International Journal of Cooperative Information Systems

QUALITY DIMENSIONS IN PROCESS DISCOVERY: THE IMPORTANCE OF FITNESS, PRECISION, GENERALIZATION AND SIMPLICITY

J.C.A.M. BUIJS and B.F. VAN DONGEN and W.M.P. VAN DER AALST

Faculty of Mathematics and Computer Science, Eindhoven University of Technology
Den Dolech 2, Eindhoven, the Netherlands

Received (Day Month Year)
Revised (Day Month Year)

Process discovery algorithms typically aim at discovering process models from event logs that best describe the recorded behavior. Often, the *quality of a process discovery algorithm* is measured by quantifying to what extent the resulting model can reproduce the behavior in the log, i.e. replay fitness. At the same time, there are other measures that compare a model with recorded behavior in terms of the precision of the model and the extent to which the model generalizes the behavior in the log. Furthermore, many measures exist to express the complexity of a model irrespective of the log.

In this paper, we first discuss several quality dimensions related to process discovery. We further show that existing process discovery algorithms typically consider at most two out of the four main quality dimensions: *replay fitness*, *precision*, *generalization* and *simplicity*. Moreover, existing approaches cannot steer the discovery process based on user-defined weights for the four quality dimensions.

This paper presents the *ETM algorithm* which allows the user to seamlessly steer the discovery process based on preferences with respect to the four quality dimensions. We show that all dimensions are important for process discovery. However, it only makes sense to consider precision, generalization and simplicity if the replay fitness is acceptable.

Keywords: process mining, process discovery, process model quality

1. Introduction

The goal of *process mining* is to automatically produce process models that accurately describe processes by considering only an organization's records of its operational processes. Such records are typically captured in the form of *event logs*, consisting of cases and events related to these cases. Using these event logs process models can be discovered.

Over the last decade, many such process discovery techniques have been developed, producing process models in various forms, such as Petri nets, BPMN-models, EPCs, YAWL-models etc. Furthermore, many authors have compared these techniques by focusing on the properties of the models produced, while at the same time the applicability of various techniques have been compared in case-studies.

In this paper, we first present a new high-level view on quality measures for process discovery. This is then related to the four commonly known quality dimensions for process discovery. We then also present the ETM algorithm, which is a genetic algorithm able to optimize the process discovery result towards any of these dimensions. By making use of so-called *process trees* [15] this algorithm ensures that the resulting model is a sound process model describing the observed log, while, at the same time, the model can be steered to emphasize different quality dimensions. Using the ETM algorithm, we can easily explore the effects of focusing on one dimension in isolation and on combinations of these dimensions.

Traditionally, when determining the quality of a mined process model, four quality dimensions are considered, e.g. simplicity, replay fitness, precision and generalization, where simplicity is a property of the model and the others relate the model to the event log. In this paper, we reconsider the notion of quality of a process mining result, by explicitly assuming the notion of a "system" outside the process model. In previous work the system was not explicitly considered while determining the quality of a process mining result. Nevertheless, the notion of the system is often implicitly assumed when discussing quality issues. In this work, a system can be a concrete information system implementation but can also refer to the context of the process, e.g. the organization, rules, economy, etc. This system may allow for people involved in the operational processes to deviate from the intended behavior of the information system, sometimes for good reasons. We show that explicitly considering the presence of such a "system" leads to new insights into the role of existing quality dimensions in process mining.

The remainder of this paper is structured as follows. In Section 2 we discuss related work in the area of process model quality assessment. Section 3 discusses the research approach and relates the various contributions of this paper. In Section 4 we discuss how the behavior of the system, event log and discovered process model are related and which measures are of importance during process discovery. In Section 5, we present our ETM algorithm. Furthermore, we present one measure for each of the four common quality dimensions and we present process trees as a convenient means of modeling sound process models. In Section 6, we then present a running example which we use throughout the remainder of the paper. Using this example, we show the quality of various existing process discovery algorithms in terms of the presented measures in Section 7. Section 8 then shows the results of focusing on a subset of the quality dimensions during process discovery. Here, we use our ETM algorithm to show that not considering all quality dimensions results in poor models. Subsection 8.4 shows the results when considering all dimensions and assigning different weights them. In Section 9 we apply existing techniques and our ETM algorithm on several real life event logs. Section 10 then discusses an extension of the approach that produces a collection of process models. Section 11 concludes the paper.

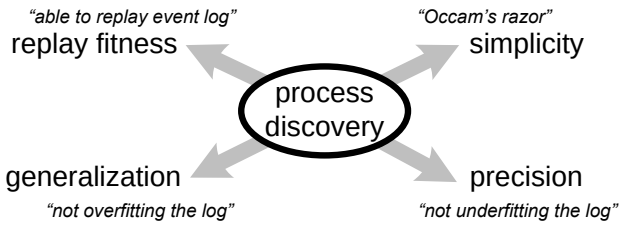


Fig. 1: Different quality dimensions for Process Model Discovery [2]

2. Related Work

As mentioned in the introduction, the goal of process mining is to automatically produce process models that accurately describe processes by considering only an organization's record of its operational processes [2]. Over the last decade many of such process discovery techniques have been developed [6, 8, 12, 20, 22, 28, 29, 34, 45, 46]. Each of these techniques uses a different approach to obtain a process model describing the observed behavior. To measure how well a process model describes the observed behavior, different quality dimensions are used, which are shown in Fig. 1.

The four different quality dimensions each cover a different aspect of the quality of a process model [2–4]:

Simplicity The simplicity dimension evaluates how simple the process model is to understand for a human. This dimension is therefore not directly related to the observed behavior but can consider the process model solitarily. Since there are different ways to describe the same behavior using different process models, choosing the simplest one is obviously best. This is also expressed by Occam's razor: "one should not increase, beyond what is necessary, the number of entities required to explain anything". However, sometimes a complex process model can only be simplified by changing the behavior, hence influencing the other quality dimensions. Several measures exist to measure how simple a process model is, for an overview we refer to [35]. However, research has also shown that size is the main complexity indicator [36].

Replay Fitness The quality dimension of replay fitness describes the fraction of the behavior in the event log that can be replayed by the process model. Several different measures exist for this quality dimension [2, 4, 7, 19, 21, 25–27, 34, 40, 44, 45]. Some measures [27, 45] consider traces of behavior as whole, checking if the whole trace can be replayed by the process model. Other measures consider the more detailed level of events within a trace and try to get a more fine-grained idea of where the deviations are. Another important difference between existing measures is that some enforce a process model to be in an accepted end state when the

whole trace is replayed. Others ignore this and allow the process model to remain in an active state when the trace ends. The most recent and robust technique [4, 7] uses a cost-based alignment between the traces in the event log and the most optimal execution of the process model. This allows for more flexibility and distinction between more and less important activities by changing the costs.

Precision The quality dimension of precision estimates the extent of the behavior allowed by the process model that is not observed in the event log. Because of the possibly unlimited behavior of the process model, in case of loops, only estimations of precision can be made. Several measures for precision have been suggested [27, 37, 40]. One of the most recent and robust techniques considers the partial state space constructed while replaying the event log on the process model [37]. The unused escaping edges from states that are visited are counted. The higher the fraction of unused edges the more additional behavior is allowed and the less precise the model is.

Generalization Replay fitness and precision only consider the relationship between the event log and the process model. However, the event log only contains a part of all the possible behavior that is allowed by the system. Generalization therefore should indicate if the process model is not "overfitting" to the behavior seen in the event log and describes the actual system. Another explanation for generalization is the likelihood that the process model is able to describe yet unseen behavior of the observed system [2].

To date only few measures for generalization exist. In [40] two measures for something related to generalization are proposed. However, these measures don't consider the event log and only measure the number of duplicate transitions in the process model. A measure that is more in line with the notion of generalization is presented in [4] and is based on the work in [13, 16, 17]. This measure uses how often certain states are visited and the number of different activities observed in each state. If a state is visited very often and only few activities are observed, it is unlikely that a new activity will be observed at the next visit. Vice versa, if each visit to a state is followed by a different activity, chances are very high that a next visit will be followed by a new activity.

Unlike existing approaches, we provide a process discovery technique that can seamlessly integrate the different quality dimensions (and possibly new criteria). Moreover, we provide new insights in the trade-off between these dimensions (e.g., by analyzing the Pareto front).

3. Research Approach

In this paper, we present a process mining algorithm that is flexible with respect to the quality dimensions it focusses on when discovering a process model and, using this algorithm, we show the relative importance of all four quality dimensions.

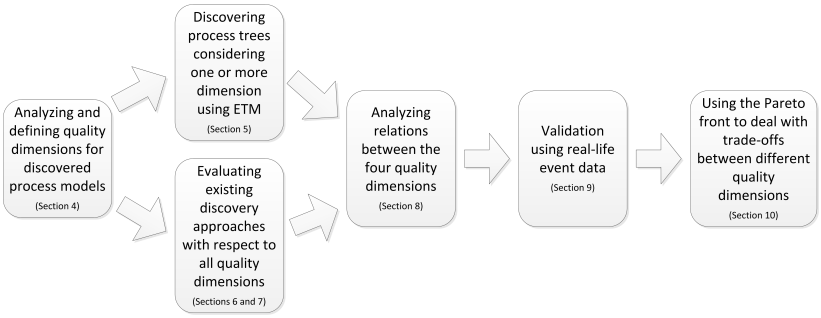


Fig. 2: Research approach and the role of the various sections in this paper

Figure 2 shows an overview of the approach followed in this paper. First, we carefully analyze the different quality dimensions starting from the viewpoint that we know the "real" process model. Our findings support the importance of the four classical quality dimensions described before (simplicity, replay fitness, precision, and generalization). Then, we present the ETM discovery algorithm. This discovery algorithm has two distinctive features compared to traditional approaches: (1) models are by definition free of deadlocks and other anomalies and (2) it is possible to seamlessly prioritize the different quality dimensions. Next to presenting the algorithm it is shown (using a running example) that traditional approaches indeed suffer from the problems the ETM discovery algorithm aims to avoid. Subsequently, the ETM algorithm is used to analyze the relationships between the different quality dimensions. This analysis shows that all quality dimensions are necessary, but that an acceptable replay fitness is a prerequisite for any form of evaluation. The findings are validated using real-life logs from the CoSeLoG project involving 10 Dutch municipalities. These experiments show that it is difficult to balance the different dimensions. Therefore, the ETM discovery algorithm does not provide one model but a collection of complementary models characterizing the Pareto front.

This paper follows the 'design science' research methodology [30]. Process discovery is clearly a "wicked problem" [39], i.e., it involves unstable requirements and constraints based on ill-defined environmental contexts, complex interactions among subcomponents of the problem, and a critical dependence upon human cognitive abilities to interpret results. This justifies the chosen research methodology. In Table 1 we reflect on the well-known seven design-science research guidelines [30].

