Automated Discovery of Process Models from Event Logs: Review and Benchmark

Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, Allar Soo

Abstract—Process mining allows analysts to exploit logs of historical executions of business processes to extract insights regarding the actual performance of these processes. One of the most widely studied process mining operations is automated process discovery. An automated process discovery method takes as input an event log, and produces as output a business process model that captures the control-flow relations between tasks that are observed in or implied by the event log. Various automated process discovery methods have been proposed in the past two decades, striking different tradeoffs between scalability, accuracy and complexity of the resulting models. However, these methods have been evaluated in an ad-hoc manner, employing different datasets, experimental setups, evaluation measures and baselines, often leading to incomparable conclusions and sometimes unrepeatable results due to the use of closed datasets. This article provides a systematic review and comparative evaluation of automated process discovery methods, using an open-source benchmark and covering twelve publicly-available real-life event logs, twelve proprietary real-life event logs, and nine quality metrics. The results highlight gaps and unexplored tradeoffs in the field, including the lack of scalability of some methods and a strong divergence in their performance with respect to the different quality metrics used.

Index Terms—Process mining, automated process discovery, survey, benchmark.

1 INTRODUCTION

Modern information systems maintain detailed trails of the business processes they support, including records of key process execution events, such as the creation of a case or the execution of a task within an ongoing case. Process mining techniques allow analysts to extract insights about the actual performance of a process from collections of such event records, also known as event logs [1]. In this context, an event log consists of a set of traces, each trace itself consisting of the sequence of events related to a given case.

One of the most widely investigated process mining operations is automated process discovery. An automated process discovery method takes as input an event log, and produces as output a business process model that captures the control-flow relations between tasks that are observed in or implied by the event log. In order to be useful, such automatically discovered process models must accurately reflect the behavior recorded in or implied by the log. Specifically, the process model discovered from an event log should be able to: (i) generate each trace in the log, or for each trace in the log, generate a trace that is similar to it; (ii) generate traces that are not in the log but that are identical or similar to traces of the process that produced the log; and (iii) not generate other traces [2]. The first property is called fitness, the second generalization and the third precision. In addition, the discovered process model should be as simple as possible, a property that is usually quantified via complexity measures.

The problem of automated discovery of process models from event logs has been intensively researched in the past two decades. Despite a rich set of proposals, state-of-the-art automated process discovery methods suffer from two recurrent deficiencies when applied to real-life logs [3]: (i) they produce large and spaghetti-like models; and (ii) they produce models that either poorly fit the event log (low fitness) or over-generalize it (low precision or low generalization). Striking a tradeoff between these quality dimensions in a robust manner has proved to be a difficult problem.

So far, automated process discovery methods have been evaluated in an ad hoc manner, with different authors employing different datasets, experimental setups, evaluation measures and baselines, often leading to incomparable conclusions and sometimes unrepeatable results due to the use of non-publicly available datasets. This work aims at filling this gap by providing: (i) a systematic review of automated process discovery methods; and (ii) a comparative evaluation of seven implementations of representative methods, using an open-source benchmark framework and covering twelve publicly-available real-life event logs, twelve proprietary real-life event logs, and nine quality metrics covering all four dimensions mentioned above (fitness, precision, generalization and complexity), as well as execution time.

The outcomes of this research are a classified inventory of automated process discovery methods and a benchmark designed to enable researchers to empirically compare new automated process discovery methods against existing ones.
in a unified setting. The benchmark is provided as an open-source command-line Java application to enable researchers to replicate the reported experiments with minimal configuration effort.

The rest of the article is structured as follows. Section 2 describes the search protocol used for the systematic literature review, whereas Section 3 classifies the methods identified in the review. Then, Section 4 introduces the experimental benchmark and results, whereas Section 5 discusses the overall findings and Section 6 acknowledges the threats to the validity of the study. Finally, Section 7 relates this work to previous reviews and comparative studies in the field and Section 8 concludes the paper and outlines future work directions.

2 SEARCH PROTOCOL

In order to identify and classify research in the area of automated process discovery, we conducted a Systematic Literature Review (SLR) through a scientific, rigorous and replicable approach as specified by Kitchenham [4].

First, we formulated a set of research questions to scope the search, and developed a list of search strings. Next, we ran the search strings on different data sources. Finally, we applied inclusion criteria to select the studies retrieved through the search.

2.1 Research questions

The objective of our SLR is to analyse research studies related to automated (business) process discovery from event logs. This means, for example, that methods performing only trace clustering or log-filtering are not considered in our analysis. To this aim, we formulated the following research questions:

RQ1 What methods exist for automated (business) process discovery from event logs?
RQ2 What type of process models can be discovered by these methods, and in which modeling language?
RQ3 Which semantic can be captured by a model discovered by these methods?
RQ4 What tools are available to support these methods?
RQ5 What type of data has been used to evaluate these methods, and from which application domains?

RQ1 is the core research question, which aims at identifying existing methods to perform (business) process discovery from event logs. The other questions allow us to identify a set of classification criteria. Specifically, RQ2 categorizes the output of a method on the basis of the type of process model discovered (i.e., procedural, declarative or hybrid), and the specific modeling language employed (e.g., Petri nets, BPMN, Declare). RQ3 delves into the specific semantic constructs supported by a method (e.g., exclusive choice, parallelism, loops). RQ4 explores what tool support the different methods have, while RQ5 investigates how the methods have been evaluated and in which application domains.

2.2 Search string development and validation

Next, we developed four search strings by deriving keywords from our knowledge of the subject matter. We first determined that the term “process discovery” is a very generic term which would allow us to retrieve the majority of methods in this area. Furthermore, we used “learning” and “workflow” as synonyms of “discovery” and “process” (respectively). This led to the following four search strings: (i) “process discovery”, (ii) “workflow discovery”, (iii) “process learning”, (iv) “workflow learning”. We intentionally excluded the terms “automated” and “automating” in the search strings, because these terms are often not explicitly used.

However, this led to retrieving many more studies than those that actually focus on automated process discovery, e.g., studies on process discovery via workshops or interviews. Thus, if a query on a specific data source returned more than one thousand results, we refined it by combining the selected search string with the term “business” or “process mining” to obtain more focused results, e.g., “process discovery AND process mining” or “process discovery AND business”. According to this criterion, the final search strings used for our search were the following:

i. “process discovery AND process mining”
ii. “process learning AND process mining”
iii. “workflow discovery”
iv. “workflow learning”

First, we applied each of the four search strings to Google Scholar, retrieving studies based on the occurrence of one of the search strings in the title, the keywords or the abstract of a paper. Then, we used the following six popular academic databases: Scopus, Web of Science, IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, to double check the studies retrieved from Google Scholar. We noted that this additional search did not return any relevant study that was not already discovered in our primary search. The search was completed in December 2017.

2.3 Study selection

As a last step, as suggested by [5], [6], [7], [8], we defined inclusion criteria to ensure an unbiased selection of relevant studies. To be retained, a study must satisfy all the following inclusion criteria.

IN1 The study proposes a method for automated (business) process discovery from event logs. This criterion draws the borders of our search scope and it is direct consequence of RQ1.
IN2 The study proposes a method that has been implemented and evaluated. This criterion let us exclude methods whose properties have not been evaluated nor analyzed.
IN3 The study is published in 2011 or later. Earlier studies have been reviewed and evaluated by De Weerdt et al. [3], therefore, we decided to focus only on the successive studies. Nevertheless, we performed a mapping of the studies assessed in 2011 [3] and their successors (when applicable), cf. Table 1.
IN4 The study is peer-reviewed. This criterion guarantees a minimum reasonable quality of the studies included in this SLR.

IN5 The study is written in English.

| α, α+, α++ | AGNEs Miner | [9], [10], [11] |
|-------------|-------------|-----------------|
| (DT) Genetic Miner | [14], [15] | Evolutionary Tree Miner | [16] |
| Heuristics Miner | [17], [18] | Heuristics Miner | [19], [20], [21], [22] |
| ILP Miner | [23] | Hybrid ILP Miner | [24], [25] |

TABLE 1: Methods assessed by De Weerdt et al. [3] (left) and the respective successors (right).

