languages such as EPC, YAWL, and BPEL to Petri nets and vice-versa.

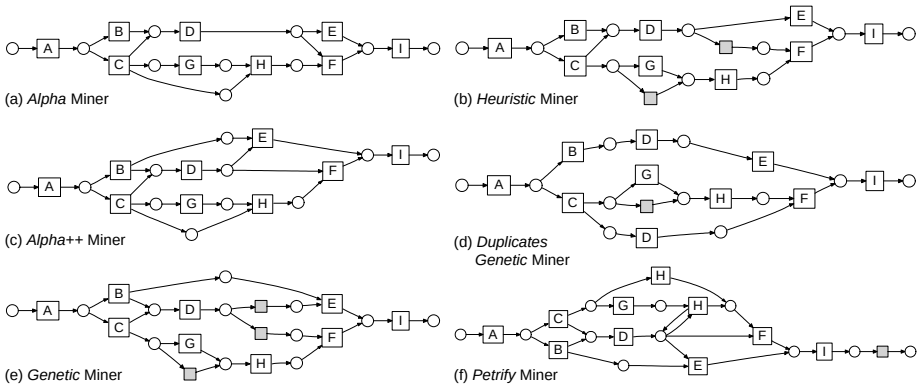



**Fig. 1.** Process models that were discovered by different process discovery algorithms based on the same log

To illustrate the dilemma, we consider a simple example log, which only contains the following five different traces: *ABDEI*, *ACDGHFI*, *ACGDHFI*, *ACHDFI*, and *ACDHFI*. We applied six different process mining algorithms that are available in ProM and obtained six different process models (for every plug-in, we used the default settings in ProM 4.1). Figure 1 depicts the mining results for the `Alpha` miner [7], the `Heuristic` miner [27], the `Alpha++` miner [29], the `Duplicates Genetic` miner and the `Genetics` miner [12], and the `Petrify` miner [4]. The models seem similar, but are all different[4]. Are they equivalent? If not, which one is the "best"?

These questions are interesting both for researchers and end users: (a) Researchers typically attempt to let their process discovery algorithms construct process models that completely and precisely reflect the observed behavior in a structurally suitable way. It would be useful to have common data sets containing logs with different characteristics, which can be used within the scientific community to systematically compare the performance of various algorithms in different, controlled environments. (b) Users of process discovery techniques, on the other hand, need to know how well the discovered model describes reality, how many cases are actually covered by the generated process description etc. For example, if in an organization process mining is to be used as a knowledge discovery tool in the context of a Business Process Intelligence (BPI) framework, it must be possible to estimate the "accuracy" of a discovered model, i.e., the "confidence" with which it reflects the underlying process. Furthermore, end users need to be able to compare the results obtained from different process discovery algorithms.

[4] Note that throughout this paper the invisible (i.e., unlabeled) tasks need to be interpreted using the so-called "lazy semantics", i.e., they are only fired if they enable a succeeding, visible task [12].

## 3 Towards a Common Evaluation Framework

In an experimental setting, we usually know the original model that was used to generate an event log. For example, the log in Figure 2(a) was created from the simulation of the process model depicted in Figure 2(b). Knowing this, one could leverage *process equivalence* notions to evaluate the discovered model with respect to the original model [1]. But in many practical situations no original model is available. However, if we assume that the behavior observed in the log is what really happened (and somehow representative for the operational process at hand), it is possible to compare the discovered model to the event log that was used as input for the discovery algorithm. This essentially results in a *conformance analysis* problem [24, 11]. In either case quality criteria need to be determined.

In the remainder of this section, we first consider a number of evaluation dimensions (Section 3.1). Afterwards, we outline ingredients of a possible evaluation framework (Section 3.2).

### 3.1 Evaluation Dimensions

Figure 2 depicts an event log (a) and four different process models (b-e). While Figure 2(b) depicts a "good" model for the event log in Figure 2(a), the remaining three models show undesirable, extreme models that might also be returned by a process mining algorithm. They illustrate that the evaluation of an event log and a process model can take place in different, orthogonal dimensions.

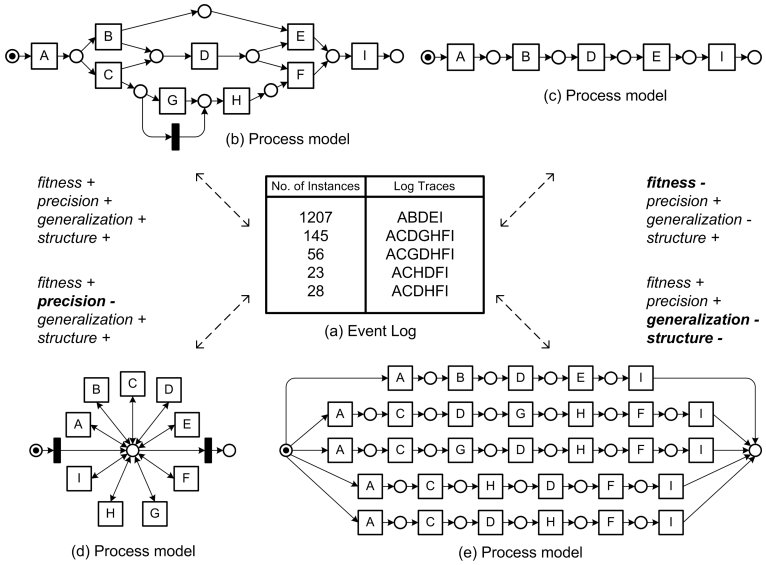



**Fig. 2.** The evaluation of a process model can take place in different dimensions

**Fitness.** The first dimension is fitness, which indicates how much of the observed behavior is captured by (i.e., "fits") the process model. For example, the model in Figure 2(c) is only able to reproduce the sequence $ABDEI$, but not the other sequences in the log. Therefore, its fitness is poor.