4. Quality Dimensions for Discovered Process Models

Traditionally, next to simplicity three quality dimensions are considered when relating an event log to a model [2–4]. Replay fitness quantifies the fraction of the log supported by the model, precision quantifies the fraction of the model not observed in the log and generalization quantifies the likelihood that previously unseen behav-

Table 1: The design-science research guidelines by Hevner, March, Park and Ram [30] can be used to justify the chosen research approach

Guideline 1: Design as an Artifact	Design science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.	Clearly identifiable artifacts are the ETM algorithm and corresponding ProM implementation.
Guideline 2: Problem relevance	The objective of design science research is to develop technology-based solutions to important and relevant business problems.	Process mining techniques are increasingly being used in industry and there is a clear need for better discovery algorithms.
Guideline 3: Design evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.	Different quality dimensions described in literature are used and discussed. The corresponding metrics are used to empirically evaluate the performance of the ETM algorithm using both syntectic and real-life data.
Guideline 4: Research contributions	Effective design science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies	The paper demonstrates that the ETM algorithm overcomes limitations of existing approaches using logical argumentation and experiments.
Guideline 5: Research rigor	Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.	The approach defines clear and unambiguous criteria to evaluate discovered models.
Guideline 6: Design as a search process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.	Starting point is a detailed analysis of the weaknesses of existing process discovery approaches. Moreover, various data sets are used for evaluation.
Guideline 7: Communication of research	Design science research must be presented effectively to both technology-oriented and management-oriented audiences.	The paper aims to motivate the need for better discovery algorithms by showing many examples and discussing the desired outcome in a non-technical manner.

ior is supported by the model. In other words, the notion of generalization considers more than just the log and the model. Instead, it also uses an implicit notion of a "system" that is being observed. In this section, we make this notion more explicit and we consider the effect of doing so on the other quality dimensions.

4.1. *A Theoretical View*

In Fig. 3, we explicitly depict the behavior of a process model, the behavior observed in the event log and the behavior of an observed system. It is shown how the behavior included in these three entities can overlap, but also where they may be disjoint. In practice there are many forms in which behavior can be described, such as traces [3], behavioral profiles [44], α-relations [6], process algebra's [10], etc. However, for the

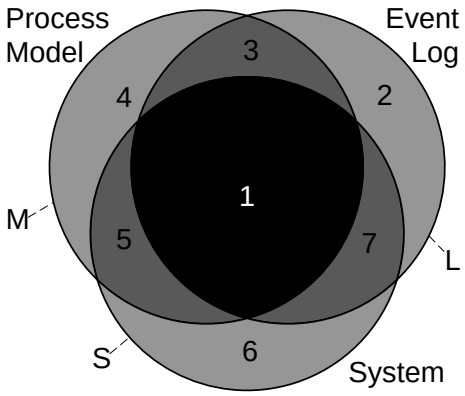


Fig. 3: Venn diagram showing how the behavior of the process model (M), event log (L) and system (S) can be disjoint or overlapping.

discussion in this paper, this is irrelevant.

The Venn diagram shown in Fig. 3 shows seven areas. We can intuitively describe the behavior contained in each area as follows:

(1) **Modeled and observed system behavior ($L \cap M \cap S$)**. The central black area in the Venn diagram contains all behavior that is the behavior of the system which is observed in the event log and possible in the process model.
(2) **Unmodeled exceptions ($(L \setminus M) \setminus S$)**. All the observed behavior that is actually non-system behavior is considered an exception. The exceptions that are not supported by the process model are contained in this area.
(3) **Modeled and observed exceptions ($(L \cap M) \setminus S$)**. Modeled and observed exceptions are those exceptions observed in the event log that are described by the process model.
(4) **Modeled but unobserved and non-system behavior ($(M \setminus S) \setminus L$)**. This contains all the behavior described by the process model which is non-system behavior and is also not found in the event log.
(5) **Modeled but unobserved system behavior ($(M \cap S) \setminus L$)**. The behavior described by the process model that is the system's behavior but is not seen in the event log.
(6) **Unmodeled and unobserved system behavior ($(S \setminus L) \setminus M$)**. All the system behavior that is neither observed in the event log nor modeled by the process model.
(7) **Unmodeled but observed system behavior ($(S \cap L) \setminus M$)**. The system behavior that is observed in the event log but not described by the process model.

It is important to realize that there is generally no way to explicitly describe

the behavior of the system, first of all since this behavior is typically infinite, but more so because there is always the possibility of unforeseen behavior in any real-world system that may even change over time. Nonetheless, the traditional goal of process mining is to *find a process model that describes this system as accurately as possible, using nothing more than the observed behavior in the log.* In the remainder of Section 4.1, we assume that the behavior of the system is known. In Section 4.2 we show how to deal with the fact that the behavior of the system is generally not known.

Relating the behavior allowed by the process model and that recorded in the event log, we can distinguish two measures commonly used in information retrieval. The *precision* between the process model and the event log expresses the amount of behavior not seen in the event log that can be produced by the process model. This can be expressed as:

$$\text{Model-Event Log Precision} = \frac{|L \cap M|}{|M|}$$

The *recall* between the model and the event log relates the behavior of the event log that can be produced by the process model versus all the observed behavior in the event log:

$$\text{Model-Event Log Recall} = \frac{|L \cap M|}{|L|}$$

Precision and recall between the process model and the event log follow the common notions of precision and recall in information retrieval. However, in process mining the problem setting is a bit different. In information retrieval instances should be classified correctly from a large set of instances. In process mining however we have observed instances in the event log that the process model should describe. This event log is created by, or obtained from, a system. So the behavior observed in the event log can also be related to the behavior of the system. This can be expressed by precision and recall between the event log and the system:

$$\text{Event Log-System Precision} = \frac{|L \cap S|}{|L|}$$

This expresses the fraction of the observed behavior in the event log that is included in the system.

The amount of overlap between the observed behavior recorded in the event log and the behavior of the system can be expressed as follows:

$$\text{Event Log-System Recall} = \frac{|L \cap S|}{|S|}$$

This expresses the fraction of the behavior of the system and seen in the event log, with respect to all the behavior of the system.

The behavior allowed by the process model can also be compared to the behavior of the system. Then again, precision can be calculated, but now for the process model w.r.t. the system. This is expressed as:

$$\text{Model-System Precision} = \frac{|S \cap M|}{|M|}$$

This fraction thus expresses the fraction of behavior that is allowed by the process model, but is not part of the behavior of the system.

Finally, recall between the model and the system expresses the fraction of the behavior expressed by the process model that is also the behavior of the system:

$$\text{Model-System Recall} = \frac{|S \cap M|}{|S|}$$

If all these six fractions are one then the three circles of the Venn diagram of Fig. 3 coincide and if all fractions are zero, then the three circles are disjoint.

In process mining, we start from a given event log L which comes from a given system S, i.e. L and S are constant. If we assume that S is known then we could simply use a genetic algorithm to discover a process model M which maximizes all of the fractions. However, the behavior of the system is unknown, but can, to some extent, be estimated from L.

4.2. *Dealing with an unknown system*

When considering the notion of a system, we rephrase the goal of process mining to: *discover a process model M from a given log L taken from an unknown, but constant system S, which maximizes all fractions listed in Section 4.1.*

The notion of Model-Event log precision directly relates to the existing quality dimension precision discussed in Section 2 and Model-Event log recall relates to replay fitness discussed there. Furthermore, if L and S are constant, then the Event log-System precision and recall are constant, i.e. these fractions become irrelevant for process discovery. In fact, many papers about process mining use the notion of noise to describe behavior observed in the log, but not part of the system (i.e. the noise level corresponds to 1−Event log-System precision). Furthermore, a certain level of completeness is often assumed which refers to the completeness of the log with respect to the system, i.e. Event log-System recall.

Things become more complex when we consider Model-System precision and Model-System recall under the assumption that the system is unknown. Basically, these metrics cannot be computed or estimated without further assumptions. Typically, process discovery algorithms use a hidden assumption that the process model they discovered from the event log does not include behavior outside of the system, i.e. they assume that $M \subseteq S$, hence model-system precision is one. This leaves the model-system recall to be estimated.

Finally, the model-system recall represents the fraction of the system which is covered by the model, i.e. if this recall is high then any behavior of the system can be explained with the model, regardless of whether this is observed or unobserved behavior. Under the assumption that $M \subseteq S$, this is what is traditionally referred to as generalization.

In Section 5 of this paper, we present the ETM algorithm, which is a process discovery algorithm that in many aspects makes the same assumptions about the relations between the log, the system and the model as other algorithms. The important difference is that the focus on the quality dimensions can be parameterized. However, before we introduce this algorithm, we present some requirements on how the various quality aspects should be measured.

4.3. *Measure Requirements*

Several measures can be created to indicate the quality of a process model in a certain dimension. However, there are some important requirements that each measure should follow.

Efficient Implementation Although in general the quality of a process model is only measured when it is required by the user, the implementation should still be efficient. For some measures the most easy way of calculation is also the most time consuming one. However, the user likely wants an almost instant answer. Furthermore, genetic algorithms, as the one we will present in Section 5, need to call each measure very often. Therefore, the performance of a genetic algorithm mainly depends on the time required by the measures to evaluate the process models [31, 32]. Approximation of the quality can greatly improve the performance of algorithms depending on the measurement.

Intuitive Results The main requirement for measures is that the results are intuitive. Process models that are extremely good or bad according to a certain measure should also be considered the best or worst according to the quality dimension the measure is related to. The same holds for a process model that according to the measure is slightly better or worse than another process model. Why it is better or worse, and by 'how much' should follow the philosophy of the quality dimension [11]. Probably even more important is that the user agrees with the result, since fitness evaluation is mainly an estimation of the user's preference [31].

Clear Specification The specification of the measure, e.g. the way it is calculated, should also be clear. If the specification cannot be understood, it cannot be verified why a certain process model has a certain value assigned to it. Furthermore, the measures should require as few parameters as possible since they make interpreting the results difficult if the parameter settings are unknown. Furthermore, in certain situations, parameters can change the results of a measure dramatically. This makes a measure unclear and less authoritative. The measure should be robust to different situations without requiring parameters.

Where possible, the measurement should be a proper metric [33], where triangle inequality is the most important property.

Orthogonal Different measures should be orthogonal to each other. If two measures both punish or reward certain aspects of a process model then that aspect is over-emphasized. This should be expressed by aiming at a good score for one measure that considers this aspect. If two measures overlap, the results become unclear and one of the measures is redundant. Furthermore, each measure should only cover a single quality dimension for the same reason.

An exception to this rule is incorporating additional specific user preferences. Those preferences are rarely independent of functional requirements on the process model. They should however be incorporated if the user desires to do so. However, the measurements for the recall and precision between the system, event log and process model should only be concerned with that aspect.

5. Process Trees and the ETM Algorithm

As stated in the introduction, we use the ETM algorithm to see the effects of (not) considering either of the four quality dimensions in process discovery. To this end, we first introduce process trees, which we use throughout the paper.

5.1. *Process Trees*

Traditional languages like BPMN, Petri nets, UML activity diagrams may be convenient ways of representing process models. However, only a small fraction of all possible models in these languages is *sound* [5], i.e. many models contain deadlocks, livelocks and other anomalies. Especially for the results presented in this paper, where the focus is on measuring the quality of the resulting models, it is essential that such unsound constructs are avoided. Therefore, we choose to use *process trees* to describe our models since all possible process trees represent sound process models.