Inclusion criteria IN3, IN4 and IN5 were automatically applied through the configuration of the search engines. After the application of the latter three inclusion criteria, we obtained a total of 2,820 studies. Then, we skimmed title and abstract of these studies to exclude those studies that were clearly not compliant with IN1. As a result of this first iteration, we obtained 344 studies.

Then, we assessed each of the 344 studies against the inclusion criteria IN1 and IN2. The (combined) assessment of IN1 and IN2 on the 344 selected studies was performed independently by two authors of this paper, whose decisions were compared in order to resolve inconsistencies with the mediation of a third author, when needed. The assessment of IN1 was based on the accurate reading of the abstract, introduction and conclusion. Whilst, to determine whether a study fulfilled IN2, we relied on the findings reported in the evaluation of the studies. As result of the iterations, we found 86 studies matching the five inclusion criteria.

However, many of these studies refer to the same automated process discovery method, i.e., some studies are either extensions, optimization, preliminaries or generalization of another study. For such reason, we decided to group the studies by either the last version or the most general one. When in doubt, the grouping decision was taken after a consultation with the main author. At the end of this process, as shown in Table 2, 35 main groups of discovery algorithms were identified.

The excel sheet available at https://drive.google.com/open?id=1fW8WLXSw2ntiPul3XVgsDIcJUrj762G reports the 344 studies found after the first iteration. For each of these studies, we explicitly refer to the inclusion criterion fulfilled for the study to be selected. Furthermore, each selected study has a reference to the group it belongs to (unless it is the main study).

Fig. 1 shows how the studies are distributed over time. We can see that the interest in the topic of automated process discovery grew over time with a sharp increase between 2013 and 2014, and lately declining close to the average number of studies per year.

2) Semantic captured in procedural models: parallelism (AND), exclusive choice (XOR), inclusive choice (OR), loop — RQ3
3) Type of implementation (standalone or plug-in, and tool accessibility) — RQ4
4) Type of evaluation data (real-life, synthetic or artificial log, where a synthetic log is one generated from a real-life model while an artificial log is one generated from an artificial model) and application domain (e.g., insurance, banking, healthcare) — RQ5.

This information is summarized in Table 2. Each entry in this table refers to the main study of the 35 groups found. Also, we cited all the studies that relate to the main one. Collectively, the information reported by Table 2 allows us to answer the first research question: “what methods exist for automated process discovery?”

In the remainder of this section, we proceed with surveying each main study method along the above classification dimensions, to answer the other research questions.

3.1 Model type and language (RQ2)

The majority of methods (26 out of 35) produce procedural models. Six approaches [25], [33], [46], [49], [75], [85] discover declarative models in the form of Declare constraints, whereas [26] produces declarative models using the WoMan formalism. The methods in [62], [68] are able to discover hybrid models as a combination of Petri nets and Declare constraints.

Regarding the modeling languages of the discovered process model, we notice that Petri nets is the predominant one. However, more recently, we have seen the appearance of methods that produce models in BPMN, a language that is more practically-oriented and less technical than Petri nets. This denotes a shift in the target audience of these methods, from data scientists to practitioners, such as business analysts and decision managers. Other technical languages employed, besides Petri nets, include Causal nets, State machines and simple Directed Acyclic Graphs,
TABLE 2: Overview of the 35 primary studies resulting from the search (ordered by year and author).

| Method | Main study | Year | Related studies | Model type | Model language | Semantic Constructs | Implementation | Evaluation |
|--------|------------|------|-----------------|------------|----------------|--------------------|----------------|-----------|
| HK     | Huang and Kumar [25] | 2012 |                  | Procedural | Petri nets | ✓ | Standalone | ✓ | ✓ |
| Declare Miner | Maggi et al. [26] | 2012 |                  | Procedural | Declare | ✓ | Apromore, Standalone | ✓ | ✓ |
| MINERlib | Di Cicco, Mele [37] | 2013 |                  | Procedural | Declare | ✓ | Apromore, Standalone | ✓ | ✓ |
| Inductive Miner - Infrequent | Lernani et al. [38] | 2013 |                  | Procedural | Decision trees | ✓ | ProM | ✓ | ✓ |
| Data-aware Declare Miner | Maggi et al. [39] | 2013 |                  | Procedural | Declare | ✓ | Apromore, Standalone | ✓ | ✓ |
| ProM-Fusion Miner | Niv, Kadosh [40] | 2014 |                  | Procedural | Decision trees | ✓ | ProM | ✓ | ✓ |
| Evolutionary Declare Miner | van den Breukelen [41] | 2014 |                  | Procedural | Declare | ✓ | Apromore, Standalone | ✓ | ✓ |
| Evolutionary Declare Miner | Butu et al. [42] | 2014 |                  | Procedural | Process trees | ✓ | ProM | ✓ | ✓ |
| Aim | Carmona, Carmona [43] | 2014 |                  | Procedural | Petri nets | ✓ | Apromore, Standalone | ✓ | ✓ |
| WeMin | Carmona, Carmona [44] | 2014 |                  | Procedural | Declare | ✓ | Apromore, Standalone | ✓ | ✓ |
| Hybrid Miner | Maggi et al. [45] | 2014 |                  | Hybrid | Declare + Petri nets | ✓ | ProM | ✓ | ✓ |
| Competition Miner | Bulich et al. [46] | 2014 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| Directed Acyclic Graphs | Vanlauwe et al. [47] | 2014 |                  | Procedural | Declare | ✓ | Apromore, Standalone | ✓ | ✓ |
| Fusion Miner | De Smath et al. [48] | 2015 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| CNMiner | Greco et al. [49] | 2015 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| Sphynx | Cao et al. [50] | 2015 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| Maximal Pattern Mining | Lepeta et al. [51] | 2015 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| Non-Atomic Declare Miner | Bernardi et al. [52] | 2016 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| RegPFA | Bredero et al. [53] | 2016 |                  | Procedural | Petri nets | ✓ | Apromore, Standalone | ✓ | ✓ |
| BPMN Miner | van Eck et al. [54] | 2016 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| TAU Miner | Li et al. [55] | 2016 |                  | Procedural | Petri nets | ✓ | Apromore, Standalone | ✓ | ✓ |
| Pimpliner | Molitor et al. [56] | 2016 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| SQL Miner | Slobswa et al. [57] | 2016 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| ProM-M | Song et al. [58] | 2016 |                  | Procedural | Petri nets | ✓ | Apromore, Standalone | ✓ | ✓ |
| CoMiner | Tanpa-Fuma et al. [59] | 2016 |                  | Procedural | Petri nets | ✓ | Apromore, Standalone | ✓ | ✓ |
| ProM-P | van den Breukelen [60] | 2017 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| Split Miner | van den Breukelen [61] | 2017 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| Stage miner | Nguyen et al. [62] | 2017 |                  | Procedural | BPMN | ✓ | Apromore, Standalone | ✓ | ✓ |
| Decomposed Process Miner | Verbraak, van den Breukelen [63] | 2017 |                  | Procedural | Petri nets | ✓ | ProM | ✓ | ✓ |
| HyProP Miner | van den Breukelen et al. [64] | 2017 |                  | Procedural | Petri nets | ✓ | ProM | ✓ | ✓ |

While Declare is the most commonly-used language when producing declarative models.

**Petri nets.** In [25], the authors describe an algorithm to extract block-structured Petri nets from event logs. The algorithm works by first building an adjacency matrix between all pairs of tasks and then analyzing the information in it to extract block-structured models consisting of basic sequence, choice, parallel, loop, optional and self-loop structures as building blocks. The method has been implemented in a standalone tool called HK.

The method presented in [12] is based on the α$ algorithm, which can discover invisible tasks involved in non-free-choice constructs. The algorithm is an extension of the well-known α algorithm, one of the very first algorithms for automated process discovery, originally presented in [1].