**Precision.** The second dimension addresses overly general models. For example, the model in Figure 2(d) allows for the execution of activities $A - I$ in any order (i.e., also the sequences in the log). Therefore, the fitness is good, but the precision is poor. Note that the model in Figure 2(b) is also considered to be a precise model, although it additionally allows for the trace $ACGHDFI$ (which is not in the log). Because the number of possible sequences generated by a process model may grow exponentially, it is not likely that all the possible behavior has been observed in a log. Therefore, process mining techniques strive for weakening the notion of *completeness* (i.e., the amount of information a log needs to contain to be able to rediscover the underlying process [7]). For example, they want to detect parallel tasks without the need to observe every possible interleaving between them.

**Generalization.** The third dimension addresses overly precise models. For example, the model in Figure 2(e) only allows for *exactly* the five sequences from the log[5]. In contrast to the model in Figure 2(b), which also allows for the trace $ACGHDFI$, no generalization was performed in the model in Figure 2(e). To determine the right level of generalization remains a challenge, especially when dealing with logs that contain *noise* (i.e., distorted data). Similarly, in the context of more unstructured and/or flexible processes, it is essential to further abstract from less important behavior (i.e., restriction rather than generalization). In general, abstraction can lead to the omission of connections between activities, which could mean lower precision or lower fitness (e.g., only capturing the most frequent paths). Furthermore, steps in the process could be left out completely. Therefore, abstraction must be seen as a different evaluation dimension, which needs to be balanced against precision and fitness.

**Structure.** The last dimension is the structure of a process model, which is determined by the vocabulary of the modeling language (e.g., routing nodes with AND and XOR semantics). Often there are several syntactic ways to express the same behavior, and there may be "preferred" and "less suitable" representations. For example, the fitness and precision of the model in Figure 2(e) are good, but it contains many duplicate tasks, which makes it difficult to read. Clearly, this evaluation dimension highly depends on the process modeling formalism, and is difficult to assess in an objective way as it relates to human modeling capabilities.

### 3.2 Evaluation Framework

Currently, process mining researchers are forced to implement their custom, ad-hoc simulation environments to generate process models and/or logs that can

[5] Note that—unlike in the log shown in Figure 2(a)—in real-life logs there are often traces that are unique. Therefore, it is unlikely that the log contains all possible traces and generalization is essential.

then be used to evaluate their mining approach, or to compare their approach to other mining approaches (see [28] and [22] for examples). In the remainder of this section we identify elements that are relevant in the context of a common evaluation framework (cf. Figure 3).

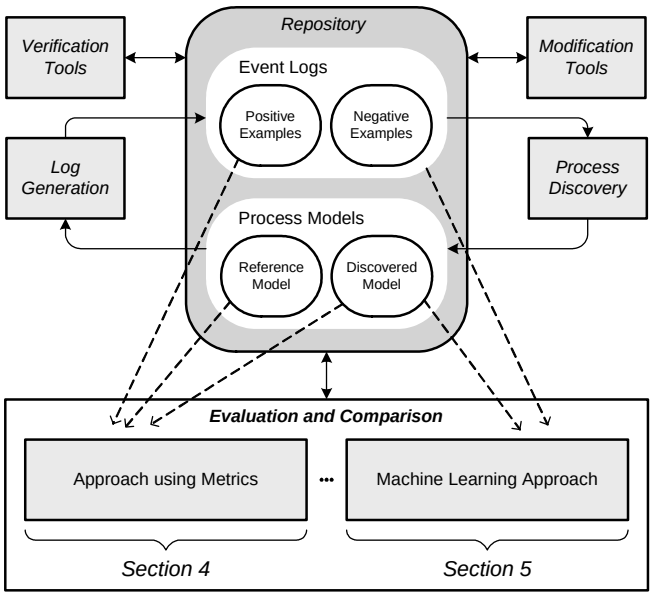



**Fig. 3.** Outline of a possible evaluation framework

**Repository.** To systematically compare process mining algorithms, there should be common data sets, which can be used and extended by different researchers to "benchmark" their algorithms on a per-dataset basis. For instance, in the machine learning community there are well know data sets (e.g., the UCI Machine Learning Repository, CMU NN-Bench Collection, Proben1, StatLog, ELENA-data, etc.) that can be used for testing and comparing different techniques. Such a process mining repository would consist of a collection of *Event Logs* and *Process Models*, and should also provide information about the process or log characteristics as these may pose special challenges. Furthermore, the results of an evaluation could be stored for later reference.

**Log Generation.** At the same time it is necessary to be able to influence both the process and log characteristics. For example, one might want to generate an event log containing noise (i.e., distorting the logged information), or a certain timing behavior (some activities taking more time than others), from a given model. For log generation, simulation tools such as CPN Tools can be used. Note that we developed ML functions to generate MXML logs in CPN Tools [13]. Moreover, a `CPN Export` plug-in in ProM can be used to extend a Petri net process model with certain data, time, and resource characteristics, and to

automatically generate a Colored Petri net including the monitors necessary to generate logs during simulation in CPN Tools [26]. Another example for log generation is the generation of "forbidden" scenarios (i.e., *negative examples*) as a complement to the actual execution log (i.e., *positive examples*).

**Modification and Verification Tools.** Furthermore, it is necessary to verify certain properties with respect to event logs or process models. For example, assumptions such as the absence of duplicate tasks in the model must be checked to ensure the validity of a certain evaluation metric (see also Section 4), or the *completeness* of the log needs to be ensured before starting the process discovery algorithm. Finally, the modification of event logs or process models might be necessary to, e.g., generate "noisy" logs (to test the performance of the process mining algorithm on distorted data), add artificial start and end activities (to fulfill certain pre-conditions), or the log needs to be split up into training and test set (see also Section 5).

**Evaluation and Comparison.** Clearly, there may be many different approaches for evaluation and comparison of the discovered process models. This is indicated by the lower half of Figure 3 and the remainder of this paper will focus on this aspect. As a first step, we looked at existing work both in the process mining and data mining domain, and describe two different evaluation approaches. In the first approach (Section 4), the quality of the discovered model is directly evaluated using existing validation metrics. Some of these metrics make use of a *reference model* (i.e., the model that was used to generate the event log). The second approach (Section 5) is based on evaluation techniques known from the machine learning domain, where a part of the input data is held back for evaluation purposes. Furthermore, it uses *negative examples* to punish over-general models.