Fig. 4 shows the possible operators of a process tree and their translation to a Petri net. A process tree contains operator nodes and leaf nodes. Operator nodes specify the relation between its children. Possible operators are sequence ($\rightarrow$), parallel execution ($\wedge$), exclusive choice ($\times$), non-exclusive choice ($\vee$) and loop execution ($\circlearrowleft$). The order of the children matters for the operators sequence and loop. The order of the children of a sequence operator specify the order in which they are executed (from left to right). For a loop, the left child is the 'do' part of the loop. After the execution of this 'do' part the right child, the 'redo' part, might be executed. After this execution the 'do' part is again enabled. The loop in Fig. 4 for instance is able to produce the traces $\langle A \rangle$, $\langle A, B, A \rangle$, $\langle A, B, A, B, A \rangle$ and so on.

Although also making use of a tree structure, a slightly different approach is taken by the Refined Process Structure Tree (RPST) [43]. The RPST approach provides "a modular technique of workflow graphs parsing to obtain fine-grained

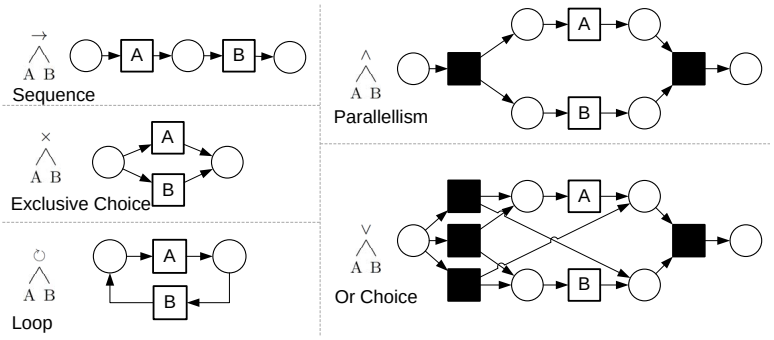


Fig. 4: Relation between process trees and block-structured Petri nets.

fragments with a single entry and single exit node" [43]. The content of these fragments are graphs themselves and are not necessarily block-structured nor sound. Each operator in a process tree however results in a block structured process part with a single entry and single exit node. Each block in a process tree can only contain a predefined control flow construct, which are shown in Fig. 4. Therefore, workflow graphs decomposed into an RPST can be more expressive than process trees but an RPST is not necessarily sound while a process tree always is.

5.2. *The Representational Bias of Process Trees*

As mentioned, the main benefit of working with process trees is the inherent soundness of block structured process models. This is an especially important property when working with genetic algorithms since they evaluate many process models. Because process trees are inherently sound, no additional effort has to be spend in detecting or fixing unsound process models. Besides soundness, there are other aspects that should be considered when determining the representation used by a process discovery algorithm. As discussed in [1] the internal representation chosen by a discovery algorithm can imply important limitations on the results it can produce. The main limitation that process trees have is the inability to express non-local dependencies without duplicating some of the activities involved. The genetic miner by Alves de Medeiros [34] for instance uses a causal matrix to represent petri nets. This allows them to produce process models that contain non-local dependencies. However they also often result in unsound process models, as is shown in Section 9. On the other hand, process models containing only local dependencies often provide a good compromise between expressive power and analyzability. Therefore, by using process trees as our internal representation we ensure soundness while at the same time maintaining enough expressive power. Furthermore, as we discuss in Section 10, our algorithm can easily be extended to return multiple process models. This allows us to return different process models that either ignore non-local dependencies or that explicitly capture these by duplicating certain activities.

5.3. *Translation of Process Trees to other Notations*

Process trees can be quite trivially translated to any of the well-known process modeling languages such as BPMN, EPC, Petri-net, etc. Because of its inherent block structure, the translation of process trees to other modeling languages results in clean process models with a clear structure. One of the most commonly used process modeling languages in industry is the BPMN notation [38]. All process trees that we will present as a result of our ETM algorithm will therefore also be shown in the BPMN process modeling notation. The translation is done in a similar way as the translation to Petri nets, as shown in Fig. 4. It should be noted however that process trees can easily be translated to any of the well-known process modeling languages.

5.4. *Quality of Process Trees*

To measure the quality of a process tree, we consider one measure for each of the four quality dimensions discussed in Section 4.2. We based these measures on existing work in each of the four areas and we adapted them for process trees [2, 4, 19, 35–37, 40].

Simplicity quantifies the complexity of the model. Simplicity is measured by comparing the size of the tree with the number of activities in the log. This is based on the observation that the size of a process model is the main factor for perceived complexity and introduction of errors in process models [36]. Furthermore, since we internally use binary trees, the number of leafs of the process tree corresponds to the number of operator nodes. Thus, if each activity is represented exactly once in the tree, that tree is considered to be as simple as possible. Therefore, simplicity is calculated as follows:

$$Q_s = 1 - \frac{\#\text{duplicate activities} + \#\text{missing activities}}{\#\text{nodes in process tree} + \#\text{event classes in event log}}$$

Duplication of activities is measured by counting the number of times the activity is repeated in the process model. An activity is missing from the process model if it is not included in the process model while it is present in the event log. These numbers are summed up and normalized by the total number of nodes in the process tree and event classes (or activities) in the event log.

This simplicity metric breaks the orthogonal requirement since leaving out, or duplicating activities in the process model, can be beneficial for the quality dimensions replay fitness and precision. However, this metric is mainly used to encode user preferences.

Replay fitness quantifies the extent to which the model can reproduce the traces recorded in the log. We use an alignment-based fitness computation defined in [4] to compute the fitness of a process tree. Basically, this technique aligns as many events as possible from the trace with activities in an execution of the model (this results in a so-called *alignment*). If necessary, events are skipped, or

activities are inserted without a corresponding event present in the log. Penalties are given for skipping and inserting activities. The final replay fitness score is calculated as follows:

$$Q_{rf} = 1 - \frac{\text{cost for aligning model and event log}}{\text{Minimal cost to align event log on model with no synchronous moves}}$$

where the denominator is the minimal costs when no match between event log and process model can take place (e.g. worst case scenario). This is used to normalize the replay fitness to a value between 0 and 1.

Calculating the alignments is a complex task and therefore takes relatively much time. This breaks with the first measurement requirement of efficient implementation. However, currently it is the most robust way of relating the process model with the event log. Moreover, the next two quality measures use the information provided by the alignments without requiring additional complex computations.

Precision indicates how much additional behavior the process model allows that is not seen in the event log. It therefore compares the state space of the tree execution while replaying the log. Our measure is inspired by [37] and counts so-called escaping edges, i.e. decisions that are possible in the model, but never made in the log. If there are no escaping edges, precision is considered to be perfect. We obtain the part of the state space used from information provided by the replay fitness, where we ignore events that are in the log, but do not correspond to an activity according to the alignment. In short, we calculate the precision as follows:

$$Q_p = 1 - \frac{\sum_{\text{visited markings}} \#\text{visits} * \frac{\#\text{outgoing edges} - \#\text{used edges}}{\#\text{outgoing edges}}}{\#\text{total marking visits over all markings}}$$

Generalization estimates how well the process model describes the (unknown) system, and not only the event log. If all parts of the process model are frequently used, the process model is likely to be generic. However, if some parts of the process model are rarely used, chances are high that the system actually allows for more behavior. Therefore we base the generalization measure on how often parts of the process model have been used while replaying the event log. For this we use the alignment provided by the replay fitness. If a node is visited more often then we are more certain that its behavior is (in)correct. If some parts of the tree are very infrequently visited, generalization is bad. Therefore, generalization is calculated as follows:

$$Q_g = 1 - \frac{\sum_{\text{nodes}} (\sqrt{\#\text{executions}})^{-1}}{\#\text{nodes in tree}}$$

The square root is taken from the number of executions because the effect of having 10 executions instead of 1 is considered a more significant improvement than from 10 to 100 executions. From each of these values the power of -1 is taken to normalize it to a value between 0 and 1. Then these values are summed

and divided by the total number of nodes in the tree to get the average for the whole tree.

Generalization is currently the quality dimension that is most difficult to express. However, results show that this measure for generalization follows the intuition for this quality dimension.

The four measures above are computed on a scale from 0 to 1, where 1 is optimal. Replay fitness, simplicity and precision can reach 1 as optimal value. Generalization however can only reach 1 in the limit i.e., the more frequent the nodes are visited, the closer the value gets to 1. The flexibility required to find a process model that optimizes a weighted sum over the four measures can efficiently be implemented using a genetic algorithm.

5.5. *The ETM Algorithm*

As discussed in Section 1 we propose the use of a genetic algorithm [14, 15] for the discovery of process models from event logs. Evolutionary algorithms have been applied to process mining discovery before in [34]. Our approach follows the same high-level steps as most evolutionary algorithms [24]. The overall approach is shown in Fig. 5. The main improvements with respect to [34] are the internal representation and the fitness calculations. By using a genetic algorithm for process discovery we gain flexibility: by changing the weights of different fitness factors we can guide the process discovery.

By using process trees as our internal representation we only consider sound process models. This drastically reduces the search space and therefore improves the performance of the genetic algorithm. Furthermore, we can apply standard tree change operations on the process trees to evolve them further. Finally, in our fitness calculation we consider *all four quality dimensions* for process models: replay fitness, precision, generalization and simplicity. The user can specify the relative importance of each dimension *beforehand*. The *ETM algorithm* (which stands for *Evolutionary Tree Miner*) will then favor those candidates that have the correct mix of the different quality dimensions.

In general, our genetic algorithm follows the process as shown in Fig. 5. The input of the algorithm is an event log describing observed behavior. In the initial

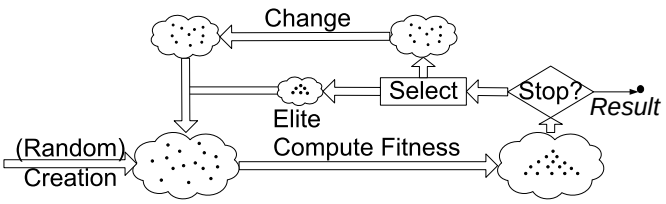


Fig. 5: The different phases of the genetic algorithm.

step a population of random process trees is generated where each activity occurs exactly once in each tree. Next the four quality dimensions are calculated for each candidate in the population. Using the weight given to each dimension the *overall fitness* of the process tree is calculated. In the next step certain stop criteria are tested such as finding a tree with the desired overall fitness. If none of the stop criteria are satisfied, the candidates in the population are changed and the fitness is again calculated. This is continued until at least one stop criterion is satisfied and the fittest candidate is then returned.

5.5.1. *Change Operations*

We apply three different change operations to the process trees in the population: replacement, crossover and mutation. The *replacement* operation is a rather drastic one: it replaces the worst process trees in the population with random new ones. The next change operation, *crossover*, takes two process trees and swaps two subtrees between them, resulting in two new process trees. Currently the subtrees are selected randomly. However, crossover in general does not guarantee to improve the overall quality of a process tree. Therefore we only apply it in a few cases to increase the diversity of the population.

The most important change operation we apply is that of *mutation*. Several mutation operations have been implemented where we make a distinction between random mutation and guided mutation. Currently, three random mutations have been implemented: adding a node, removing a node and changing a node. Examples of these mutations are shown in Fig. 6. None of these mutations guarantee improvement in any of the quality dimensions, since the operations are applied randomly. Furthermore, since we use binary trees in the current implementation, these mutation operations need to perform additional actions to maintain this binary form.