In [96], the authors propose a generic divide-and-conquer framework for the discovery of process models from large event logs. The method allows to part the event log in smaller logs and discover a model from each of them. The output is then assembled from all the models discovered from the sublogs. The method aims to produce high-quality models reducing overall the complexity. The preliminary studies [97], [98], [99], [100], [101] widely illustrate the idea of splitting a large event log into a collection of smaller logs to improve the performance of a discovery algorithm.

Van Zelst et al. [24], [102], [103] propose an improvement of the ILP miner implemented in [23], their method is based on hybrid variable-based regions. Through hybrid variable-based regions, it is possible to vary the number of variables used within the ILP problems being solved. Using a different number of variables has an impact on the average computation time for solving the ILP problem.

In [97], [78], the authors propose an approach that allows the discovery of Petri nets using the theory of grammatical inference. The method has been implemented as a standalone application called RegPFA.

The approach proposed in [87] is based on the observation that activities with no dependencies in an event log can be executed in parallel. In this way, this method can discover process models with concurrency even if the logs fail to meet the completeness criteria. The method has been implemented in a tool called ProM-D.

In [55], the authors propose the use of numerical abstract domains for discovering Petri nets from large logs while guaranteeing formal properties of the discovered models. The technique guarantees the discovery of Petri nets that can reproduce every trace in the log and that are minimal in describing the log behavior.

The approach introduced in [88] addresses the problem of discovering sound Workflow nets from incomplete logs. The method is based on the concept of invariant occurrence between activities, which is used to identify sets of activities (named conjoint occurrence classes) that can be used to infer the behaviors not exhibited in the log.

In [83], the authors leverage data carried by tokens in the execution of a business process to track the state changes in the so-called token logs. This information is used to improve the performance of standard discovery algorithms.

**Process trees.** The Inductive Miner [38] and the Evolutionary Tree Miner [16] are both based on the extraction of process trees from an event log. Concerning the former, many different variants have been proposed during the last years, but its first appearance is in [99]. Successively, since that method was unable to deal with infrequent behavior, an improvement was proposed in [38], which efficiently drops infrequent behavior from logs, still ensuring that the discovered model is behaviorally correct (sound) and highly fitting. Another variant of the Inductive Miner is presented in [40]. This variant can minimize the impact of incompleteness of the input logs. In [42], the authors discuss ways of systematically treating lifecycle information in the discovery task. They introduce a process discovery technique that is able to handle lifecycle data to distinguish between concurrency and interleaving. The method proposed in [41] provides a graphical support for navigating the discovered model and the one described in [45] can deal with cancelation or error-handling behaviors (i.e., with logs...
presented in [43] and [44] combines scalability with quality guarantees. It can be used to mine large event logs and produces sound models.

In [16], Buijs et al. introduce the Evolutionary Tree Miner. This method is based on a genetic algorithm that allows the user to drive the discovery process based on preferences with respect to the four quality dimensions of the discovered model: fitness, precision, generalization, and complexity. The importance of these four dimensions and how to address their balance in process discovery is widely discussed in the related studies [50], [51], [52], [53], [54].

Causal nets. Greco et al. propose a discovery method that returns causal nets [69], [70]. A causal net is a net where only the causal relation between activities in a log is represented. The proposed method encodes causal relations gathered from an event log and if available, background knowledge in terms of precedence constraints over the topology of the resulting process models. A discovery algorithm is formulated in terms of reasoning problems over precedence constraints.

In [71], the authors propose a method for automated process discovery using Maximal Pattern Mining where they discover recurrent sequences of events in the traces of the log. Starting from these patterns they build process models in the form of causal nets.

ProDiGen, a standalone miner by Vazquez et al. [73], [74], allows users to discover causal nets from event logs using a genetic algorithm. The algorithm is based on a fitness function that takes into account completeness, precision and complexity and specific crossover and mutation operators.

Another method that produces causal nets is the Proximity Miner, presented in [89], [90]. This method extracts behavioral relations between the events of the log which are then enhanced using inputs from domain experts.

Finally, in [95], the authors propose a method to discover causal nets that optimizes the scalability and interpretability of the outputs. The process under analysis is decomposed into a set of stages, such that each stage can be mined separately. The proposed technique discovers a stage decomposition that maximizes modularity.

State machines. The CSM Miner, discussed in [81], [82], discovers state machines from event logs. Instead of focusing on the events or activities that are executed in the context of a particular process, this method concentrates on the states of the different process perspectives and discover how they are related with each other. These relations are expressed in terms of Composite State Machines. The CSM Miner provides an interactive visualization of these multiperspective state-based models.

BPMN models. In [80], Conforti et al. present the BPMN Miner, a method for the automated discovery of BPMN models containing sub-processes, activity markers such as multi-instance and loops, and interrupting and non-interrupting boundary events (to model exception handling). The method has been subsequently improved in [79] to make it robust to noise in event logs.

Another method to discover BPMN models has been presented in [72]. In this approach, a hierarchical view on process models is formally specified and an evolution strategy is applied on it. The evolution strategy, which is guided by the diversity of the process model population, efficiently finds the process models that best represent a given event log.

A further method to discover BPMN models is the Dynamic Constructs Competition Miner [63], [65], [66]. This method extends the Constructs Competition Miner presented in [64]. The method is based on a divide-and-conquer algorithm which discovers block-structured process models from logs.

In [22], the authors propose a discovery method that produces simple process models with low branching complexity and consistently high and balanced fitness, precision and generalization. The approach combines a technique to filter the directly-follows graph induced by an event log, with an approach to identify combinations of split gateways that accurately capture the concurrency, conflict and causal relations between neighbors in the directly-follows graph.

Fodina [93], [94] is a discovery method based on the Heuristics Miner [18]. However, differently from the Heuristics Miner, Fodina is more robust to noisy data, is able to discover duplicate activities, and allows for flexible configuration options to drive the discovery according to end user inputs.

In [21], the authors present the Flexible Heuristics Miner. This method can discover process models containing non-trivial constructs but with a low degree of block structuredness. At the same time, the method can cope well with noise in event logs. The discovered models are a specific type of Heuristics nets where the semantics of splits and joins is represented using split/join frequency tables. This results in easy to understand process models even in the presence of non-trivial constructs and log noise. The discovery algorithm is based on that of the original Heuristics Miner method [18]. In [22], the method presented in [21] has been improved as anomalies were found concerning the validity and completeness of the resulting process model. The improvements have been implemented in the Updated Heuristics Miner. A data-aware version of the Heuristics Miner that takes into consideration data attached to events in a log has been presented in [21]. Finally, in [13], [91], the authors propose an improvement of the Heuristics Miner algorithm to separate the objective of producing accurate models and that of ensuring their structuredness and soundness. Instead of directly discovering a structured process model, the approach first discovers accurate, possibly unstructured (and unsound) process models, and then transforms the resulting process model into a structured (and sound) one.

Declarative models. In [27], the authors present the first basic approach for mining declarative process models expressed using Declare constraints [104], [105]. This approach was improved in [26] using a two-phase approach. The first phase is based on an apriori algorithm used to identify frequent sets of correlated activities. A list of candidate constraints is built on the basis of the correlated activity sets. In the second phase, the constraints are checked by replaying the log on specific automata, each accepting only those traces that are compliant to one constraint. Those constraints satisfied by a percentage of traces higher than
a user-defined threshold, are discovered. Other variants of
the same approach are presented in [28], [29], [30], [31], [32].
The technique presented in [28] leverages prior knowledge to
to guide the discovery task. In [29], the approach is adapted
to be used in cross-organizational environments in which
different organizations execute the same process in different
variants. In [30], the author extends the approach to discover
metric temporal constraints, i.e., constraints taking into account
the time distance between events. Finally, in [31], [32], the
authors propose mechanisms to reduce the execution
times of the original approach presented in [26].
MINERful [33], [34], [35] discovers Declare constraints
using a two-phase approach. The first phase computes
statistical data describing the occurrences of activities and
their interplay in the log. The second one checks the validity
of Declare constraints by querying such a statistic data
structure (knowledge base). In [36], [37], the approach is ex-
tended to discover target-branched Declare constraints, i.e.,
constraints in which the target parameter is the disjunction
of two or more activities.