## 4 Approach using Metrics

Process mining literature describes several measures to assess the quality of a model. In the desire to discover a "good" model for the behavior observed in the execution log, shared notions of, e.g., fitness, precision and structure have been developed. Naturally, these quality measures can be used to evaluate and compare discovered process models. They are typically applied in a specific mining context, i.e., assumptions are made with respect to the corresponding mining algorithm. However, these assumptions do not need to be relevant for the applicability of the defined validation metrics, and, therefore, it is interesting to investigate to which extent these metrics can be used in a more general evaluation setting.

In the remainder of this section, we first review existing metrics that have been used by different authors to validate the quality of their (mined) models (Section 4.1). Here the dimensions defined in Section 3.1 are used to structure the review. Then, we calculate some of these metrics for the process models shown in Figure 1 (Section 4.2).

### 4.1 Review of Existing Metrics

To provide an overview about the reviewed validation metrics in the process mining area, Table 1 summarizes the following aspects:

- Which of the *dimensions*, i.e., fitness, precision, generalization, and structure (cf. Section 3.1), are addressed by this metric?
- Which *input* is required to calculate the metric (reference log and/or reference model)?
- What is the *output value range*?
- What is the computational *complexity*? Here, we give a rough indication based on whether the state space of the model needs to be explored ($-$), some form of log replay is performed ($+/-$), or purely structural aspects of the model need to be considered ($+$).
- For which process *model type* is the metric defined?
- Is it already *implemented in the ProM framework*?
- Which *further assumptions* need to hold to calculate the metric, or to obtain a valid result?

In the following, we briefly review the metrics listed in Table 1 (for further details the interested reader is kindly referred to the cited papers).

**Metrics to quantify *fitness*.** Fitness refers to how much behavior in a log is correctly captured (or can be reproduced) by a model. From literature, we identified five metrics that relate to the fitness dimension: Completeness (both *completeness* [17] and $PF_{complete}$ [12]), Fitness ($f$) [24], Parsing Measure (`PM`) and Continuous Parsing Measure (`CPM`) [28]. All these metrics are based on replaying logs in process models and they mainly differ in what they consider to be the "unit of behavior".

- For the metrics *completeness* and `PM`, the *trace* is the unit of behavior. Both metrics quantify which percentage of the traces in a log can also be generated by a model. For the remaining three metrics (`CPM`, $f$, and $PF_{complete}$), both *traces and tasks* are units of behavior. The metric `CPM` gives equal importance to the total number of correctly replayed traces and the total number of correctly replayed tasks.
- The metric $f$ is more fine-grained than the metric `CPM` because it also considers which problems (missing and remaining tokens) happened during the log replay. This way, it equally punishes problems of events that cannot be replayed as the corresponding activity is "not activated", and the problem of activities that "remain activated" in an improper way.
- The metric $PF_{complete}$ is very similar to the metric $f$, but it also takes into account trace frequencies when weighing the problems during the log replay. Consequently, problems in low-frequent traces become less important than problems in traces that are very frequent in the log.

**Table 1.** Overview of existing validation metrics in the process mining area

| *Authors* | *Metrics* | *Dimensions* | | | | *Required Input* | | | *Output Value Range* | *Complexity* | *Model Type* | *Implem. in ProM* | *Further Assumptions* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *fitness* | *precision* | *generalization* | *structure* | *log* | *mined model* | *reference model* | | | | | |
| Greco et al. [17] | *completeness* | ✓ | | | | ✓ | ✓ | | [0,1] | +/− | Workflow | see `PM` | |
| | *soundness* | | ✓ | | | ✓ | ✓ | | [0,1] | − | Schema | | No loops |
| Weijters et al. [28] | Parsing Measure (`PM`) | ✓ | | | | ✓ | ✓ | | [0,1] | +/− | Heuristic net | ✓ | see $PF_{complete}$ |
| | Continuous Parsing Measure (`CPM`) | ✓ | | | | ✓ | ✓ | | [0,1] | +/− | | | |
| Rozinat et al. [24] | Fitness ($f$) | ✓ | | | | ✓ | ✓ | | [0,1] | +/− | Petri net | ✓ | Sound WF-net |
| | Behavioral Appropriateness ($a'_B$) | | ✓ | | | ✓ | ✓ | | [0,1] | − | | ✓ | |
| | Structural Appropriateness ($a'_S$) | | | | ✓ | ✓ | ✓ | | [0,1] | − | | ✓ | |
| Alves de Medeiros et al. [12] | Completeness ($PF_{complete}$) | ✓ | | | | ✓ | ✓ | | (−∞,1] | +/− | Heuristic net | ✓ | No duplicate tasks share input tasks or output tasks |
| | Behavioral Precision ($B_P$) | | ✓ | | | ✓ | ✓ | ✓ | [0,1] | +/− | | ✓ | |
| | and Recall ($B_R$) | | | ✓ | | ✓ | ✓ | ✓ | [0,1] | +/− | | ✓ | |
| | Structural Precision ($S_P$) | | | | ✓ | | ✓ | ✓ | [0,1] | + | | ✓ | − |
| | and Recall ($S_R$) | | | | ✓ | | ✓ | ✓ | [0,1] | + | | ✓ | |
| | Duplicates Precision ($D_P$) | | | | ✓ | | ✓ | ✓ | [0,1] | + | | ✓ | Same task labels in both models |
| | and Recall ($D_R$) | | | | ✓ | | ✓ | ✓ | [0,1] | + | | ✓ | |
| van Dongen et al. [14] | *Causal Footprint* | | ✓ | ✓ | | | ✓ | ✓ | [0,1] | − | Petri net or EPC | ✓ | No duplicate tasks |
| Pinter et al. [22] | *precision* and | | | | ✓ | | ✓ | ✓ | [0,1] | + | Flowmark language | see $S_P$ | |
| | *recall* | | | | ✓ | | ✓ | ✓ | [0,1] | + | | see $S_R$ | |

**Metrics to quantify *precision* and *generalization*.** Precision and generalization refer to how much more behavior is captured in the model than was observed in the log. The following metrics focus only on the precision dimension: *soundness* [17], (advanced) Behavioral Appropriateness ($a'_B$) [24], and Behavioral Precision ($B_P$) [12]. The Behavioral Recall ($B_R$) [12] metric focuses only on the generalization dimension, and the *Causal Footprint* [14] addresses both precision and generalization (basically evaluating the equivalence of a mined model with respect to the reference model).