Using the information provided by the replay fitness measure, mutation can be applied more efficiently. For instance, when the replay fitness indicates that a certain activity is often observed in the event log but cannot be replayed in the process tree, we can add the activity in that location. Vice versa, if the replay fitness indicates that an activity, or even a whole subtree, is often skipped then this can be removed.

(a) Reference tree. (b) Adding D in parallel to A (c) Removing B (d) Change root to ∧.

Fig. 6: Examples of possible edits on a tree (a).

A combination of these operations is updating an activity to a more appropriate activity. Finally, information is also recorded about the behavior of the control-flow nodes. For instance, if we observe that an $\wedge$ operator always first executes the whole left hand side tree and then the complete right hand side tree, we can change the operator type to a $\rightarrow$ to improve precision. Please note that the guided mutation operations are applied using thresholds. Since the importance of different quality dimensions can be changed, it might be beneficial to improve precision at the cost of replay fitness for instance. None of the mutation operations guarantee improving one quality dimension without reducing another but in general improve at least one quality dimension.

Our genetic algorithm has been implemented as a plug-in for the ProM framework. We used this implementation for all experiments presented in the remainder. The algorithm stops as soon as a perfect candidate was found, i.e. with optimal fitness, or after 1.000 generations. In [15] we have shown that 1.000 generations are typically enough to find the optimal solution, especially for processes with few activities. All other settings were selected according to the optimal values presented in [15].

6. Running Example

Throughout the remainder of the paper, we use a running example, describing a simple loan application process of a financial institute, providing small consumer credit through a webpage. When a potential customer fills in a form and submits the request on the website, the process is started by activity `A` which is sending an e-mail to the applicant to confirm the receipt of the request. Next, three activities are executed in parallel. Activity `B` is a check of the customer's credit history with a registration agency. Activity `C` is a computation of the customer's loan capacity and activity `D` is a check whether the customer is already in the system. This check is skipped if the customer filled in the application while being logged in to the personal page, since then it is obsolete. After performing some computations, the customer is notified whether the loan was accepted (activity `E`, covering about 20% of the cases) or rejected (activity `F`, covering about 80% of the cases). Finally, activity `G` is performed, notifying the applicant of the outcome.

A Petri net of the loan application model is shown in Fig. 7 and the log we obtained through simulation is shown in Tab. 2.

7. Results of Current Process Discovery Algorithms

In order to validate that our small running example provides enough of a challenge for existing process discovery techniques, we applied several existing techniques, many of which resulted in *unsound* process models. We translated the behavior of each model to a process tree, in order to measure the quality of the result. Where applicable, we stayed as close as possible to the parallel behavior of the original model. Fig. 8 to 14 show the results of the various algorithms.

Table 2: The event log

Trace	#
A B C D E G	6
A B C D F G	38
A B D C E G	12
A B D C F G	26
A B C F G	8
A C B E G	1
A D B C F G	1
A D B C E G	1
A D C B F G	4
A C D B F G	2
A C B F G	1

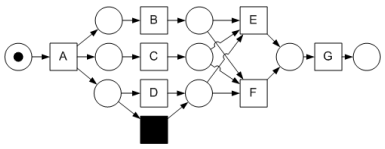
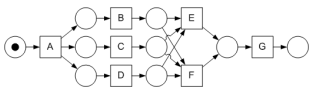


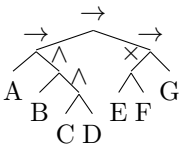
Fig. 7: Petri net of a loan application process. (`A` = send e-mail, `B` = check credit, `C` = calculate capacity, `D` = check system, `E` = accept, `F` = reject, `G` = send e-mail)

For all algorithms, unless mentioned otherwise, the publicly available version, as included in the ProM process mining framework[a], version 6.2, has been used.

7.1. *The α Algorithm*

One of the earliest process discovery algorithms is the α algorithm [6]. The α-algorithm is one of the most basic process discovery algorithms, and does not take any parameters. The result of the α algorithm is exactly the Petri net shown at the left hand side of Fig. 8. This Petri net can be easily translated to the process tree notation without changing the intended behavior. The process tree that is the result of this translation is shown at the right hand side of Fig. 8. Please note that activities `E` and `F` in the Petri net also function as a silent parallel join. In the process tree this has been split in a control flow node (as parents of `B`, `C` and `D`) and a choice between activities `E` and `F`.





f: 0.992	p: 0.995
s: 1.000	g: 0.889

Fig. 8: Result of the α algorithm [6] (sound)

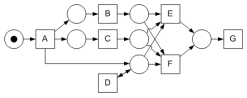
Next to the process tree obtained after conversion is a table that shows the results of the applied metrics, one for each of the four quality dimensions. The replay fitness of 0.992, indicated by the 'f' character, indicates that the process tree is able to replay almost all behavior. Only skipping activity `D`, which sometimes happens in the event log, is not possible in the process models. The precision score of 0.995, as is indicated by the 'p' character, shows that the process model does not allow for too much additional behavior. This is caused by the fact that the event log does not include all the possible orders of `B`, `C` and `D`. Simplicity is perfect (s=1.000)

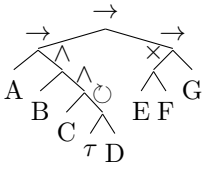
[a]The ProM Framework can be obtained via `www.promtools.org`

since all activities are included exactly once. Generalization is not bad with a score of 0.889, especially considering that generalization can only reach 1.000 in the limit.

7.2. *The ILP Miner*

A process discovery algorithm that ensures perfect replay fitness is the ILP miner [46]. The result of running this algorithm on the running example's event log is shown at the left hand side of in Fig. 9. This result is obtained by using the following, mostly default, parameters: 'Java-ILP and LpSolve' as solvers, 'Petri net (empty after completion)' as ILP variant, number of places per causal dependency and enabled the option of 'Search for separate initial places'. None of the ILP extensions were enabled. The ILP miner directly produced the Petri net as shown in Fig. 9.





f: 1.000	p: 0.784
s: 0.933	g: 0.830

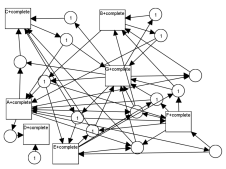
Fig. 9: Result of the ILP miner [46] (Ensuring empty net after completion, sound)

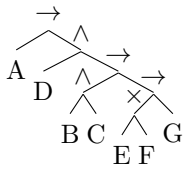
Also this Petri net can be directly translated to a process tree, without changing its behavior. The guarantee of the ILP miner, that it always produces a perfectly fitting process model, is indicated by the perfect score for replay fitness. However, this comes as the cost of precision, as is shown by the precision value of 0.784. This is caused by the model allowing for loops of activity `D` while the event log never contains more than one instance of `D` per trace. Simplicity is not perfect because a τ transition has been introduced and because of this generalization is also a bit worse than for the previous process model.

7.3. *Language-based region theory*

The result of the language-based region theory [12] can be obtained by running the ILP miner plug-in and set the number of places to 'Basic Representation', disable the 'search for separate initial places' checkbox and check the option to discover an 'Elementary Net'. This produces the Petri net that is shown in Fig. 10. It is clear to see that the resulting model is overly complex and incomprehensible.

The translation to a process tree results in the process tree as is shown in Fig. 10. Since there is no option to skip `D` the replay fitness score is only 0.992. The process model also includes activities `E`, `F` and `G` in the parallel part, which results in a precision of 0.957 since these are always strictly executed at the end of the process. Simplicity on the process tree is perfect since each activity occurs exactly





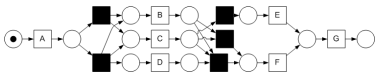
f: 0.992	p: 0.957
s: 1.000	g: 0.889

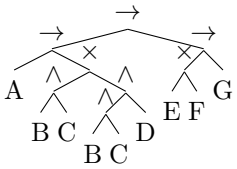
Fig. 10: Result of the language-based region theory [12] (The model is overly complex and incomprehensible, but sound)

once. Note that the translation from the Petri net to the process tree simplified the model drastically, while maintaining its behavior. Finally generalization is at the best value that can be obtained for this particular event log.

7.4. *Heuristic Miner*

The heuristics miner [45] has been developed to be more noise-resistant than most other process discovery algorithms. When applied to running example the Petri net as shown in the left hand side of Fig. 11 is obtained, after converting the Heuristics net to a Petri net. All thresholds and other default settings (e.g. 'all tasks connected' enabled and 'long distance dependency' disabled) were maintained. Other settings were experimented with but did not result in a process model with a better quality score. Unfortunately, the resulting Petri net is not sound. Tokens are left in the Petri net for instance before task `B` when the following firing sequence is executed: $\langle A, C, E, G \rangle$.





f: 1.000	p: 0.986
s: 0.875	g: 0.852

Fig. 11: Result of the heuristic miner [45] (Unsound, tokens are left behind.)

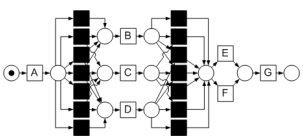
Since the Petri net is not sound, it is not possible to directly translate it to a process tree. Therefore the process tree as shown in Fig. 11 represents a sound interpretation of the intended behavior of the Petri net. Most notably is the choice between two different parallel branches, one with `D` and the other without. This is as the process model was meant as discovered by the heuristic miner, as is indicated by the two silent transitions in the beginning of the Petri net.

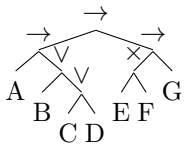
The different quality measures of the process tree show that the process tree can correctly replay all the behavior recorded in the event log. The precision is however not perfect and a bit worse than the process tree discovered by the α algorithm.

Simplicity is not perfect because of the duplication of activities `B` and `C`. Finally, generalization is low considering the other process models.

7.5. *Multi-phase Miner*

The result after running the Multi-phase miner [22] as included in ProM 5.2, with the default settings, results in an EPC model. Converting this EPC to a Petri net results in the Petri net as shown at the left hand side of hand side of Fig. 12. The process model is relaxed sound but is not easy to understand due to all the silent transitions before and after activities `B`, `C` and `D`. The process tree relations show that all these three activities are included in an $\vee$ construct, and can therefore be skipped. Although this results in a perfect replay fitness, it is not very precise since activities `B` and `C` are never skipped in the event log. Generalization is as high as it reasonably can be for this event log and simplicity is perfect.



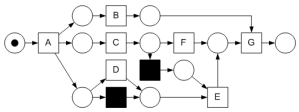


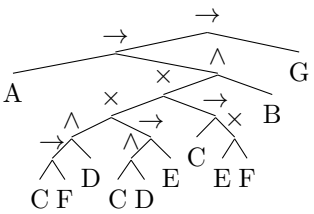
f: 1.000	p: 0.830
s: 1.000	g: 0.889

Fig. 12: Result of the Multi-phase miner [22] (Model is guaranteed "relaxed sound" and the tree reflects this.)