The approach presented in [46] is the first approach for
the discovery of Declare constraints with an extended
semantics that take into consideration data conditions. The
data-aware semantics of Declare presented in this paper is
based on first-order temporal logic. The method presented in
[27], [28] is based on the use of discriminative rule mining
to determine how the characteristics of the activity lifecycles
in a business process influence the validity of a Declare
constraint in that process.

Other approaches for the discovery of Declare con-
straints have been presented in [49], [50]. In [49], the authors
present the Evolutionary Declare Miner that implements the
discovery task using a genetic algorithm. The SQLMiner,
presented in [50], is based on a mining approach that
directly works on relational event data by querying a log
with standard SQL. By leveraging database performance
technology, the mining procedure is extremely fast. Queries
can be customized and cover process perspectives beyond
control flow [50].

The WoMan framework, proposed by Ferilli in [56] and
further extended in the related studies [57], [58], [59], [60],
[61], includes a method to learn and refine process models
from event logs, by discovering first-order logic constraints.
It guarantees incrementality in learning and adapting the
models, the ability to express triggers and conditions on the
process tasks and efficiency.

Further approaches. In [47], [48], the authors introduce a
monitoring framework for automated process discovery. A
monitoring context is used to extract traces from relational
event data and attach different types of metrics to them.
Based on these metrics, traces with certain characteristics
can be selected and used for the discovery of process models
expressed as directly-follows graphs.

Vasilecas et al. [62], present a method for the extraction of
directed acyclic graphs from event logs. Starting from these
graphs, they generate Bayesian belief networks, one of the
most common probabilistic models, and use these networks
to efficiently analyze business processes.

In [84], the authors show how conditional partial order
graphs, a compact representation of families of partial or-
ders, can be used for addressing the problem of compact
and easy-to-comprehend representation of event logs with
data. They present algorithms for extracting both the control
flow as well as relevant data parameters from a given event
log and show how conditional partial order graphs can be
used to visualize the obtained results. The method has been
implemented as a Workcraft plug-in and as a standalone
application called PGminer.

The Hybrid Miner, presented in [62], puts forward the
idea of discovering a hybrid model from an event log
based on the semantics defined in [106]. According to such
semantics, a hybrid process model is a hierarchical model,
where each node represents a sub-process, which may be
specified in a declarative or procedural way. Petri nets are
used for representing procedural sub-processes and Declare
for representing declarative sub-processes.

[68] proposes an approach for the discovery of hybrid
models based on the semantics proposed in [107]. Differently
from the semantics introduced in [106], where a
hybrid process model is hierarchical, the semantics defined
in [107] is devoted to obtain a fully mixed language, where
procedural and declarative constructs can be connected with
each other.

3.2 Procedural language constructs (RQ3)
All the 26 methods that discover a procedural model can
detect the basic control-flow structure of sequence. Out of
these methods, only four can also discover inclusive choices,
but none in the context of non-block-structured models. In
fact, [16], [38] are able to directly identify block-structured
inclusive choices (using process trees), while [79], [96] can
detect this construct only when used on top of the methods
in [16] or [38] (i.e., indirectly).

The remaining 22 methods can discover constructs for
parallelism, exclusive choice and loops, with the exception
of [47], which can detect exclusive choice and loops but not
parallelism, [34], which can detect parallelism and exclusive
choice but not loops, and [67], which can discover exclusive
choices only.

3.3 Implementation (RQ4)
Over 50% of the methods (19 out of 35) provide an imple-
mentation as a plug-in for the ProM platform. The reason
behind the popularity of ProM can be explained by its open-
source and portable framework, which allows researchers
to easily develop and test new discovery algorithms. Also,
ProM is the first software tool for process mining. One of
the methods which has a ProM implementation [33] is also
available as standalone tool. The works [19], [29], [52], [95]
provide both a standalone implementation and a further
implementation as a plug-in for Apromore. Apromore is
an online process analytics platform, also open source, and
has a growing consensus among academics as a process
mining tool oriented towards end users. Finally, one method
[84] has been implemented as a plug-in for Workcraft, a
platform for designing concurrent systems.

1. http://promtools.org
2. http://apromore.org
3. http://workcraft.org
Notice that 22 tools out of 35 are made publicly available to the community. These exclude 4 ProM plug-ins and 9 standalone tools.

### 3.4 Evaluation data and domains (RQ5)

The surveyed methods have been evaluated using three types of event logs: (i) real-life logs, i.e., logs of real-life process execution data; (ii) synthetic logs, generated by replaying real-life process models; and (iii) artificial logs, generated by replaying artificial models.

We found that the majority of methods (31 out of 35) were tested using real-life logs. Among them, 11 approaches (cf. [19], [25], [26], [33], [62], [68], [69], [75], [77], [82], [93]) were further tested against synthetic logs, while 13 approaches (cf. [12], [16], [19], [55], [56], [63], [68], [71], [72], [79], [83], [84], [94]) against artificial logs. Finally, one method was tested both on synthetic and artificial logs only (cf. [73]), while [24], [88] were tested on artificial logs and [49] on synthetic logs only. Among the methods that employ real-life logs, we observed a growing trend in employing publicly-available logs, as opposed to private logs which hamper the replicability of the results due to not being accessible.

Concerning the application domains of the real-life logs, we noticed that several methods used a selection of the logs made available by the Business Process Intelligence Challenge (BPIC), which is held annually as part of the BPM Conference series. These logs are publicly available at the 4TU Centre for Research Data, and cover domains such as healthcare (used by [19], [33], [38], [46], [71], [85], [92]), banking (used by [19], [33], [88], [62], [67], [72], [77], [81], [85], [92], [96]), IT support management in automotive (cf. [16], [26], [38], [55], [96]), and public administration (cf. [16], [26], [38], [55], [96]). A public log pertaining to a process for managing road traffic fines (also available at the 4TU Centre for Research Data) was used in [19], [92]. In [84], the authors use logs from various domains available at http://www.processmining.be/actitrac/

Besides these publicly-available logs, a range of private logs were also used, originating from different domains such as logistics (cf. [87], [89]), traffic congestion dynamics (69), employers habits (cf. [55], [81]), automotive [12], healthcare [19], [72], [92], and project management and insurance (cf. [47], [79]).

### 4 Benchmark

Using a selection of the methods surveyed in this paper, we conducted an extensive benchmark to identify relative advantages and tradeoffs. In this section, we justify the criteria of the methods selection, describe the datasets, the evaluation setup and metrics, and present the results of the benchmark. These results, consolidated with the findings from the systematic literature review, are then discussed in Section 5.

#### 4.1 Methods selection

Assessing all the methods that resulted from the search would not be possible due to the heterogeneous nature of the inputs required and the outputs produced. Hence, we decided to focus on the largest subset of comparable methods. The methods considered were the ones satisfying the following criteria:

- an implementation of the method is publicly accessible;
- the output of the method is a Petri net or a model seamlessly convertible into a Petri net (i.e., process trees and BPMN models).

The second criterion is a requirement dictated by the metrics used to evaluate the accuracy of a discovered model (illustrated later in this section) that can only be computed on top of Petri nets.

The application of these criteria resulted in an initial selection of the following methods (corresponding to one third of the total studies): αS [12], Inductive Miner [39], Evolutionary Tree Miner [16], Fodina [93], Structured Heuristic Miner 6.0 [19], Split Miner [92], Hybrid ILP Miner [103], RegPFA [77], Stage Miner [95], BPMN Miner [79], Decomposed Process Mining [109].

A posteriori, we excluded the latter four due to the following reasons: Decomposed Process Mining, BPMN Miner, and Stage Miner were excluded as such approaches follow a divide-and-conquer approach which could be applied on top of any discovery method to improve its results; on the other hand, we excluded RegPFA because its output is a graphical representation of a Petri net (DOT), which could not be seamlessly serialized into the standard Petri net format.