- The metric *soundness* calculates the percentage of traces that can be generated by a model and are in a log. The problem with this metric is that it only makes sense when the log contains *all* possible traces. As mentioned earlier, this might be unrealistic when the target model has many tasks in parallel and is impossible when the target model has loop constructs.
- The $a'_B$ metric derives *"sometimes" follows and precedes relations* (reflecting alternative or parallel behavior) for tasks in a log and for tasks in a model, and compares these relations. The less relations derived from the model can also be derived from the log, the less precise is the model. While these relations can be derived from the log in a linear way, deriving them from the model requires an exploration of the state space of the model.
- The *behavioral precision* ($B_P$) and *recall* ($B_R$) quantify the precision and generalization of the mined model with respect to the input log and the reference model that was used to generate this log. The metrics $B_P$ and $B_R$ measure the intersection between the set of enabled tasks that the mined and reference models have at every moment of the log replay. This intersection is further weighed by the frequency of traces in the log. $B_P$ measures how much extra behavior the mined model allows for with respect to a given reference model and log. $B_R$ quantifies the opposite. These metrics have the advantage that they capture the *moment of choice* in the models and the differences of behavior in low and high frequent traces.
- The metric *Causal Footprint* measures behavioral similarity based on two models' structures. It works by (i) mapping these models to their *causal closure graphs*, (ii) transforming these graphs to vectors in a multidimensional space, and (iii) measuring the cosine distance between these two vectors.

**Metrics to quantify *structure*.** The structure of a model consists of (i) its tasks, (ii) the connections (or dependencies) between these tasks, and (iii) the semantics of the split/join points. The existing metrics for structural comparison typically focus on one or more of these constituent elements, but not all of them.

- The metric Structural Appropriateness ($a'_S$) [24] measures if a model is less compact (in terms of the number of tasks) than necessary. It punishes models with extra alternative duplicate tasks and redundant invisible tasks independently of the model behavior (so it can also be used to compare models with different behaviors).

– The metrics *precision* and *recall* [22] (like the respective metrics Structural Precision ($S_P$) and Structural Recall ($S_R$) [12]) quantify how many connections the mined model and the reference models have in common. When the mined model has connections that do not appear in the reference model, *precision* will have a value lower than 1. When the reference model has connections that do not appear in the mined model, *recall* will have a value smaller than 1.
– The metrics Duplicates Precision ($D_P$) and Duplicates Recall ($D_R$) [12] are similar to the metrics *precision* and *recall* [22]. The only difference is that they check how many duplicate tasks the mined and reference models have in common with respect to each other, under the assumption that the number of task labels is the same for the discovered model and the reference model (which is the case in a typical mining setting).

### 4.2 Example Comparison using Metrics

We compare the discovered models depicted in Figure 1 using some of the validation metrics discussed in above. For those metrics that need a reference log and/or reference model, we use the log in Figure 2(a) and the model in Figure 2(b), respectively. The results of the comparison can be computed with the `Control Flow Benchmark` plug-in in ProM (cf. Section 6), and are given in Table 2.

**Table 2.** Overview of the presented *fitness*, *precision* and *generalization*, and *structure* metric values for the process models in Figure 1(a)–(f). If the assumptions of the metric are not fulfilled, and therefore no valid result can be obtained, "n.a." (not applicable) is given rather than the calculated value

| | *Alpha* | *Alpha++* | *Genetic* | *Dupl. GA* | *Heuristic* | *Petrify* |
|---|---|---|---|---|---|---|
| `PM` | 0.965 | 0.965 | 1.0 | 1.0 | 1.0 | 1.0 |
| $f$ | 0.995 | 0.995 | 1.0 | 1.0 | 1.0 | 1.0 |
| $PF_{complete}$ | 0.993 | 0.993 | 1.0 | 1.0 | 1.0 | 1.0 |
| $a'_B$ | 0.646 | 1.0 | 1.0 | 1.0 | 0.664 | 1.0 |
| $B_P$ | 0.964 | 0.993 | 1.0 | 1.0 | 0.972 | 1.0 |
| $B_R$ | 0.983 | 0.983 | 1.0 | 1.0 | 1.0 | 0.997 |
| *Causal Footprint* | 0.975 | 0.986 | 1.0 | n.a. | 0.988 | n.a. |
| $a'_S$ | 1.0 | 1.0 | 1.0 | 0.818 | 1.0 | 0.727 |
| $S_P$ | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.854 |
| $S_R$ | 0.938 | 1.0 | 1.0 | 0.938 | 0.938 | 1.0 |
| $D_P$ | 1.0 | 1.0 | 1.0 | 0.9 | 1.0 | 0.9 |
| $D_R$ | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

It is clear that the values of the different metrics are not directly comparable to each other as they measure different aspects of quality, and at a different level of granularity; they are even defined for different modeling formalisms. Nevertheless, they can highlight potential problems and provide an overall indication

of quality. It remains a continuous challenge to define good validation metrics that can be used to evaluate desired aspects of process models in an efficient, stable and analyzable way.