7.6. *The Genetic Miner*

The Genetic Miner [34] is run with a population of 20, a target fitness of 1.0 and for a maximum of $10,000$ generations. The result is an unsound Petri net since tokens are left behind before task `E` when executing the activities $\langle A, B, C, D, F, G\rangle$.





f: 1.000	p: 0.922
s: 0.737	g: 0.790

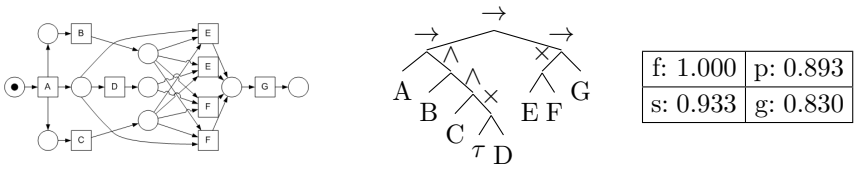
Fig. 13: Result of the genetic miner [34] (Unsound, tokens left behind.)

When translating the behavior described by this model, for example the fact that the activities `B` and `D` are parallel to `F`, we obtained the process tree as shown at the

right hand side in Fig. 13. Although replay fitness is perfect in the translated process tree, precision is mediocre. Simplicity is rather low, caused by the duplication of several activities. This also influenced generalization since the process tree contains several choices between different process model parts.

7.7. *State-based Region Theory*

Applying the state-based region theory [9,18,23,41], by executing the plug-in 'Mine Transition System' followed by a conversion to Petri nets using region theory, results in the Petri net as shown in Fig. 14. For the transition system mining the default settings the maximum set size has been set to unlimited and all activities have been included.



f: 1.000	p: 0.893
s: 0.933	g: 0.830

Fig. 14: Result of the state-based region theory

The resulting Petri net is sound and includes the option to execute activities `E` and `F` without executing `D` explicitly by duplicating activities `E` and `F`. We have translated this Petri net to a process tree without duplicating activities `E` and `F`, which would have reduced simplicity and generalization. Replay fitness of the resulting process tree is perfect. However, precision is rather low because of the explicit choice to skip `D`. In this translation simplicity and generalization are rather high.

7.8. *Why Existing Algorithms Fail*

The results of existing process discovery algorithms, as shown in Fig. 8 to 14, clearly indicate that, even on our small example, only the α-algorithm was able to balance the four quality dimensions well. Several algorithms even produce an unsound result. Moreover, the α-algorithm was "lucky" for this small example. In general, this algorithm produces models that are not fitting or not precise. Therefore, in Section 8, we first investigate combining various dimensions and show that *all of them have to be considered in order to discover a sound, easy to understand process model, accurately describing the log under consideration.* In Subsection 8.4 we show that assigning different weights to the dimensions results in different process models. Then, in Section 9, we show that only our ETM algorithm is able to balance all quality dimensions for real life event logs.

8. Ignoring Quality Dimensions

The examples shown in figures 8 to 14 show that various existing process mining techniques perform differently on all four quality dimensions and they often provide unsound models. In this section, we use the ETM algorithm to discover a process model on the given log, while varying which of the quality dimensions should be considered. We show some optimal models that resulted from different runs of the algorithm and discuss their properties. For each result we show the process tree, the scores for each of the four quality dimensions and the translation of the process tree to BPMN.

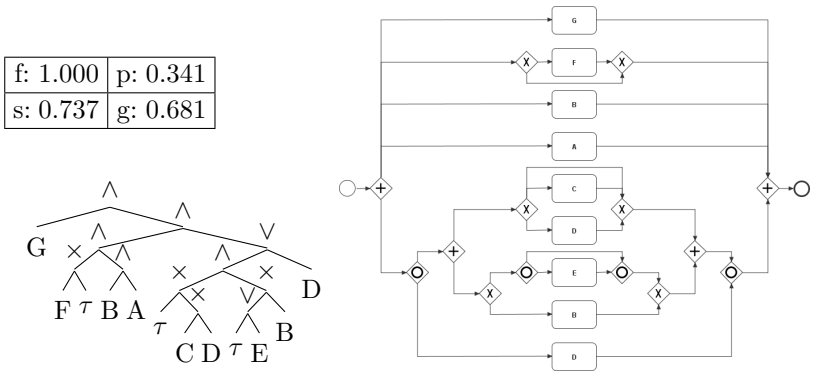
8.1. *Considering Only One Quality Dimension*

Fig. 15a shows an example process tree, and the corresponding BPMN model, that was discovered when focusing solely on the *replay fitness* dimension. Although the process model is able to replay the event log perfectly, the model allows for more behavior than is seen in the event log. Since adding parts to the process tree might improve replay fitness, and removing parts never does, the process tree will keep growing until perfect replay fitness is reached. This is bad for simplicity since activities will be duplicated (activities `B` and `D` in Fig. 15a) and certain parts of the tree will never be used (the rightmost `B` and the leftmost `D` are never used when replaying the event log). And although the BPMN process model looks very structured, it is harder to understand than the process model used to describe the running example.

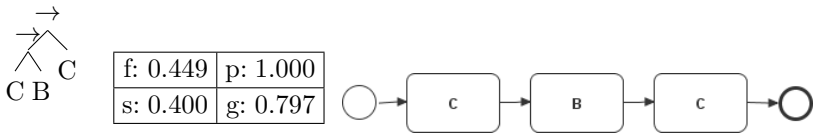
In order to obtain trees that do not allow for behavior that is not observed in the event log, we considered only the *precision* dimension in the genetic algorithm. An example of such a tree is shown in Fig. 15b, which has a perfect precision because it can only generate the trace $\langle C, B, C \rangle$. The translation to BPMN shows how simple the model really is and that it indeed can only produce one trace. A process tree will have perfect precision if each trace it can generate is used in an alignment. Since the tree of Fig. 15b can only generate one trace, each alignment between the event log and the tree uses this path of execution. However, the low replay fitness score indicates that the process model in Fig. 15b has little to do with the behavior that is recorded in the event log.

When only considering *simplicity*, we get trees such as the one in Fig. 15c, where each activity is included exactly once. The corresponding BPMN model nicely demonstrates this. However, the process model does not really describe the observed process executions in the event log well, which is indicated by the low scores on replay fitness and precision.

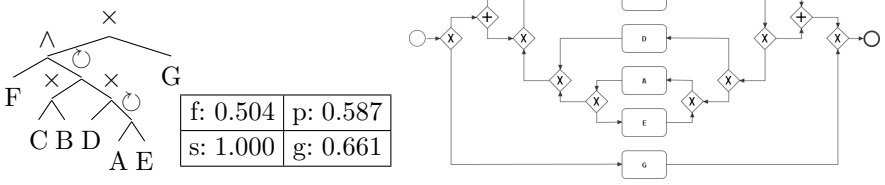
Fig. 15d shows the process tree and BPMN model that has the best generalization score when solely focusing on generalization. As mentioned before, generalization cannot reach 1, as this would require all possible behavior to be observed infinitely often. Since generalization takes the number of node visits into account, the score is improved if nodes are visited more often, where the visits are again measured on the closest matching execution of the model for each trace. By placing



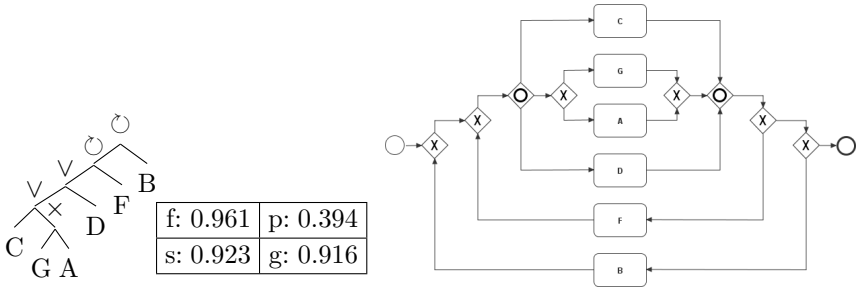
(a) Only replay fitness



(b) Only Precision



(c) Only Simplicity



(d) Only Generalization

Fig. 15: Process trees discovered when considering each of the quality dimensions separately

↺ operators high in the tree, and activities `F` and `B` in the 'redo' part, the loops and the nodes in the 'do' part are executed more often, hence improving generalization.

8.2. *Always Considering Replay Fitness*

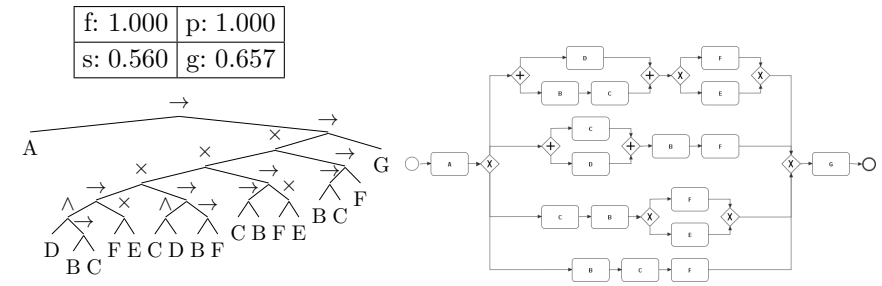
The discussion in the previous section showed that none of the quality dimensions should be considered in isolation. Furthermore, we validated the choice of many existing process discovery techniques to put emphasis on replay fitness, i.e. if the replay fitness is not good enough, the other quality dimensions add little value as the discovered model does not describe the recorded behavior. On the other hand, achieving a perfect replay fitness is not always necessary or desired.

When focusing on *replay fitness and precision*, the goal is to find a process model that describes all traces, and not much more, much like the region-based algorithms the results of which are depicted in Fig. 9, 10 and 14. In general, a model that contains an initial exclusive choice between all unique traces in the log has perfect precision and replay fitness. Each choice is taken at least once and each trace in the event log is a sequence in the process model. This always results in a perfect replay fitness. For our running example the process tree and BPMN model as shown in Fig. 16a also have both a perfect replay fitness and precision. Each part of the process tree is used to replay a trace in the event log and no behavior that is not present in the event log can be produced by the process tree. However, since both process models are fairly big, the simplicity score is low and more importantly, the generalization is not very high either. This implies that, although this model is very precise, it is not likely that it explains any future, unseen behavior.

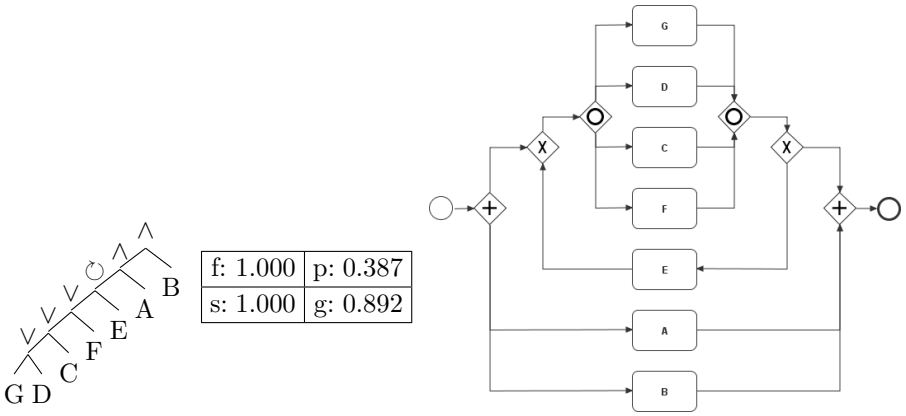
Next we consider *replay fitness and simplicity*, the result of which is shown in Fig. 16b. When considering only replay fitness, we obtained fairly large models, while simplicity should keep the models compact. The process models discovered when considering both replay fitness and simplicity contain each activity exactly once and hence has perfect simplicity. At the same time all traces in the event log can be replayed. However, the process tree contains two ∧, one ↺ and three ∨ nodes that allow for (far) more behavior than is seen in the event log. This is reflected in the low precision score in combination with the high generalization.