We also considered including commercial process mining tools in the benchmark. Specifically, we investigated Disco, Celonis, Mini, and myInvenio. Disco and Minit are not able to produce business process models from event logs. Instead, they can only produce directly-follows graphs, which do not have an execution semantics. Indeed, when a given node (task) has several incoming arcs, a directly-follows graph does not tell us whether or not the task in question should wait for all its incoming tasks to complete, or just for one of them, or a subset of them. A similar remark applies to split points in the directly-follows graph. Given their lack of execution semantics, it is not possible to directly translate a directly-follows graph into a BPMN models or a Petri net. Instead, one has to determine what is the intended behavior at each split and join point, which is precisely what several of the automated process discovery techniques based on directly-follows graph do (e.g., the Inductive Miner or Split Miner).

Celonis and myInvenio can produce BPMN process models but all they do is to insert OR (inclusive) gateways at the split and join points of the process map. The resulting BPMN process models cannot be translated to Petri nets using existing mappings from BPMN to Petri nets [108]. The
Split Miner adopts a similar approach (it initially inserts OR-join gateways at join points in the directly-follows graph), but it then applies an algorithm to turn these OR-join gateways into combinations of XOR and AND gateways – at the expense of potentially making the process model unsafe (but deadlock-free). It would have been possible to take the output of Celonis and myInvenio and apply an approach similar to the Split Miner to remove the OR-join gateways, but the resulting technique would be in essence the Split Miner itself, and the latter is already included in the benchmark.

In conclusion, the final selection of methods for the benchmark contained: α§, Inductive Miner (IM), Evolutionary Tree Miner (ETM), Fodina (FO), Structured Heuristic Miner 6.0 (S-HM6), Split Miner (SM), and Hybrid ILP Miner (HILP).

4.2 Setup and datasets

To guarantee the reproducibility of our benchmark and to provide the community with a tool for comparing new methods with the ones evaluated in this paper, we developed a command-line Java application that performs measurements of accuracy and complexity metrics on the seven discovery methods selected above. The tool can be easily extended to incorporate new discovery methods and metrics.

For our evaluation, we used two datasets. The first is the collection of real-life event logs publicly available at the 4TU Centre for Research Data as of March 2017. Out of this collection, we considered the BPI Challenge (BPIC) logs, the Road Traffic Fines Management Process (RTFMP) log, and the SEPSIS Cases log. These logs record executions of business processes from a variety of domains, e.g., healthcare, finance, government and IT service management. For our evaluation we held out those logs that do not explicitly capture business processes (i.e., the BPIC 2011 and 2016 logs), and those contained in other logs (e.g., the Environmental permit application process log). Finally, in three logs (i.e., the BPIC14, BPIC15, and BPIC17 logs), we applied the filtering technique proposed in [109] to remove infrequent behavior. This was necessary since all the models discovered by the considered methods exhibited very poor accuracy (F-score close to 0 or not computable) on these logs, making the comparison useless.

Table 3 reports the characteristics of the twelve logs used. These logs are widely heterogeneous ranging from simple to very complex, with a log size ranging from 681 traces (for the BPIC152 log) to 150,370 traces (for the RTFMP log). A similar variety can be observed in the percentage of distinct traces, ranging from 0.2% to 80.6%, and the number of event classes (i.e., activities executed within the process), ranging from 7 to 82. Finally, the length of a trace also varies from very short, with traces containing only one event, to very long with traces containing 185 events.

The second dataset is composed of twelve proprietary logs sourced from several companies around the world.

Table 4 reports the characteristics of these logs. Also in this case, the logs are quite heterogeneous, with the number of traces (and the percentage of distinct traces) ranging from 225 (of which 99.9% distinct) to 787.657 (of which 0.01% distinct). The number of recorded events varies between 4,434 and 2,099,835, whilst the number of event classes ranges from 8 to 310.

We performed our benchmark on a 6-core Intel Xeon CPU E5-1650 v3 @ 3.50GHz with 128GB RAM running Java 8. We allocated a total of 16GB to the heap space, and we enforced a timeout of four hours for the discovery phase and one hour for measuring each of the quality metrics.

4.3 Evaluation metrics

For all selected discovery metrics we measured the following accuracy and complexity metrics: recall (a.k.a. fitness), precision, generalization, complexity, and soundness.

Fitness measures the ability of a model to reproduce the behavior contained in a log. Under trace semantics, a fitness of 1 means that the model can reproduce every trace in the log. In this paper, we use the fitness measure proposed in [110], which measures the degree to which every trace in the log can be aligned (with a small number of errors) with a trace produced by the model. In other words, this measures tells us how close on average a given trace in the log can be aligned with a trace that can be generated by the model.

Precision measures the ability of a model to generate only the behavior found in the log. A score of 1 indicates that any trace produced by the model is contained in the log. In this paper, we use the precision measure defined in [111], which is based on similar principles as the above fitness measure. Recall and precision can be combined into a single measure known as F-score, which is the harmonic mean of the two measurements $\left(2 \cdot \frac{\text{Precision} \cdot \text{Fitness}}{\text{Precision} + \text{Fitness}}\right)$.

Generalization refers to the ability of an automated discovery algorithm to discover process models that generate traces that are not present in the log but that can be produced by the business process under observation. In other words, an automated process discovery algorithm has a high generalization on a given event log if it is able to discover a process model from the event log, which generates traces that: (i) are not in the event log, but (ii) can be produced by the business process that produced the event log. Note that unlike fitness and precision, generalization is a property of an algorithm on an event log, and not a property of the model produced by an algorithm when applied to a given event log.

In line with the above definition, we use k-fold cross-validation [112] to measure event logs. This k-fold cross-validation approach to measure generalization has been advocated in several studies in the field of automated process discovery [2, 113, 114, 115]. Concretely, we divide the log into $k$ parts, we discover a model from $k - 1$ parts (i.e., we hold-out one part), and measure the fitness of the discovered model against the part held out. This is repeated for every possible part held out. Generalization is the mean of the fitness values obtained for each part held out. A generalization of one means that the discovered model produces traces in the observed process, even if those traces are not in the log from which the model was discovered, and
that the discovered model is accurate and does not introduce extra behavior (i.e., does not over-generalize the behavior recorded in the log).

In the results reported below, we use a value of $k = 3$ for performance reasons (as opposed to the classical value of $k = 10$). The fitness calculation for most of the algorithm-log pairs is slow, and repeating it 10 times, for every algorithm-log combination is costly. To test if the results could be affected by this choice of $k$, we used $k = 10$ for SM and IM on the BPIC12 log, and found that the value of the 10-fold generalization measure was within one percentage point of that of the 3-fold generalization measure.

**Complexity** quantifies how difficult it is to understand a model. Several complexity metrics have been shown to be (inversely) related to understandability [116], including Size (number of nodes); Control-Flow Complexity (CFC) (the amount of branching caused by gateways in the model) and Structuredness (the percentage of nodes located directly inside a block-structured single-entry single-exit fragment).

Lastly, **soundness** assesses the behavioral quality of a process model by reporting whether the model violates one of the three soundness criteria [117]: i) option to complete, ii) proper completion, and iii) no dead transitions.

### 4.4 Benchmark results

We performed two types of evaluations. The first evaluation was meant to compare all the process discovery methods using their default parameters. In the second evaluation, we wanted to perform a similar comparison using hyperparameter optimization. Due to their extremely long execution times, for this second evaluation we held out $\alpha^8$ and ETM which would have been prohibitive for a hyperparameter optimization exercise. Additionally, we excluded HILP since we did not find any input parameters which could be used to optimize the f-score of the models produced. For the remaining four methods, we evaluated the following input parameters: the two filtering thresholds required in input by SM and S-HM, the single threshold required in input by IM, and the threshold and the boolean flag required in input by FO. Since all the thresholds range from 0.0 to 1.0, we used steps of 0.05 for IM, and steps of 0.10 for the thresholds of SM, S-HM, and FO. For FO, we considered all the possible combinations of the filtering threshold and the boolean flag.

