On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery

Joos C.A.M. Buijs, Boudewijn F. van Dongen, and Wil M.P. van der Aalst

Eindhoven University of Technology, The Netherlands
{j.c.a.m.buijs,b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl

Abstract. 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 many other metrics 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, several metrics exist to measure the complexity of a model irrespective of the log.

In this paper, we 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 can not steer the discovery process based on user-defined weights for the four quality dimensions.

This paper also 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.

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.

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 focussing on the properties of the models produced, while at the same time the applicability of various techniques have been compared in case-studies.

Figure 1 shows four quality dimensions generally used to discuss results of process discovery techniques, namely:

Replay Fitness. Replay fitness quantifies the extent to which the discovered model can accurately reproduce the cases recorded in the log. Typical algorithms guaranteeing perfect replay fitness are region-based approaches [7, 21]

R. Meersman et al. (Eds.): OTM 2012, Part I, LNCS 7565, pp. 305–322, 2012.

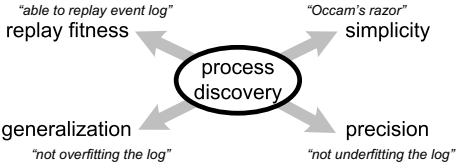


Fig. 1. Different quality dimensions for Process Model Discovery [3]

and the multi-phase miner [10]. Other techniques, such as the heuristics miner [19] and some genetic algorithms [14] use replay fitness as their guiding principle when discovering a process model, but do not guarantee optimal results.

Simplicity. The complexity of a process model is captured by the simplicity dimension. Process discovery algorithms often result in spaghetti-like process models [3], which are process models that are very hard to read. A class of process discovery algorithms that strongly focusses on simplicity is the class of α-algorithms [3, 11, 20], derived from the original α algorithm [5]. These discovery techniques generally result in simple models, but with poor replay fitness and/or precision.

Precision. It is trivial to discover a simple process model that can reproduce the event log. Such a model is generally referred to as the flower-model [16] and is an extreme example of an underfitting process model. A flower model is able to produce any arbitrary finite sequence of events. Therefore, precision quantifies the fraction of the behavior allowed by the model which is not seen in the event log. The region-based algorithms mentioned before [7, 21] are good examples of algorithms that guarantee optimal precision, i.e. they guarantee to allow only minimally more behavior than seen in the log.

Generalization. Finally, generalization assesses the extent to which the resulting model will be able to reproduce future behavior of the process. In that sense, generalization can also be seen as a measure for the confidence on the precision. Consider for example a very precise model that captures each individual case in the log as a separate path in the model. If there are many possible paths, it is unlikely that the next case will fit. Examples of generalizing algorithms are the fuzzy miner [13] and the heuristics miner [19].

The overview above shows that many process discovery algorithms focus on one or two of the dimensions. However, none of these algorithms is able to guide the result towards a particular dimension.

In this paper, we also present the ETM algorithm, which is a genetic algorithm able to optimize the process discovery result towards any of the four dimensions. By making use of so-called *process trees* [8] this algorithm ensures that the resulting model is a sound process model describing the observed log, while at the same time, the model is *optimal with respect to a weighted average over replay fitness, simplicity, precision and generalization.* Using the ETM algorithm, we

can easily explore the effects of focussing on one dimension in isolation and on combinations of these dimensions.

The remainder of this paper is structured as follows. In Section 2, we present our ETM algorithm. Furthermore, we present one metric for each of the four quality dimensions and we present process trees as a convenient means of modeling sound process models. In Section 3, 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 metrics. Section 4 then shows the results of focussing on a subset of the quality dimensions during process discovery. Here, we use our ETM algorithm to show that such a narrow focus results in poor models. Section 5 shows the result when considering all dimensions. In Section 6 we apply existing techniques and our ETM algorithm on several real life event logs. Section 7 concludes the paper.

2 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.

2.1 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*, 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.

Figure 2 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. 2 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) [17]. The RPST approach provides "a modular technique of workflow graphs parsing to obtain fine-grained fragments with a single entry and single exit node" [17]. The content of these

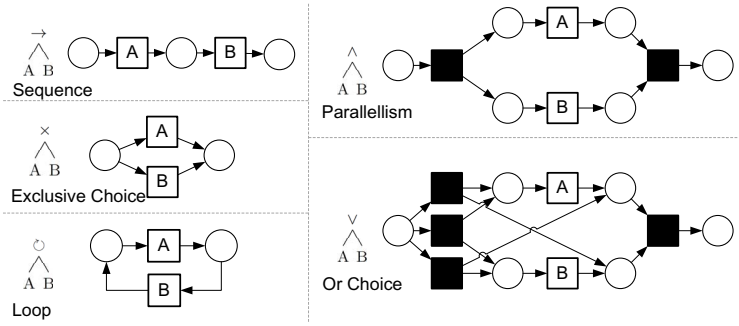


Fig. 2. Relation between process trees and block-structured Petri nets

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. However, each block in a process tree can only contain a predefined control flow construct, which are shown in Figure 2. Therefore, workflow graphs decomposed into an RPST can be more expressive than a process tree but an RPST is not necessarily sound while a process tree always is.

2.2 Quality of Process Trees

To measure the quality of a process tree, we consider one metric for each of the four quality dimensions. We based these metrics on existing work in each of the four areas and we adopted them for process trees [1–3, 6, 9, 16]. We do not present the precise formulae that compute the values of these metrics here, as they do not matter for the results in this paper.