The process tree and BPMN model that are found when focusing on the combination of *replay fitness and generalization* is shown in Fig. 16c. The process tree shows many similarities with the process tree found when solely considering generalization. Activity `E` has been added to the 'do' part of the ↺ to improve the replay fitness. However, it also reduces the generalization dimension since it is only executed 20 times. Furthermore, the tree is still not very precise.

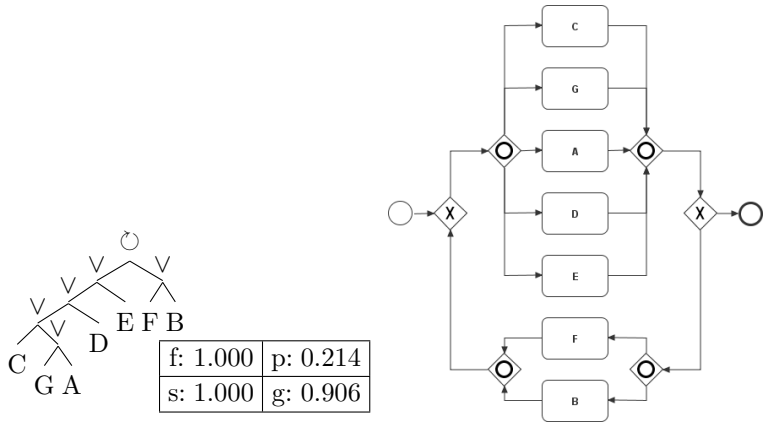
In contrast to the trees in Section 8.1, the various process trees discussed in this section mainly capture the behavior observed in the event log. However, they either are overfitting (i.e. they are too specific) or they are underfitting (i.e. they are too generic). Hence, considering replay fitness in conjunction with only one other dimension still does not yield satisfying results. Therefore, in Section 8.3, we



(a) Replay Fitness and Precision



(b) Replay Fitness and Simplicity



(c) Replay Fitness and Generalization

Fig. 16: Process trees discovered when considering replay fitness and one of the other quality dimensions

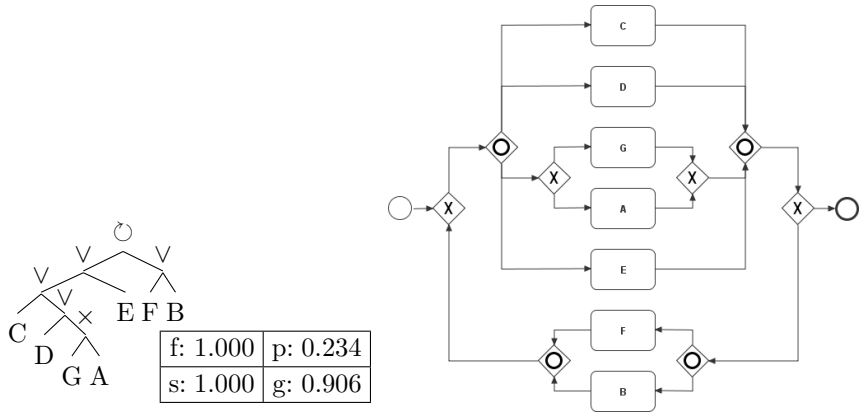
consider three out of four dimensions, while never ignoring replay fitness.

8.3. *Ignoring One Dimension*

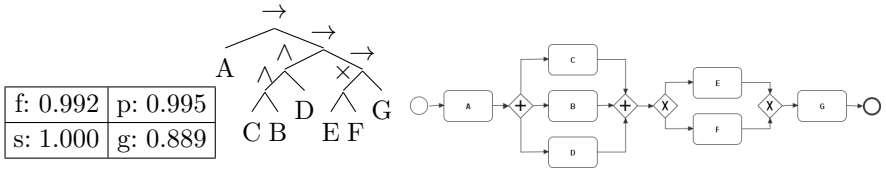
We just showed that replay fitness, in conjunction with one of the other quality dimensions, is insufficient to judge the quality of a process model. However, most process discovery algorithms steer on just two quality dimensions. Hence we consider three of the four quality dimensions.

Fig. 17 shows the three process trees that are discovered when ignoring one of the quality dimensions, but always including replay fitness. Fig. 17a shows the process tree and BPMN model found when ignoring precision. The resulting process tree is similar to the one in Fig. 16c, which was based on replay fitness and generalization only. The only difference is that the parent of `A` and `G` has changed from $\vee$ to $\times$. Since the $\vee$ was actually only used as an $\times$, this only influences precision. Hence the other measures have the same values. The BPMN models of Fig. 16c and Fig. 17a are also very similar where the latter one only has an additional $\times$ block.

The process tree which is discovered when ignoring generalization is the same a when simplicity is ignored and is shown in Fig. 17b. This is due to the fact that both simplicity and generalization are optimal in this tree. In other words, when weighing all four dimensions equally, this tree is the best possible process tree to



(a) no precision



(b) no generalization or no simplicity

Fig. 17: Considering 3 of the 4 quality dimensions

describe the process.

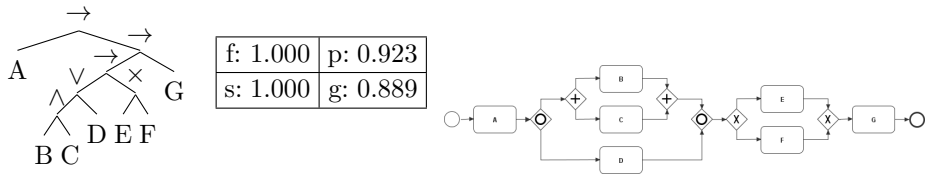
Interestingly, this tree is the same as the result of the α-algorithm (Fig. 8). However, as mentioned earlier, the α-algorithm is not very robust. This will also be demonstrated in Section 9 using real life event logs.

8.4. *Weighing Dimensions*

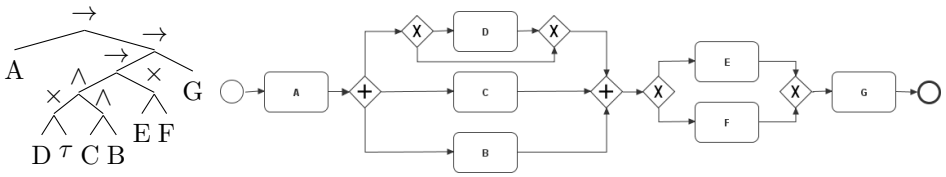
The process trees shown in Fig. 17 have trouble replaying all traces from the event log while maintaining a high precision. However, since process discovery is mostly used to gain insights into the behavior recorded in the log, it is generally required that the produced model represents the log as accurately as possible, i.e. that both replay fitness and precision are high. By giving more weight to replay fitness, while still taking precision into account, our genetic algorithm can accommodate this importance. Fig. 18a shows the process tree and BPMN model resulting from our algorithm when giving 10 times more weight to replay fitness than the other three quality dimensions. As a result the process tree is able to replay all traces from the event log while still maintaining a high precision.

Let us compare this process with the process tree of Fig. 17b. This is also the process tree or BPMN model produced when all quality dimensions are weighted equally. It can be seen that the price to pay for improving fitness was a reduction in precision. This can be explained by looking at the change made to the process model: activity `D` is now in an $\vee$ relation with activities `B` and `C`. Replay fitness is hereby improved since the option to skip activity `D` is introduced. However, the process tree now also allows for skipping the execution of both `B` and `C`. Something which is never observed in the event log.

Furthermore, the process tree of Fig. 18a performs better than the model we



(a) Process tree discovered when replay fitness is 10 times more important than all other dimensions



(b) Process tree of the model used for simulation (Translated manually from Fig. 7)

Fig. 18: Weighing dimensions and original process model.

originally used for simulating the event log as can be seen in Fig. 18b. The original tree performs equal on replay fitness but worse on the other three quality dimensions. Precision is worse because the state space of the original model is bigger while some paths are not used. Simplicity is also worse because an additional τ node is used in the original tree, hence the tree is two nodes bigger than optimal. Furthermore, since the τ node is only executed ten times, the generalization reduces as well because the other nodes are executed more than 10 times, thus the average visits per node decreases.

9. Experiments Using Real Life Event Logs

In the previous sections we discussed the results of various existing process discovery techniques on our running example. We also demonstrated that all four quality dimensions should be considered when discovering a process model. In this section we apply a selection of process discovery techniques, and our ETM algorithm, on three event logs from real information systems. Using these event logs, and the running example, we show that our ETM algorithm is more robust than existing process discovery techniques.
In this section we consider the following event logs:

(1) The event log **L0** is the event log as presented in Tab. 2. L0 contains 100 traces, 590 events and 7 activities.
(2) **Event Log L1** contains 105 traces, 743 events in total, with 6 different activities.
(3) **Event Log L2** contains 444 traces, 3.269 events in total, with 6 different activities.
(4) **Event Log L3** contains 274 traces, 1.582 events in total, with 6 different activities.

Event logs L1, L2 and L3 are extracted from information systems of municipalities participating in the CoSeLoG[b] project. Since some of the existing process discovery techniques require a unique start and end activity, all event logs have been filtered to contain only those traces that start with the most common start activity and end with the most common end activity. Furthermore, activity names have been renamed to the letters `A...F`.

From the process discovery algorithms discussed in Section 7 we selected four well-known algorithms: the α-algorithm [6], the ILP-miner [46], the heuristics miner [45] and the genetic algorithm by Alves de Medeiros [34]. Because we do not have enough space to show all process models we show some important characteristics of the resulting Petri nets in Tab. 3.