From the results in Table 2 we can see that: (i) All discovered models except the models returned by the *Alpha* and *Alpha++* miner *fit* the log (cf. metrics PM, $f$, and $PF_{complete}$). The algorithms *Alpha* and *Alpha++* have problems to return fitting models because they cannot correctly discover the *skipping* of tasks. Note that task "G" can be skipped in the model in Figure 2(b); (ii) Among the fitting models, the *Genetic*, *Duplicates GA* and *Petrify* miner returned also *precise* models (cf. metrics $B_P$ and $a'_B$). Note that there is a *long-term dependency* between task "B" and "E"; (iii) The *Petrify* miner returned an overly precise model, which does not allow for the trace *ACGHDFI* (cf. metric $B_R$); (iv) The *structure* of the models returned by the *Duplicates GA* and *Petrify* miner contains unnecessary duplicate and invisible tasks (cf. metrics $D_P$, $D_R$, and $a'_S$).

This demonstrates that the discussed metrics are able to successfully capture real deficiencies of the discovered models. According to the results in Table 2, all the discovered models in Figure 1 are relatively good models, among which the model discovered by the *Genetic* miner (cf. Figure 1(e)) seems to be the best solution for the example log in Figure 2(a).

## 5 Machine Learning Approach

From a theoretical point of view, process discovery is related to some work discussed in the Machine Learning (ML) domain. In [9, 15, 16, 23] the limits of inductive inference are explored. For example, in [16] it is shown that the computational problem of finding a minimum finite-state acceptor compatible with given data is NP-hard. Several of the more generic concepts discussed in these papers could be translated to the domain of process mining. It is possible to interpret the process discovery problem as an inductive inference problem specified in terms of rules, a hypothesis space, examples, and criteria for successful inference. However, despite the many relations with the work described in [9, 15, 16, 23] there are also many differences, e.g., we are mining at the net level rather than sequential or lower level representations (e.g., Markov chains, finite state machines, or regular expressions), and therefore process mining needs to deal with various forms of concurrency.

Furthermore, there is a long tradition of theoretical work dealing with the problem of inferring grammars out of examples: given a number of sentences (traces) out of a language, find the simplest model that can generate these sentences. There is a strong analogy with the process-mining problem: given a number of process traces, can we find the simplest process model that can generate these traces. A good overview of prominent computational approaches for learning different classes of formal languages is given in [21] and a special issue of the machine learning journal about this subject [19]. Many issues important in the language-learning domain are also relevant for process mining (i.e. learning

from only positive examples, how to deal with noise, measuring the quality of a model, etc.). However, as indicated before, an important difference between the grammar inference domain and the process-mining domain is the problem of concurrency in the traces: concurrency seems not relevant in the grammar inference domain.

Given the similarities between the problems addressed in the process mining and the ML domain, it appears to be beneficial to consider existing validation mechanisms used in the ML community and test their applicability in the process mining area. In the remainder of this section, we investigate how the well-known *k-fold cv setup* can be used for the evaluation of process discovery algorithms (Section 5.1), and apply the approach to the discovery algorithms used in Figure 1 (Section 5.2).

### 5.1 K-fold CV Setup for Process Mining

Within the ML community, there is a relatively simple experimental framework called *k-fold cross validation* [20]. Based on the observation that estimating the performance of the learned model on the learning material leads to overly optimistic results, a strict separation between training data and test (or validation) data is advocated. The following steps can be distinguished: The available data is divided into $k$ subsets numbered 1 to $k$. The ML-algorithm is trained $k$ times. In training $i$ ($1 \leq i \leq k$), subset $i$ is used as test material, the rest of the material, i.e., $\{1, ..., i-1, i+1, ..., k\}$, is used as learning material. The performance of the ML-algorithm on the current data set (or the performance of the definitive learned model) is estimated by the average (or otherwise combined) error over the $k$ **test** sets. The framework can also be used for parameter optimization, and for comparing the performance of different learning algorithms (using the paired T-test to calculate a confidence interval).

We want to investigate whether this validation framework is useful for the evaluation of process discovery algorithms. One can think of the concept to be learned as the process behavior, which should be captured by the discovered process model (the knowledge description). The starting point in a typical k-fold-cv experiment is a data set with *positive* and *negative* examples. However, during *process discovery* an event log only contains *positive* examples; negative examples are only available if the underlying model is known (impossible process behavior). To apply the k-fold cross validation framework in our process discovery setting, we have to answer the following questions: (i) *How to deal with the lack of negative examples?*, (ii) *How to partition the data in training and test material (so that both parts contain representative behavior)?*, and (iii) *How to calculate the error for each test run i?* Different options are possible and need to be investigated in more detail; in this paper only two illustrative combinations are chosen.

**(i) Negative examples.** We want to generate negative examples, for which we use the following simple solution: Starting with an event log called `EventLog` another event log (`EventLogRandom`) is generated by building $m$ random traces

of length $n$ (where $m$ is the number of traces in `EventLog` and $n$ is the number of different activities in event log `EventLog`). In the presence of a reference model, one could also use more sophisticated approaches that generate negative examples that better reflect really "forbidden" behavior. Note that without an explicit reference model, the log `EventLogRandom` may accidentally contain positive examples.

**(ii) Partitioning the data.** To partition the 1459 positive examples (cf. total number of instances) of the event log in Figure 2(a) into training and test data, we take two different approaches. First, we use $k = 10$ (a commonly used value for $k$), and randomly partition the log in 10 pieces of equal size. In this 10-fold cv experiment, we then use 9 of the 10 partitions as training data and the remaining part as test data. However, because we only have 5 different traces in this simple example log, each of the 10 partitions contains these 5 traces (in different frequencies), and, therefore, the test set does not contain really "new" data. Here, the running example seems too simple[6]. Therefore, we also chose another partitioning strategy, where $k$ is the number of different traces in the log (i.e., here $k = 5$) and each partition contains those instances that exhibit the same sequence of activities. In this 5-fold cv experiment, 4 out of these 5 partitions were then used as training data, and the remaining part as test data. A problem here is, however, that—because the example the log only contains the minimal number of traces needed to rediscover the model—important behavior is missing in the training data.