The results of the first evaluation are shown in the tables [5-8] and [8] where the best score for each measure on each log is highlighted in bold. In the tables, we used “-” to report that a given accuracy or complexity measurement could not be reliably obtained due to syntactical or behavioral issues in the discovered model (i.e., a disconnected model or an unsound model). Additionally, to report the occurrence of a timeout or an exception during the execution of a discovery method we used “t/o” and “ex”, respectively.

The first evaluation shows the absence of a clear winner among the discovery methods tested, although almost each of them clearly showed specific benefits and drawbacks.

HILP experienced severe difficulties in producing useful outputs. The method often produced disconnected models or models containing multiple end places without providing information about the final marking (a well defined final marking is required in order to measure fitness and precision). Due to these difficulties, we could only assess model complexity for HILP, except for the simplest event

---

11. This to have a similar number of data points across all methods.
| Log      | Discovery Method | Accuracy | Gen. Complexity | Sound | Exec. Time(s) |
|----------|-----------------|----------|----------------|-------|---------------|
|          | Method | Fitness | Precision | F-score | Size | Class | Struct. |          |          |               |
|          |         |         |           |         |      |       |         |          |          |               |
| BPIC12   | αβ     | 1/0     | 1/0       | 1/0     | 1/0  | 1/0   | 1/0     | t/o      | 10/12    | t/o          |
|          | IM     | 0.98    | 0.50      | 0.66    | 0.98 | 59    | 37      | 1.00     | yes      | 6.6          |
|          | ETM    | 0.44    | 0.82      | 0.57    | t/o  | 67    | 16      | 1.00     | yes      | 14.400       |
|          | FO     | -       | -         | -       | 102  | 117   | 0.13    | no       | 9.66     |              |
|          | S-HMα  | -       | -         | -       | 88   | 46    | 0.40    | no       | 227.8    |              |
|          | HILP   | -       | -         | -       | 300  | 460   | -       | no       | 772.2    |              |
|          | SM     | 0.75    | 0.76      | 0.76    | 0.75 | 53    | 32      | 0.72     | yes      | 0.58         |
|          |         |         |           |         |      |       |         |          |          |               |
|          | αβ     | -       | -         | -       | 18   | 3     | -       | no       | 10/12    | t/o          |
|          | IM     | 0.82    | 1.00      | 0.90    | 0.82 | 9     | 4       | 1.00     | yes      | 0.1          |
|          | ETM    | 1.00    | 0.70      | 0.82    | t/o  | 38    | 38      | 1.00     | yes      | 14.400       |
|          | FO     | -       | -         | -       | 25   | 23    | 0.60    | no       | 0.06     |              |
|          | S-HMα  | 0.94    | 0.99      | 0.97    | 0.94 | 15    | 6       | 1.00     | yes      | 130.0        |
|          | HILP   | -       | -         | -       | 10   | 3     | -       | yes      | 0.1      |              |
|          | SM     | 0.94    | 0.97      | 0.96    | 0.94 | 12    | 7       | 1.00     | yes      | 0.03         |

**TABLE 5**: Default parameters evaluation results for the BPIC logs.
log (the PRT5), where HILP had performance comparable to the other methods.

αS showed scalability issues, timing out in eight event logs (33% of the times). Although none of the discovered models stood out in accuracy or in complexity, αS in general produced models striking a good balance between fitness and precision (except for the BPIC13 inc log).

FO struggled in delivering sound models, discovering only eight sound models. Nevertheless, its outputs were usually highly fitting, scoring five times the best fitness.

S-HM6 performed better than FO, although it also ended up producing unsound models. Of the 16 sound models discovered, nine scored the best fitness. However, precision varied according to the input event log, demonstrating that the performance of this method is bound to the type of log provided in input.

The remaining three methods, i.e., IM, ETM, and SM, consistently performed very well across the whole evaluation, excelling either in fitness, precision or f-score. IM scored 20 times a fitness greater than 0.90 (of which 8 times the highest), though, IM did not stand out for its precision. ETM and SM achieved 19 times a precision greater than 0.80, and ETM precision was the best 10 times. However, ETM scored high precision at the cost of lower fitness. Lastly, SM stood out for its F-score (i.e., high and balanced fitness and precision), achieving an F-score above 0.80 and

---

**TABLE 6: Default parameters evaluation results for the public logs.**

| Log   | Discovery Method | Accuracy | Gen. (3-Fold) | Complexity | Sound? | Exec. Time (sec) |
|-------|------------------|----------|---------------|------------|--------|------------------|
|       | Fitness | Precision | F-score | Size | CFC | Struct. |         |          |
| RTFMP |       |           |        |      |     |        |         |          |
| IM    | 0.90   | 0.67      | 0.77   | 0.90 | 20  | 1.00   | no      | 1,068.54 |
| ETM   | 0.99   | 0.81      | 0.89   | t/o  | 23  | 1.00   | yes     | 14,400   |
| FO    | -      | -         | -      | t/o  | 30  | 0.53   | no      | 0.95     |
| S-HM6 | 0.88   | 0.77      | 0.82   | 0.88 | 59  | 1.00   | yes     | 122.16   |
| HILP  | -      | -         | -      | 195 | 271 | 1.00   | no      | 2.59     |
| SM    | 0.98   | 0.99      | 0.98   | 29  | 18  | 0.79   | yes     | 0.47     |
| SEPSIS|       |           |        |      |     |        |         |          |
| IM    | 0.99   | 0.45      | 0.62   | 0.96 | 50  | 1.00   | yes     | 0.4      |
| ETM   | 0.83   | 0.66      | 0.74   | t/o  | 108 | 1.00   | yes     | 14,400   |
| FO    | -      | -         | -      | 60  | 63  | 0.28   | no      | 0.17     |
| S-HM6 | 0.92   | 0.42      | 0.58   | 0.92 | 279 | 198   | 1.00    | yes      | 242.7    |
| HILP  | -      | -         | -      | 87  | 129 | no     | no      | 1.6      |
| SM    | 0.73   | 0.86      | 0.79   | 0.73 | 31  | 0.97   | yes     | 0.05     |

**TABLE 7: Default parameters evaluation results for the proprietary logs - 1.**
outperforming the other methods for 18 times. Despite such good results, SM was not able to consistently produce sound models, as showed in the case of the PRT11 log.

In terms of complexity, IM, ETM, and SM delivered good results. IM and ETM always discovered sound and fully block-structured models (struct. of 1.00). ETM and SM discovered models that were within the two smallest models for more than the 80% of the inputs, and that had low CFC, being the ones with the lowest CFC respectively on 14 and 6 logs. On the execution time, SM was the clear winner. It systematically outperformed all the other methods, regardless the input. It was the fastest discovery method 23 times out of 24, discovering a model in less than a second over 19 logs. On the other hand, ETM was the slowest discovery method, reaching the timeout of four hours over 22 event logs.

Finally, Table 8 displays the results of the hyper-parameter evaluation. The purpose of this second evaluation was to explore the solutions’ space of each discovery method, to understand if they can achieve higher F-score when optimally tuned. According to this aim, the results of the hyper-parameter evaluation are extremely positive. All the discovery methods were able to improve their F-scores for almost all the inputs (values highlighted in italic in Table 8). Precisely, IM, FO, S-HM₆, and SM achieved better F-scores in 19, 14, 15, and 11 event logs, respectively. This denotes that IM default parameters are not optimal when focusing on the discovery of models with high F-score. FO and S-HM₆ default parameters are more reliable. Whilst, SM showed to achieve the best with the default parameters more than the 50% of the times. Lastly, SM again outperformed all the other methods, scoring the highest hyper-parameter optimized F-score over 20 event logs.

In conclusion, a method outperforming all other across all metrics could not be identified. All that aside, IM, ETM and SM showed to be the most effective methods when the focus is either on fitness, precision, or F-score, respectively. Nevertheless, all these three methods suffer from one common weakness, which is the inability to handle large-scale real-life logs, as reported for the PRT11 log in our evaluation.