The α-algorithm and ILP Miner produce sound Petri nets for each of the four input logs. In contrast, the Genetic Miner never produces a sound Petri net for the

[b]See http://www.win.tue.nl/coselog

Table 3: Petri Net properties of discovered models.
Legend: s?: whether the model is sound (✓) or unsound (✗);
#p: number of places; #t: number of transitions; #arcs: number of arcs

	L0				L1				L2				L3			
	s?	#p	#t	#arcs	s?	#p	#t	#arcs	s?	#p	#t	#arcs	s?	#p	#t	#arcs
α-algorithm	✓	9	7	20	✓	3	6	4	✓	3	6	4	✓	6	6	10
ILP Miner	✓	7	7	19	✓	4	6	9	✓	2	6	11	✓	4	6	9
Heuristics	✗	12	12	30	✓	12	15	28	✓	12	16	32	✗	10	11	23
Genetic	✗	10	9	21	✗	13	20	42	✗	11	20	36	✗	10	11	25

Table 4: Quality of Process Tree translations of Several Discovery Algorithms (*italic* results indicate unsound models, the best model is indicated by a gray cell background)

	L0		L1		L2		L3	
α-algorithm	f: 0.992	p: 0.995	f: 1.000	p: 0.510	f: 1.000	p: 0.468	f: 0.976	p: 0.532
	s: 1.000	g: 0.889	s: 0.923	g: 0.842	s: 0.923	g: 0.885	s: 0.923	g: 0.866
	overall: 0.969		overall: 0.819		overall: 0.819		overall: 0.824	
ILP Miner	f: 1.000	p: 0.748	f: 1.000	p: 0.551	f: 1.000	p: 0.752	f: 1.000	p: 0.479
	s: 0.933	g: 0.830	s: 0.857	g: 0.775	s: 0.923	g: 0.885	s: 0.857	g: 0.813
	overall: 0.887		overall: 0.796		overall: 0.890		overall: 0.787	
Heuristics	*f: 1.000*	*p: 0.986*	f: 0.966	p: 0.859	f: 0.917	p: 0.974	*f: 0.995*	*p: 1.000*
	s: 0.875	*g: 0.852*	s: 0.750	g: 0.746	s: 0.706	g: 0.716	*s: 1.000*	*g: 0.939*
	overall: 0.928		overall 0.830		overall: 0.828		*overall: 0.983*	
Genetic	*f: 1.000*	*p: 0.922*	*f: 0.997*	*p: 0.808*	*f: 0.905*	*p: 0.808*	*f: 0.987*	*p: 0.875*
	s: 0.737	*g: 0.790*	*s: 0.750*	*g: 0.707*	*s: 0.706*	*g: 0.717*	*s: 0.750*	*g: 0.591*
	overall: 0.862		*overall: 0.815*		*overall: 0.784*		*overall: 0.801*	
ETM	f: 0.992	p: 0.995	f: 0.901	p: 0.989	f: 0.863	p: 0.982	f: 0.995	p: 1.000
	s: 1.000	g: 0.889	s: 0.923	g: 0.894	s: 0.923	g: 0.947	s: 1.000	g: 0.939
	overall: 0.969		overall: 0.927		overall: 0.929		overall: 0.983	

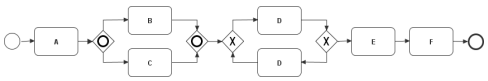
event logs. The Heuristics Miner produces a sound solution for two out of the four event logs.

For each of the discovered Petri nets we created process tree representations, describing the same behavior. If a Petri net was unsound, we interpreted the sound behavior as closely as possible. For each of these process trees the evaluation of each of the four measures, and the overall average fitness, is shown in Tab. 4.

For event log L1 both the α-algorithm and the ILP miner find process models that can replay all behavior. But, as is also indicated by the low precision, these allow for far more behavior than observed in the event log. This is caused by transitions without input places that can occur an arbitrary number of times. The heuristics miner is able to find a reasonably fitting process model, although it is also not very precise since it contains several loops. The genetic algorithm finds a model similar to that of the heuristics miner, although it is unsound and contains even more loops. The ETM algorithm finds a process tree, of which the BPMN translation is shown in Fig. 19a, that scores high on all dimensions. If we want to improve replay fitness even more we can make it 10 times more important as the other quality dimensions. This results in the process model shown in Fig. 19b. With an overall (unweighted) fitness of 0.884 it is better than all process models found by other algorithms while at the same time having a perfect replay fitness.

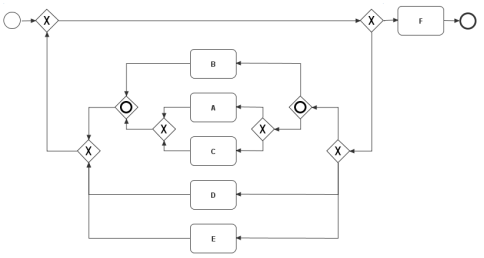
Event log L2 shows similar results: the α-algorithm and the ILP miner are able to find process models that can replay all behavior but allow for far more behavior. The heuristics miner and genetic miner again found models with several loops. The ETM algorithm was able to find a process model, shown in Fig. 20a, that scores high on all dimensions but less so on replay fitness. If we emphasize replay fitness 10 times more than the other dimensions, we get the process model shown

f: 0.901	p: 0.989
s: 0.923	g: 0.894



(a) All dimensions weight 1

f: 0.996	p: 0.775
s: 0.923	g: 0.843



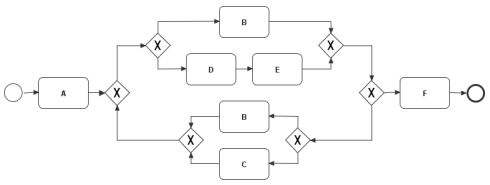
(b) Replay Fitness weight 10, rest 1

Fig. 19: Process Trees discovered for L1

in Fig. 20b. Although replay fitness improved significantly, the other dimensions, especially precision and simplicity, are reduced.

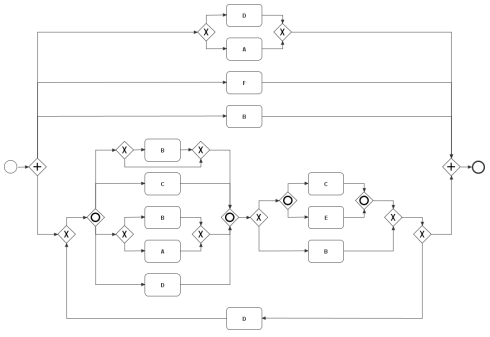
For event log L3 the observations for the last two event logs still hold. Both the α-algorithm and the ILP miner provide fitting process models that allow for far more behavior. Both the heuristics miner and the genetic algorithm result in unsound models. However, the sound interpretation of the heuristics model is the same as the sound process model found by the ETM algorithm, which is shown in Fig. 21a. Replay fitness is almost perfect. However, we let the ETM algorithm discover a process model with real perfect replay fitness, which is shown in Fig. 21b. This requires making replay fitness 1.000 times more important than the others and results in a process tree that has perfect replay fitness but scores bad on precision. However, as we have seen before, this is a common trade-off and the process tree is still more precise than the one found by the ILP miner which also has a perfect replay fitness.

Investigating the results shown in Tab. 4 we see that on two occasions a process model similar to the one found by the ETM algorithm was found by another algorithm. However, the α-algorithm was not able to produce sensible models for any of the three real life event logs. The heuristics miner once produced a process model of which the sound behavior matched the process tree the ETM algorithm discovered. However, our algorithm always produced sound process models superior



f: 0.863	p: 0.982
s: 0.923	g: 0.947

(a) All dimensions weight 1



f: 0.964	p: 0.415
s: 0.571	g: 0.838

(b) Replay fitness weight 10, rest weight 1

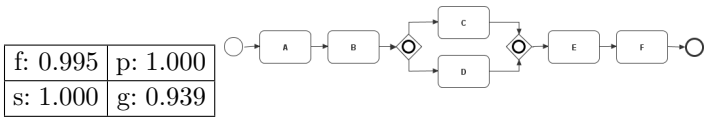
Fig. 20: Process Trees discovered for L2

to the others. Furthermore, the ETM algorithm can be steered to improve certain dimensions of the process model as desired.

10. Building a Pareto Front

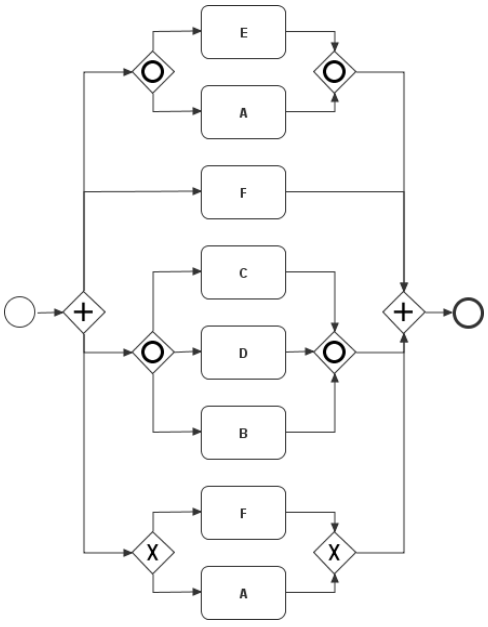
By assigning weights to the different dimensions the ETM algorithm is able to produce a process model that balances the dimensions as desired. However, it is hard to specify the weights required beforehand. Consider for instance the process model of Fig. 21b where fitness was weighed $1,000$ times more than the other dimensions in order to get perfect replay fitness. In Fig. 19b and Fig. 20b however, fitness was weighed 10 times more important than the other dimensions to get a similar result. It is sometimes hard to know beforehand how to set the weights to get the desired results.

Therefore, we prefer to avoid assigning weights to the different dimensions up-front. By presenting the user with a collection of process models to choose from,



f: 0.995	p: 1.000
s: 1.000	g: 0.939

(a) All dimensions weight 1



f: 1.000	p: 0.502
s: 0.857	g: 0.900

(b) Replay fitness weight 1000, rest weight 1

Fig. 21: Process Trees discovered for L3

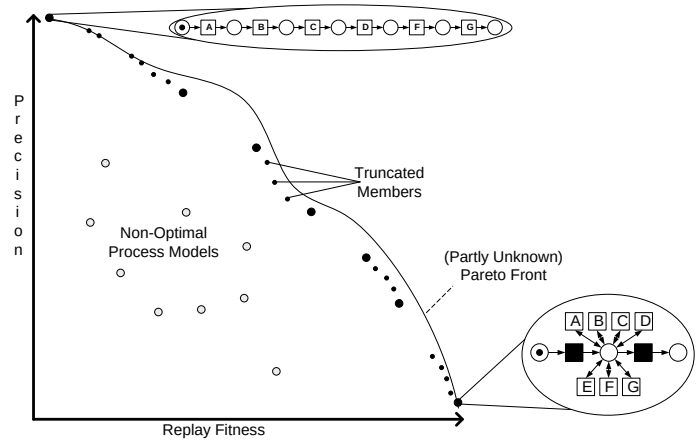


Fig. 22: Example of a Pareto front on two dimensions.

the user can pick the process model that has the desired trade-offs. This can be achieved by constructing a Pareto front [42] of process models. Fig. 22 shows an example of a Pareto front for the dimensions replay fitness and precision. Each dot in the graph represents a process model with a certain replay fitness and precision value. The open dots in the lower middle area are non-optimal process models, e.g. one of the dimensions can be improved without reducing the quality in any of the other dimensions. The solid black dots represent the current estimation of the Pareto front. Currently, for these models there is no model known where one dimension has a better score without reducing one of the other dimensions. The ideal or real Pareto front, as indicated by the dotted curved line, shows that some improvements can still be made.

However, the Pareto front can grow very large because in general at least 4 dimensions are considered which each have infinitely many possible values. Therefore, the Pareto front can be truncated by removing process models that are similar, i.e., representative examples are selected from groups of similar models. The bigger dots shown in Fig. 22 are the most diverse process models in the current front. This is determined by looking at the distance between the process models in all dimensions. When the Pareto front is truncated process models that are too similar to others are removed, until the desired Pareto front size is obtained.

The Pareto front can be easily constructed by the ETM algorithm by replacing the group of elite process trees with the Pareto front and updating it during the generations. Since the Pareto front size is not fixed, and in general will grow over time, the ETM algorithm is slightly adjusted to select a fixed number of process trees from the current Pareto front as input for the new mutation cycle. After applying the different mutations, the new trees are considered to be added to the Pareto front and the Pareto front is updated.