**(iii) Calculating the error.** For each test run $i$, we first discover a process model based on the training data. Then, we calculate the error *pos* based on the positive examples of the test set (should be parsed by the model), and the error *neg* based on the negative examples (should not be parsed by the model). The idea is that—while the model should account for enough *fitness* and *generalization* to parse the positive examples from the test set—negative examples should not be accepted frequently as otherwise the *precision* is low. To calculate the errors *pos* and *neg*, we used the fitness metric $f$ (cf. Section 4; other fitness metrics could be chosen). For each test run $i$, the overall error is then $(pos + (1 - neg))/2$, where *pos* and *neg* are fractions of positive and negative examples that "fit". The average error of all the $k$ test runs is taken as the result of the $k$-fold cv experiment. For example, for the *Heuristic* miner the overall error for each of the 10 test runs evaluates to (1.0 + (1.0-0.389))/2 = 0.805.

### 5.2 Example Comparison using the ML Approach

This section compares the discovered models depicted in Figure 1 using the setup described in Section 5.1. As explained earlier, the performance of the definite

[6] As indicated in Section 3.1, real-life logs often contain traces that are unique. Typically, traces are hardly reproduced in the log and logs are far from complete (i.e., only contain a fraction of the possible behavior). Therefore, the test set will contain also unseen (i.e., "new") traces in more realistic scenarios.

learned model is estimated based on the average error over the k **test** sets. The results of this comparison both for the 10-fold and the 5-fold cv experiment are given in Table 3.

**Table 3.** Overview of the evaluation results for 10-fold and 5-fold cv experiment

| | *Alpha* | *Alpha++* | *Genetic* | *Dupl. GA* | *Heuristic* | *Petrify* |
|---|---|---|---|---|---|---|
| 10-fold cv exp. | 0.814 | 0.818 | 0.813 | 0.795 | 0.805 | 0.787 |
| 5-fold cv exp. | 0.702 | 0.701 | 0.705 | 0.705 | 0.716 | 0.696 |

One of the advantages of the presented machine learning approach is that well-known statistical techniques can be applied to investigate whether one algorithm performs *significantly* better than others on a particular data set. For example, we performed a paired T-test for 15 combinations, where we compared the performance of all the 6 process discovery algorithms based on the data sets used for the 10-fold cv experiment in a pair-wise manner[7]. From this, we can conclude that—using the fitness measure $f$ and the earlier described experimental set-up—the results are all statistically significant using a 95% confidence interval, except for one combination (namely *Duplicates GA* and *Petrify*). The significance is most likely influenced by the fact that all partitions in the 10-fold cv experiment contained all the 5 different traces from the log (in different frequencies), and, therefore, almost all[8] the discovered models were identical for each of the $i$ test runs.

The results of the two cv experiments show that it is problematic to apply this technique in a setting where exact models should be discovered and only a limited amount of generalization (namely, detecting parallelism) is desired. For example, partitioning the data correctly is difficult. However, the approach seems attractive in situations where more abstraction is required to discover models from logs that contain noise, or less structured behavior. More research is needed to investigate this in detail.

## 6 Control Flow Benchmark Plug-in in ProM

The Process Mining (ProM) framework is an extensible tool suite that supports a wide variety of process mining techniques in the form of plug-ins [2]. To directly support the evaluation and comparison of (discovered) process models in the ProM tool, we implemented a `Control Flow Benchmark` plug-in[9] that computes some of the metrics presented in Section 4, and provides an integrated

[7] If we compare each of the 6 process mining algorithms with every other algorithm then this yields $\binom{6}{2} = \frac{6!}{2!\cdot(6-2)!} = 15$ pairs.

[8] The *Duplicates GA* miner is sensitive to the frequency of traces in the training set, and therefore discovered different models.

[9] The `Control Flow Benchmark` plug-in can be downloaded together with ProM and and all the example files used in this paper from *http://www.processmining.org*.

view on the results. Note that this plug-in can be easily extended: only the `BenchmarkMetric` interface needs to be implemented and the new metric can be added. The approach described in Section 5.1 is currently only partially supported by ProM, i.e., only the calculation of the error for each test run $i$ (iii) can be achieved with the help of the `Control Flow Benchmark` plug-in while the generation of negative examples (i) and the partitioning of the log into training and test data (ii) is not yet supported.

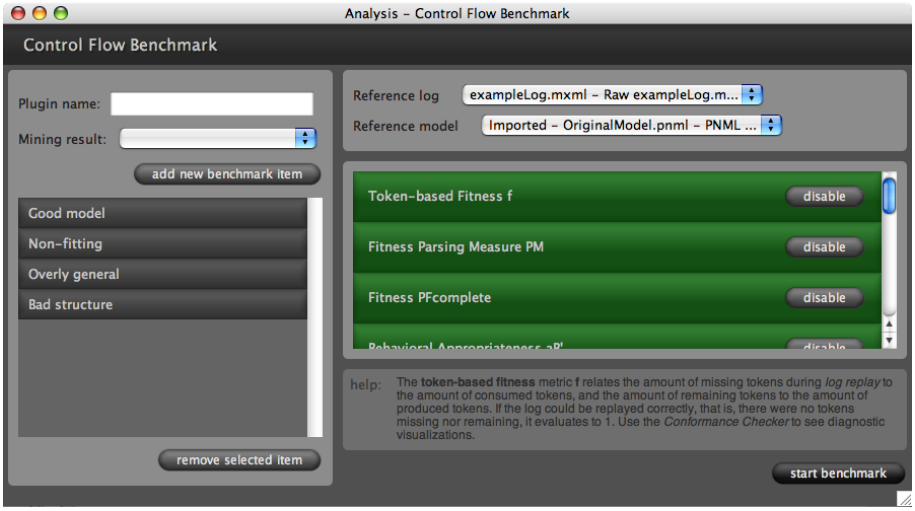



**Fig. 4.** Screenshot of the `Control Flow Benchmark` settings. A set of models can be selected to be compared with respect to a log and/or a reference model based on a number of different metrics

Figure 4 depicts a screenshot of the `Control Flow Benchmark` settings. The plug-in accepts a *reference log*, a *reference model*, and an arbitrary number of *benchmark items* (e.g., discovered process models) as a Petri net. Furthermore, the user can select the *metrics* to be calculated. In Figure 4 the four models from Figure 2 were added as benchmark items: Figure 2(b) as "Good model", Figure 2(c) as "Non-fitting", Figure 2(d) as "Overly general", and Figure 2(e) as "Bad structure". Furthermore, the log shown in Figure 2(a) is used as the reference log, and the model from Figure 2(b) is used as the reference model.