5 Discussion

Our review highlights a growing interest in the field of automated process discovery, and confirms the existence of a wide and heterogeneous number of proposals. Despite such a variety, we can clearly identify two main streams: methods that output procedural process models, and methods that output declarative process models. Further, while the latter ones only rely on declarative statements to represent a process, the former provide various language alternatives, though, most of these methods output Petri nets.

The predominance of Petri nets is driven by the expressive power of this language, and by the requirements of

Table 8: Default parameters evaluation results for the proprietary logs - 2.
| Log | Metrics | IM | FO | S-HM* | SM |
|-----|---------|----|----|-------|----|
| BPIC12 | Fitness | 0.91 | 0.82 | 0.96 | 0.91 |
|       | Precision | 0.64 | 0.41 | 0.66 | 0.83 |
|       | F-score | 0.71 | 0.54 | 0.78 | 0.87 |
| BPIC17 | Fitness | 0.99 | - | 0.96 | 0.94 |
|       | Precision | 0.98 | - | 1.00 | 0.97 |
|       | F-score | 0.98 | - | 0.98 | 0.96 |
| BPIC17 | Precision | 1.00 | - | 0.93 | 0.91 |
|       | F-score | 0.71 | - | 0.98 | 0.98 |
|       | S-HM* | 0.83 | - | 0.96 | 0.94 |
| BPIC14t | Fitness | 0.75 | 0.94 | 0.91 | 0.80 |
|       | Precision | 0.97 | 0.85 | 0.84 | 0.99 |
|       | F-score | 0.85 | 0.89 | 0.88 | 0.89 |
| BPIC15tt | Fitness | 1.00 | 0.90 | 0.88 | 0.95 |
|       | Precision | 0.57 | 0.88 | 0.89 | 0.86 |
|       | F-score | 0.72 | 0.89 | 0.89 | 0.90 |
| BPIC15tt | Precision | 0.69 | 0.99 | 0.99 | 0.81 |
|       | F-score | 0.79 | 0.63 | 0.62 | 0.86 |
|       | S-HM* | 0.74 | 0.77 | 0.76 | 0.83 |
| BPIC15tt | Fitness | 0.77 | 0.80 | 0.81 | 0.78 |
|       | Precision | 0.80 | 0.86 | 0.77 | 0.94 |
|       | F-score | 0.79 | 0.83 | 0.79 | 0.85 |
| BPIC15tt | Precision | 0.73 | 0.76 | 0.99 | 0.77 |
|       | F-score | 0.87 | 0.87 | 0.66 | 0.90 |
|       | S-HM* | 0.80 | 0.81 | 0.79 | 0.83 |
| BPIC15tt | Fitness | 0.65 | 0.81 | 0.82 | 0.86 |
|       | Precision | 0.87 | 0.91 | 0.94 | 0.90 |
|       | F-score | 0.75 | 0.86 | 0.87 | 0.88 |
| BPIC15tt | Precision | 1.00 | 1.00 | 0.99 | 0.95 |
|       | F-score | 0.70 | 0.10 | 0.70 | 0.85 |
|       | S-HM* | 0.82 | 0.12 | 0.81 | 0.90 |
| RTFMP | Fitness | 0.96 | 1.00 | 0.98 | 0.99 |
|       | Precision | 0.72 | 0.94 | 0.96 | 1.00 |
|       | F-score | 0.82 | 0.97 | 0.97 | 1.00 |
| SEPSIS | Fitness | 0.66 | 0.74 | 0.92 | 0.85 |
|       | Precision | 0.91 | 0.67 | 0.42 | 0.73 |
|       | F-score | 0.76 | 0.70 | 0.58 | 0.79 |
| PRT1 | Fitness | 1.00 | 0.98 | 0.96 | 0.98 |
|       | Precision | 0.82 | 0.92 | 0.98 | 0.99 |
|       | F-score | 0.90 | 0.95 | 0.97 | 0.98 |
| PRT2 | Fitness | 1.00 | - | 1.00 | 0.81 |
|       | Precision | - | 0.17 | - | 0.74 |
|       | F-score | - | 0.30 | - | 0.77 |
| PRT3 | Fitness | 0.87 | 1.00 | 0.99 | 1.00 |
|       | Precision | 0.93 | 0.86 | 0.85 | 0.92 |
|       | F-score | 0.90 | 0.92 | 0.91 | 0.96 |
| PRT4 | Fitness | 0.86 | 1.00 | 0.93 | 0.99 |
|       | Precision | 1.00 | 0.87 | 0.96 | 1.00 |
|       | F-score | 0.92 | 0.93 | 0.95 | 0.99 |
| PRT5 | Fitness | 1.00 | 1.00 | 1.00 | 1.00 |
|       | Precision | 1.00 | 1.00 | 1.00 | 1.00 |
|       | F-score | 1.00 | 1.00 | 1.00 | 1.00 |
| PRT6 | Fitness | 0.92 | 1.00 | 0.98 | 0.94 |
|       | Precision | 1.00 | 0.91 | 0.96 | 1.00 |
|       | F-score | 0.96 | 0.95 | 0.97 | 0.97 |
| PRT7 | Fitness | 0.88 | 0.99 | 1.00 | 0.93 |
|       | Precision | 1.00 | 1.00 | 1.00 | 1.00 |
|       | F-score | 0.93 | 0.99 | 1.00 | 0.96 |
| PRT8 | Fitness | 0.79 | 1.00 | 0.93 | 0.99 |
|       | Precision | 0.37 | 0.14 | 0.42 | 0.66 |
|       | F-score | 0.51 | 0.25 | 0.58 | 0.79 |
| PRT9 | Fitness | 0.93 | - | 0.96 | 0.99 |
|       | Precision | 0.68 | - | 0.98 | 1.00 |
|       | F-score | 0.78 | - | 0.97 | 0.99 |
| PRT10 | Fitness | 0.99 | 0.99 | 0.98 | 0.97 |
|       | Precision | 0.79 | 0.93 | 0.83 | 0.97 |
|       | F-score | 0.88 | 0.96 | 0.90 | 0.98 |
| PRT11 | Fitness | ex | - | - | - |
|       | Precision | ex | - | - | - |
|       | F-score | ex | - | - | - |
| PRT12 | Fitness | 0.97 | 1.00 | - | 0.98 |
|       | Precision | 0.85 | 0.80 | - | 0.97 |
|       | F-score | 0.91 | 0.89 | - | 0.98 |

### TABLE 9: Hyperparameters-optimization evaluation results.

The methods used to assess the quality of the discovered process models (chiefly, fitness, precision and F-score). Despite some modeling languages have a straightforward conversion to Petri nets, the strict requirements of these quality assessment tools represent a limitation for the proposals in this research field. For the same reason, it was not possible to compare the two main streams, so we decided to focus on the evaluation and comparison on the procedural methods, which in any case, have a higher practical relevance than their declarative counterparts, given that declarative process models are hardly used in practice.

Our benchmark shows benefits and drawbacks of the procedural automated process discovery methods, as well their limitations. These latter include lack of scalability for large and complex logs, and strong differences in the output models, across the various quality metrics. Regarding this aspect, the majority of methods were not able to excel in accuracy or complexity, except for IM, ETM and SM. Indeed, these three ones were the only ones to consistently perform very well in fitness (IM), precision (ETM, SM), F-score (SM), complexity (IM, ETM, SM) and execution time (SM). Nevertheless, our evaluation shows that even IM, ETM and SM can fail when challenged with large-scale unfiltered real-life events logs, as shown in the case of the event log PRT11.

To conclude, even if many proposals are available in this research area, and some of them are able to systematically deliver good to optimal results, there is still space for research and improvements. Furthermore, it is important to highlight that the great majority of the methods do not have a working or available implementation. This hampers their systematic evaluation, so one can only rely on the results reported in the respective papers. Finally, for those methods we assessed, we were not able to identify a unique winner, since the best methods showed to either maximize fitness, precision or F-score. Despite these considerations, it can be noted that there has been significant progress in this field in the past five years. Indeed, IM, ETM and SM clearly outperformed the discovery methods developed in the previous decade and their extensions (i.e., Agnes and S-HM*).