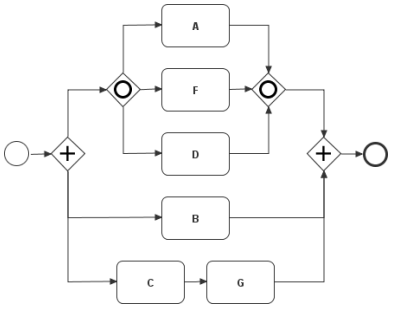
10.1. *Pareto Front for the Running Example*

If a Pareto front is constructed for the running example, using the process models discovered so far by the ETM algorithm, it contains six of the eleven process models shown in Fig. 15 through Fig. 18. It will not contain the process model discovered when only replay fitness is taken into account, which is shown in Fig. 15a. This process tree is dominated by the process tree discovered when considering both replay fitness and precision (Fig. 16a) since they both have perfect replay fitness but the precision of the latter model is better. For the reason the process tree found when considering replay fitness and simplicity (Fig. 16b) is included since it scores better on simplicity. Another process tree that is in the Pareto front is the tree discovered when ignoring precision (Fig. 17a), and the one found when considering only precision (Fig. 15b. The process tree that is discovered when ignoring either simplicity or generalization, which is the same as the one discovered by the α algorithm is also included in the Pareto front.

Additionally, the process tree as shown in Fig. 23 will be included in the Pareto front since it balances the four quality dimensions very well.

When considering the process models found by the existing process discovery algorithms, some of them are also in the Pareto front. The model discovered by the α-algorithm is on the Pareto front: it is the same as the process tree found by the ETM algorithm. However, also the sound interpretation of the model discovered by the Heuristics miner (Fig. 11) is included in the Pareto front because of the balance of the different quality dimensions.

A Pareto front changes over time, i.e. during the execution of the ETM algorithm, process models may disappear from it as they are replaced by better ones. Therefore, the Pareto front as discussed may be improved if the ETM algorithm is allowed to run longer.



f: 0.980	p: 0.613
s: 0.923	g: 0.898

Fig. 23: Process Tree that was not discovered before but will be in the Pareto Front.

10.2. *Using the Pareto front to indicate non-local dependencies*

Another issue that is addressed by returning multiple process models is that of modeling non-local dependencies using process trees. Consider for instance a process for obtaining a driving license for either a car or a motorbike. First driving classes for either car or motorbike need to be followed. After this a theoretical exam needs to be taken, which is the same for both. Next the practical test needs to be done, which is of course different for cars and motorbikes. Two process models describing this process are shown in Fig. 24, with the activities renamed to `A` to `J`. In this example process activity `F` (the practical exam for cars) should only be executed if activity `C` (driving class for cars) was executed before. Similarly, activity `G` (practical exam for motorbike) should only be chosen if earlier activity `D` (driving class for motorbike) was executed. However, in between always activity `E` (theoretical exam) should be executed.

The Pareto front constructed by the ETM algorithm will include both process models shown in Fig. 24. In the process model of Fig. 24a the dependencies between the activities are not modeled and activity `F` could be executed even if activity `D` has been performed earlier. This is not possible in the process model of Fig. 24b where activity `E` has been duplicated. The process model of Fig. 24b has a perfect precision. The process model of Fig. 24a does not have a perfect precision since this process model can produce traces not observed in the event log. However, this process model scores better on the dimensions generalization and simplicity.

In general, the process model where the non-local dependency is expressed by duplicating the activities in between scores better on precision. This process model restricts certain activity combinations from occurring which are not observed in the event log, hence increasing precision. This however always comes at the cost of both generalization and simplicity. Therefore the Pareto front will always include both process model variants. This allows the user to make the decision which of the two process models to prefer.

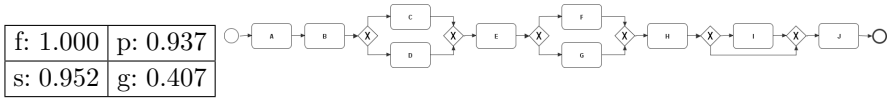


Fig. 24: Two process trees where activities F and G depend on C and D respectively.

11. Conclusion

The quality of process discovery algorithms is generally measured using four dimensions, namely replay fitness, precision, generalization and simplicity. Many existing process discovery algorithms focus on only two or three of these dimensions and generally, they do not allow for any parameters indicating to what extent they should focus on any of them. Furthermore, most process discovery algorithms assume that the event log is complete in describing the behavior of the system. Moreover, they assume that the event log is noise-free and that the process model is a correct description of the system.

In this paper we provide an overview of the relationship between the behavior of the system, the observed behavior in the event log and the possible behavior of the process model. The existing four quality dimensions are positioned within this comparison and we present the ETM algorithm to discover process trees on a log which *can be configured* to optimize for a weighted average over the quality dimensions, i.e. a model can be discovered that is optimal given the weights to each parameter. Finally, the ETM algorithm is *guaranteed to produce sound process models*.

We use our ETM algorithm to show that *all four quality dimensions are necessary* when doing process discovery and that none of them should be left out. However, the replay fitness dimension, indicating to what extent the model can reproduce the traces in the log, is more important than the other dimensions.

Using both an illustrative example and three real life event logs we demonstrate the need to consider all four quality dimensions. Moreover, our algorithm is able to balance all four dimensions is a seamless manner.

To prevent the need to configure the weights assigned to each quality dimension, a Pareto front can be constructed by the ETM algorithm. The Pareto front contains a number of process trees, each balancing the quality dimensions in a different way. Currently the main challenge is to find a way to present the Pareto front to the user such that they can understand the differences between the process models and make an informed choice which process model to select.

References

1. W.M.P. van der Aalst. On the Representational Bias in Process Mining. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on*, pages 2–7. IEEE, 2011. pages
2. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011. pages
3. W.M.P. van der Aalst. Mediating Between Modeled and Observed Behavior: The Quest for the "Right" Process. In *IEEE International Conference on Research Challenges in Information Science (RCIS 2013)*, pages 31–43. IEEE Computing Society, 2013. pages
4. W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012. pages

5. W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011. pages
6. 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. pages
7. A. Adriansyah, B. van Dongen, and W.M.P. van der Aalst. Conformance Checking using Cost-Based Fitness Analysis. In *IEEE International Enterprise Computing Conference (EDOC 2011)*, pages 55–64. IEEE Computer Society, 2011. pages
8. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer-Verlag, Berlin, 1998. pages
9. E. Badouel and P. Darondeau. Theory of Regions. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer-Verlag, Berlin, 1998. pages
10. J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, Cambridge, 1990. pages
11. F.D. Banzhaf, W.and Francone, R.E. Keller, and P. Nordin. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998. pages
12. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007. pages
13. C. Boender and A. Rinnooy Kan. A Bayesian Analysis of the Number of Cells of a Multinomial Distribution. *The Statistician*, 32(1-2):240–248, 1983. pages
14. J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In *OTM Federated Conferences, 20th International Conference on Cooperative Information Systems (CoopIS 2012)*, volume 7565, pages 305–322, 2012. pages
15. J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. A Genetic Algorithm for Discovering Process Trees. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1 –8, june 2012. pages
16. J.A. Bunge and M. Fitzpatrick. Estimating the Number of Species: A Review. *Journal of the American Statistical Association*, 88(421):pp. 364–373, 1993. pages
17. K.P. Burnham and W.S. Overton. Robust Estimation of Population Size When Capture Probabilities Vary Among Animals. *Ecology*, 60(5):pp. 927–936, 1979. pages
18. M.P. Cabasino, A. Giua, and C. Seatzu. Identification of Petri Nets from Knowledge of Their Language. *Discrete Event Dynamic Systems*, 17(4):447–474, 2007. pages
19. T. Calders, C. W. Günther, M. Pechenizkiy, and A. Rozinat. Using Minimum Description Length for Process Mining. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 1451–1455, New York, NY, USA, 2009. ACM. pages
20. 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. pages
21. 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. pages

22. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Mining: Aggregating Instances Graphs into EPCs and Petri Nets. In D. Marinescu, editor, *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 35–58. Florida International University, Miami, Florida, USA, 2005. pages
23. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368, 1989. pages
24. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003. pages
25. K. Gerke, J. Cardoso, and A. Claus. Measuring the compliance of processes with reference models. In Robert Meersman, Tharam Dillon, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2009*, volume 5870 of *Lecture Notes in Computer Science*, pages 76–93. Springer Berlin Heidelberg, 2009. pages
26. S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust Process Discovery with Artificial Negative Events. *Journal of Machine Learning Research*, 10:1305–1340, 2009. pages
27. G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering Expressive Process Models by Clustering Log Traces. *IEEE Transaction on Knowledge and Data Engineering*, 18(8):1010–1027, 2006. pages
28. C.W. Günther and W.M.P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer-Verlag, Berlin, 2007. pages
29. J. Herbst and D. Karagiannis. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 9:67–92, 2000. pages
30. A.R. Hevner, S.T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Q.*, 28(1):75–105, March 2004. pages
31. Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing Journal*, 9(1):3–12, 2005. pages
32. J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992. pages
33. M. Kunze, M. Weidlich, and M. Weske. Behavioral similarity - a proper metric. In *Business Process Management (BPM 2011)*, pages 166–181, 2011. pages
34. A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007. pages
35. J. Mendling, G. Neumann, and W.M.P. van der Aalst. Understanding the Occurrence of Errors in Process Models Based on Metrics. In F. Curbera, F. Leymann, and M. Weske, editors, *Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007)*, volume 4803 of *Lecture Notes in Computer Science*, pages 113–130. Springer-Verlag, Berlin, 2007. pages
36. J. Mendling, H.M.W. Verbeek, B.F. van Dongen, W.M.P. van der Aalst, and G. Neumann. Detection and Prediction of Errors in EPCs of the SAP Reference Model. *Data and Knowledge Engineering*, 64(1):312–329, 2008. pages
37. J. Munoz-Gama and J. Carmona. Enhancing Precision in Process Conformance: Stability, Confidence and Severity. In N. Chawla, I. King, and A. Sperduti, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, Paris,

France, April 2011. IEEE. pages
38. OMG. Business Process Model and Notation (BPMN). Object Management Group, dtc/2010-06-05, 2010. pages
39. H. Rittel and M. Webber. Dilemmas in a General Theory of Planning. *Policy Sciences*, 4(2):155–169, 1973. pages
40. A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008. pages
41. M. Sole and J. Carmona. Process Mining from a Basis of Regions. In J. Lilius and W. Penczek, editors, *Applications and Theory of Petri Nets 2010*, volume 6128 of *Lecture Notes in Computer Science*, pages 226–245. Springer-Verlag, Berlin, 2010. pages
42. D.A. Van Veldhuizen and G.B. Lamont. Evolutionary Computation and Convergence to a Pareto Front. In *Late Breaking Papers at the Genetic Programming 1998 Conference*, pages 221–228, 1998. pages
43. J. Vanhatalo, H. Völzer, and J. Koehler. The Refined Process Structure Tree. *Data and Knowledge Engineering*, 68(9):793–818, 2009. pages
44. M. Weidlich, A. Polyvyanyy, N. Desai, and J. Mendling. Process compliance measurement based on behavioural profiles. In *Advanced Information Systems Engineering*, pages 499–514. Springer, 2010. pages
45. A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. Alves de Medeiros. Process Mining with the Heuristics Miner-algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology, Eindhoven, 2006. pages
46. J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 94:387–412, 2010. pages