As discussed in Section 4, some metrics require a reference model to be present while others do not. Similarly, there are metrics that only compare the benchmark items to the reference model, and, therefore, do not need a reference log. If the input needed for a certain metric is not provided, the metric will remain disabled. Upon pressing the *start benchmark* button, the plug-in will transparently establish the mapping between the tasks in a model and the events in the log, potentially convert the Petri net to another modeling formalism, calculate the selected metrics, and present the user with the results. All the

benchmark metrics return values between 0 (interpreted as the "worst" value) and 1 (interpreted as the "best" value). Furthermore, each metric is expected to check its assumptions and to indicate the result being "invalid" if pre-conditions are not met.

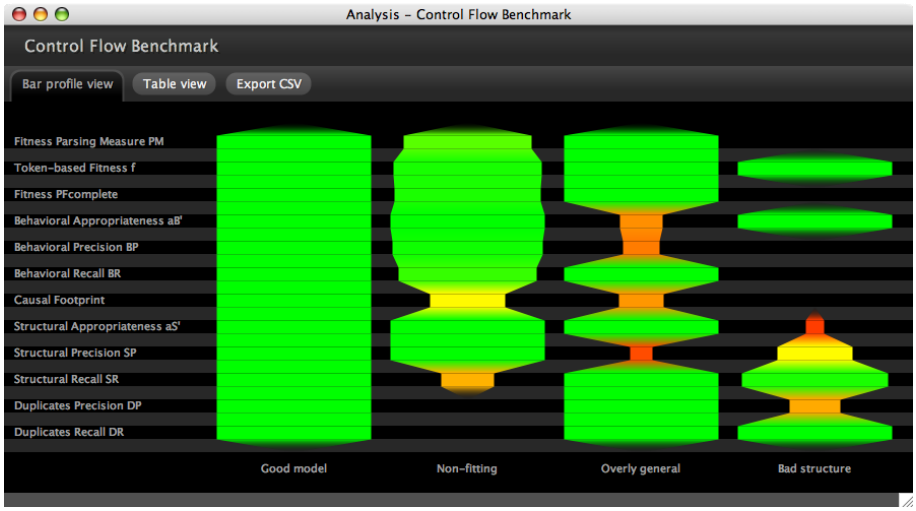
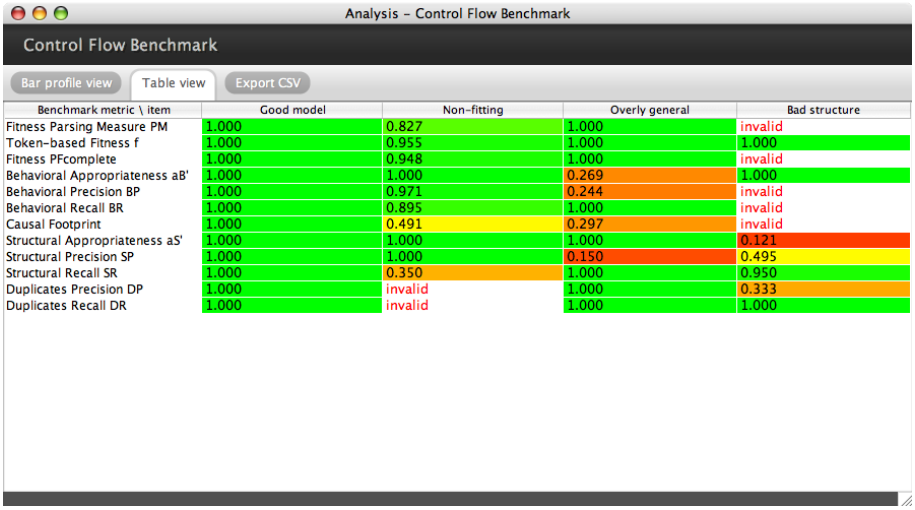



**Fig. 5.** Screenshot of the Bar Profile view in the `Control Flow Benchmark` plug-in in ProM. The visualization of the metric values provides an overview, and makes it easy to spot problems

The first result view is the *Bar Profile view* (see screenshot in Figure 5). It visualizes the calculated values along a so-called bar profile in a fixed order. Green and wide segments resemble "good" values while red and narrow segments resemble "problematic" values according to the corresponding metric. Invalid values are left out (i.e., the bar profile will have a gap at this place). This view is intended to provide a graphical overview that makes it easy to spot problematic areas, which can be subsequently inspected in further detail.