### 6 Threats to Validity

The first threat to validity refers to the potential selection bias and inaccuracies in data extraction and analysis typical of literature reviews. In order to minimize such issues, our systematic literature review carefully adheres to the guidelines outlined in [4]. Concretely, we used well-known literature sources and libraries in information technology to extract relevant works on the topic of automated process discovery. Further, we performed a backward reference search to avoid the exclusion of potentially relevant papers. Finally, to avoid that our review was threatened by insufficient reliability, we ensured that the search process could be replicated by other researchers. However, the search may produce different results as the algorithm used by source libraries to rank results based on relevance may be updated (see, e.g., Google Scholar).

The experimental evaluation on the other hand is limited in scope to techniques that produce Petri nets (or models in languages such as BPMN or Process Trees, which can
be directly translated to Petri nets). Also, it only considers main studies identified in the SLR with an available implementation. In order to compensate for these shortcomings, we published the benchmarking toolset as open-source software in order to enable researchers both to reproduce the results herein reported and to run the same evaluation for other methods, or for alternative configurations of the evaluated methods.

Another limitation is the use of only 24 event logs, which to some extent limits the generalizability of the conclusions. However, the event logs included in the evaluation are all real-life logs of different sizes and features, including different application domains. To mitigate this limitation, we have structured the released benchmarking toolset in such a way that the benchmark can be seamlessly rerun with additional datasets.

### 7 Related Work

A previous survey and benchmark of automated process discovery methods has been reported by De Weerdt et al. [3]. This survey covered 27 approaches, and it assessed 7 of them. We used it as starting point for our study.

The benchmark reported by De Weerdt et al. [3] includes seven approaches, namely AGNESMiner, \( \alpha +, \alpha ++, \) Genetic Miner (and a variant thereof), Heuristics Miner and ILP Miner. In comparison, our benchmark includes \( \alpha \$ \) (which is an improved version of \( \alpha + \) and \( \alpha ++ \)), Structured Heuristics miner (which is an extension of Heuristics Miner), Hybrid ILP Miner (an improvement of ILP), Evolutionary Tree Miner (which is a genetic algorithm postdating the evaluation of De Weerdt et al. [3]). Notably, we did not include AGNESMiner due to the very long execution times (as suggested by the authors in a conversation over emails exchanged during this work).

Another difference with respect to the previous survey [3], is that in our paper we based our evaluation both on public and proprietary event logs, whilst the evaluation of De Weerdt et al. [3] is solely based on artificial event logs and closed datasets, due to the unavailability of public datasets at the time of that study.

In terms of results, De Weerdt et al. [3] found that Heuristics Miner achieved a better F-score than other approaches and generally produced simpler models, while ILP achieved the best fitness at the expense of low precision and high model complexity. Our results show that SM achieves even better F-score and lower model complexity than other techniques, followed by ETM and IM, which excelled for precision and fitness (respectively). Thus it appears that in this field, in the last years, progress has been pursued successfully.

Another previous survey in the field is outdated [118], and a more recent one is not intended to be comprehensive [119], but rather limits on plug-ins available in the ProM toolset. Another related effort is CoBeFra – a tool suite for measuring fitness, precision and model complexity of automatically discovered process models [120].

### 8 Conclusion

This article presented a Systematic Literature Review (SLR) of automated process discovery methods and a comparative evaluation of existing implementations of these methods using a benchmark covering twelve publicly-available real-life event logs, twelve proprietary real-life event logs, and nine quality metrics. The toolset used in this benchmark is available as open-source software and the 50% of the event logs are publicly available. The benchmarking toolset has been designed in a way that it can be seamlessly extended with additional methods, event logs, and evaluation metrics.

The SLR put into evidence a vast number of automated process discovery methods (344 relevant papers were analysed). Traditionally, many of these proposals produce Petri nets, but more recently, we observe an increasing number of methods that produce models in other languages, including BPMN and declarative constraints. We also observe a recent emphasis on methods that produce block-structured process models.

The results of the empirical evaluation show that methods that seek to produce block-structured process models (Inductive Miner and Evolutionary Tree Miner) achieve the best performance in terms of fitness or precision, and complexity. Whilst, methods that do not restrict the topology of the generated process models (Split Miner), produce process models of higher quality in terms of F-score, although these methods cannot guarantee soundness (though they can guarantee deadlock-freedom). We also observed that in the case of very complex event logs, it is necessary to use a filtering method prior to applying existing automated process discovery methods. Without this filtering, the precision of the resulting models was close to zero. A direction for future work is to develop automated process discovery techniques that incorporate adaptive filtering approaches so that they can auto-tune themselves to deal with very complex logs.

Another limitation observed while conducting the benchmark, was the lack of universal measures of fitness and precision, which would be applicable not only to Petri nets (or BPMN models that can be mapped to Petri nets), but equally well to declarative or data-driven process modeling notations. Developing more universal measures of fitness and precision is another possible target of future work.

### ACKNOWLEDGMENTS

This research is partly funded by the Australian Research Council (grant DP150103356) and the Estonian Research Council (grant IUT20-55). It is also partly supported by the H2020-RISE EU project FIRST (734599), the Sapienza grant DAKIP and the Italian projects Social Museum and Smart Tourism (CTN01_00034_23154), NEPTIS (PON03PE_00214_3), and RoMA - Resilence of Metropolitan Areas (SCN_00064).

### REFERENCES

[1] W. M. P. van der Aalst, T. Weijters, and L. Maruster, “Workflow mining: Discovering process models from event logs,” *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, 2004.

[2] W. van der Aalst, *Process Mining: Data Science in Action*. Springer, 2016.

[3] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, “A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs,” *Information Systems*, vol. 37, no. 7, pp. 654–676, 2012.

[4] B. Kitchenham, “Procedures for performing systematic reviews,” *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.
Adriano Augusto is a joint-PhD student at University of Tartu (Estonia) and Queensland University of Technology (Australia). He graduated in Computer Engineering at Polytechnic of Turin (Italy) in 2016, presenting a master thesis in the field of Process Mining.

Raffaele Conforti is a Post-Doctoral Research Fellow at the Queensland University of Technology, Australia. He conducts research on process mining and automation, with a focus on automated process discovery, quality improvement of process event logs and process-risk management.

Marlon Dumas is Professor of Software Engineering at University of Tartu, Estonia. Prior to this appointment he was faculty member at Queensland University of Technology and visiting researcher at SAP Research, Australia. His research interests span across the fields of software engineering, information systems and business process management. He is co-author of the textbook “Fundamentals of Business Process Management” (Springer, 2013).

Marcello La Rosa is Professor of Information Systems at the Queensland University of Technology, Australia. His research interests include process mining, consolidation and automation. He leads the Apromore Initiative (www.apromore.org), a strategic collaboration between various universities for the development of a process analytics platform, and co-authored the textbook “Fundamentals of Business Process Management” (Springer, 2013).

Fabrizio Maria Maggi is an Associate Professor at the University of Tartu, Estonia. He worked as Post-Doctoral Researcher at the Department of Mathematics and Computer Science, Eindhoven University of Technology. His research interest span business process management, data mining and service-oriented computing.

Andrea Marrella is Post-Doctoral Research Fellow at Sapienza Università di Roma. His research interests include human-computer interaction, user experience design, knowledge representation, reasoning about action, automated planning, business process management. He has published over 40 research papers and articles and 1 book chapter on the above topics.

Massimo Mecella is Associate Professor (with a Full Professor Habilitation) with Sapienza Università di Roma. His research focuses on service oriented computing, business process management, Cyber-Physical Systems and Internet-of-Things, advanced interfaces and human-computer interaction, with applications in fields such as digital government, smart spaces, healthcare, disaster/crisis response & management, cyber-security. He published more than 150 research papers and chaired different conferences in the above areas.

Allar Soo Allar Soo is student in the Masters of Software Engineering at University of Tartu. His Masters thesis is focused on automated process discovery and its use in practical settings.