For the extreme models from Figure 2 we can clearly see some differences. The "Good model" shows a consistently green and wide profile, while the "Non-fitting" model has a reduced fitness (although the three fitness values `PM`, $f$, and $PF_{complete}$ are still relatively high because the most frequent trace is correctly captured) and less behavior and less connections than the reference model (cf. *Causal Footprint* and Structural Recall $S_R$). The "Overly general" model exhibits extra behavior (cf. Behavioral Appropriateness $a'_B$, Behavioral Recall $B_R$, and *Causal Footprint*) and has more connections than the reference model (cf. Structural Precision $S_P$). Finally, the model with the "Bad structure" shows a reduced Structural Appropriateness $a'_S$ as it has many unnecessary duplicate tasks. Similarly, the Duplicates Precision $D_P$ is low.




| Benchmark metric \ item | Good model | Non-fitting | Overly general | Bad structure |
|---|---|---|---|---|
| Fitness Parsing Measure PM | 1.000 | 0.827 | 1.000 | invalid |
| Token-based Fitness f | 1.000 | 0.955 | 1.000 | 1.000 |
| Fitness PFcomplete | 1.000 | 0.948 | 1.000 | invalid |
| Behavioral Appropriateness aB' | 1.000 | 1.000 | 0.269 | 1.000 |
| Behavioral Precision BP | 1.000 | 0.971 | 0.244 | invalid |
| Behavioral Recall BR | 1.000 | 0.895 | 1.000 | invalid |
| Causal Footprint | 1.000 | 0.491 | 0.297 | invalid |
| Structural Appropriateness aS' | 1.000 | 1.000 | 1.000 | 0.121 |
| Structural Precision SP | 1.000 | 1.000 | 0.150 | 0.495 |
| Structural Recall SR | 1.000 | 0.350 | 1.000 | 0.950 |
| Duplicates Precision DP | 1.000 | invalid | 1.000 | 0.333 |
| Duplicates Recall DR | 1.000 | invalid | 1.000 | 1.000 |

**Fig. 6.** Screenshot of the Table view in the `Control Flow Benchmark` plug-in in ProM. The concrete metric values can be inspected in detail and exported to a CSV file

The view can be changed to *Table view* (see screenshot in Figure 6) for inspecting the detailed results. The exact values are shown for each benchmark item, or "invalid" is given if it could not be computed for a certain case (for example, the *Causal Footprint* metric cannot deal with the duplicate tasks in the model with the "Bad Structure"). Finally, the results can be exported as Comma Separated Values (CSV) and analyzed/visualized using different analysis tools.

## 7 Conclusion

Adequate validation techniques in the process mining domain are needed to evaluate and compare discovered process models both in research and practice. We have outlined an evaluation framework and introduced evaluation dimensions for discovered process models. Furthermore, we described two evaluation strategies, and presented a `Control Flow Benchmark` plug-in in the ProM framework, which can be used to evaluate models with these two approaches.

The metrics discussed in the context of the first approach can be directly applied to evaluate the quality of a process model. Nevertheless, many obstacles such as bridging the gap between different modeling languages, defining good validation criteria and good metrics that capture them remain. The second approach seems especially promising in less structured environments where more abstraction is required, and should be subject to further research. Finally, completely new evaluation approaches might help to move towards a common validation methodology for process mining techniques. Moreover, a comprehensive set of benchmark examples (ideally containing both artificial and real-life data) is needed.

## Acknowledgements

This research is supported by Technology Foundation STW, EIT, SUPER, NWO, and the IOP program of the Dutch Ministry of Economic Affairs. The authors would also like to thank all ProM developers for their on-going work on process mining techniques.

## References

1. W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters. Process Equivalence: Comparing Two Process Models Based on Observed Behavior. In S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors, *BPM 2006*, volume 4102 of *Lecture Notes in Computer Science*, pages 129–144. Springer-Verlag, Berlin, 2006.
2. W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, Berlin, 2007.
3. W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713–732, 2007.
4. W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach using Transition Systems and Regions. BPM Center Report BPM-06-30, BPMcenter.org, 2006.
5. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
6. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
7. 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.
8. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
9. D. Angluin and C.H. Smith. Inductive Inference: Theory and Methods. *Computing Surveys*, 15(3):237–269, 1983.
10. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
11. J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.
12. A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2006.

13. A.K. Alves de Medeiros and C.W. Guenther. Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In K. Jensen, editor, *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 177–190, 2005.
14. B.F. van Dongen, J. Mendling, and W.M.P. van der Aalst. Structural Patterns for Soundness of Business Process Models. In *EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 116–128, Washington, DC, USA, 2006. IEEE Computer Society.
15. E.M. Gold. Language Identfication in the Limit. *Information and Control*, 10(5):447–474, 1967.
16. E.M. Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3):302–320, 1978.
17. G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.
18. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
19. P. Langley, editor. *Grammar Induction*, Special issue of Machine Learning, Volume 2, Number 1. Kluwer Academic Publishers, Boston/Dordrecht, 1987.
20. T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
21. R. Parekh and V. Honavar. Automata Induction, Grammar Inference, and Language Acquisition. In Dale, Moisl, and Somers, editors, *Handbook of Natural Language Processing*. New York: Marcel Dekker, 2000.
22. S.S. Pinter and M. Golani. Discovering Workflow Models from Activities Lifespans. *Computers in Industry*, 53(3):283–296, 2004.
23. L. Pitt. Inductive Inference, DFAs, and Computational Complexity. In K.P. Jantke, editor, *Proceedings of International Workshop on Analogical and Inductive Inference (AII)*, volume 397 of *Lecture Notes in Computer Science*, pages 18–44. Springer-Verlag, Berlin, 1889.
24. A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Accepted for publication in Information Systems: DOI 10.1016/j.is.2007.07.001*.
25. A. Rozinat, I.S.M. de Jong, C.W. Günther, and W.M.P. van der Aalst. Process Mining of Test Processes: A Case Study. BETA Working Paper Series, WP 220, Eindhoven University of Technology, Eindhoven, 2007.
26. A. Rozinat, R.S. Mans, and W.M.P. van der Aalst. Mining CPN Models: Discovering Process Models With Data from Event Logs. In K. Jensen, editor, *Proceedings of the Seventh Workshop on the Practical Use of Coloured Petri Nets and CPN Tools*, pages 57–76. University of Aarhus, Denmark, 2006.
27. 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.
28. A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. Alves de Medeiros. Process Mining with HeuristicsMiner Algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology, Eindhoven, 2006.
29. L. Wen, J. Wang, and J.G. Sun. Detecting Implicit Dependencies Between Tasks from Event Logs. In *Asia-Pacific Web Conference on Frontiers of WWW Research and Development (APWeb 2006)*, Lecture Notes in Computer Science, pages 591–603. Springer, 2006.