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 [6] 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:

$$Q_{rf} = 1 - \frac{\text{cost for aligning model and event log}}{\text{Minimal cost to align arbitrary event log on model and vice versa}}$$

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.

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 finding that the size of a process model is the main factor for perceived complexity and introduction of errors in process models [15]. Furthermore, since we internally use binary trees, the number of leaves of the process tree has a direct influence on 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:

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

Precision compares the state space of the tree execution while replaying the log. Our metric is inspired by [1] 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, the precision is perfect. We obtain the part of the statespace 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 considers the frequency with which each node in the tree needs to be visited if the model is to produce the given 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 four metrics 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 metrics can efficiently be implemented using a genetic algorithm.

2.3 The ETM Algorithm

As discussed in Section 1 we propose the use of a genetic algorithm for the discovery of process models from event logs. Evolutionary algorithms have been applied to process mining discovery before in [4, 14]. Our approach follows the same high-level steps as most evolutionary algorithms [12], which are shown in Figure 3. The main improvements with respect to [4, 14] 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. 3. The input of the algorithm is an event log describing observed behavior. In the initial 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 by swapping subtrees between them (crossover) or by changing, adding or removing nodes (mutation). After these changes the fitness is again calculated. This is continued until at least one stop criterion is satisfied and the fittest candidate is then returned.

Our genetic algorithm has been implemented as a plug-in for the ProM framework [18]. 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 [8] 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 [8].

3 Running Example

Throughout 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

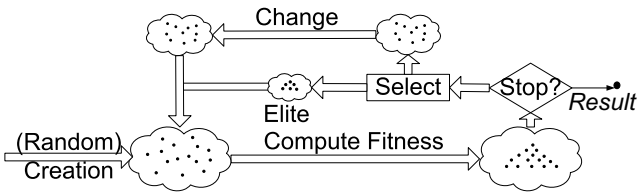


Fig. 3. The different phases of the genetic algorithm

Table 1. 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

Trace	#
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

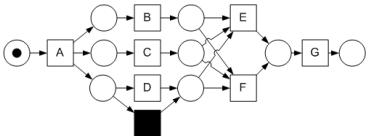


Fig. 4. Petri net of a loan application process. (`A` = send e-mail, `B` = check credit, `C` = calculate capacity, `D` = check system, `E` = accept, `F` = reject, `G` = send e-mail).

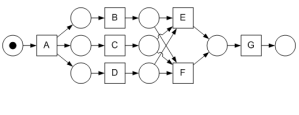
to the applicant to confirm the receipt of the request. Next, three activities are executed in parallel. Activity `B` is a check of the customer's credit history with a registration agency. Activity `C` is a computation of the customer's loan capacity and activity `D` is a check whether the customer 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.

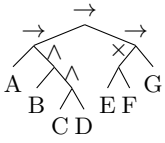
A Petri net of the loan application model is shown in Figure 4 and the log we obtained through simulation is shown in Table 1.

3.1 Results of Process Discovery Algorithms

In order to validate that our small 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. Figures 5 to 11 show the results of the various algorithms.

Figures 5 to 11 clearly indicate that, 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 4, 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*





f: 0,992	p: 0,995
s: 1,000	g: 0,889

Fig. 5. Result of the α algorithm [5] (sound)

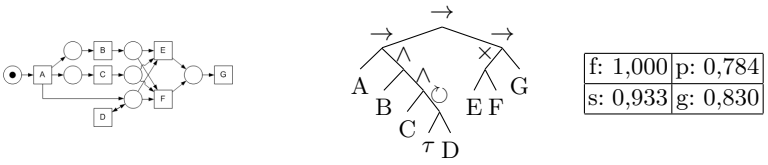


Fig. 6. Result of the ILP miner [21] (Ensuring empty net after completion, sound)

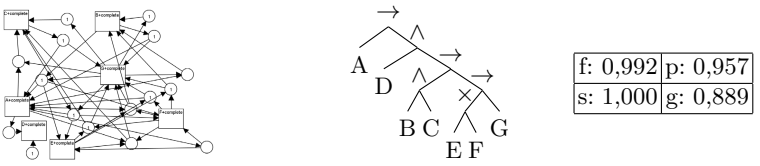


Fig. 7. Result of the language-based region theory [7] (The model is overly complex and incomprehensible, but sound)

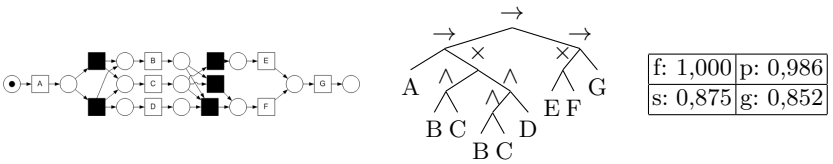


Fig. 8. Result of the heuristic miner [19] (Unsound, tokens are left behind.)

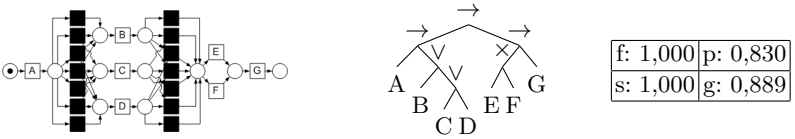


Fig. 9. Result of the Multi-phase miner [10] (Model is guaranteed "relaxed sound" and the tree reflects this.)

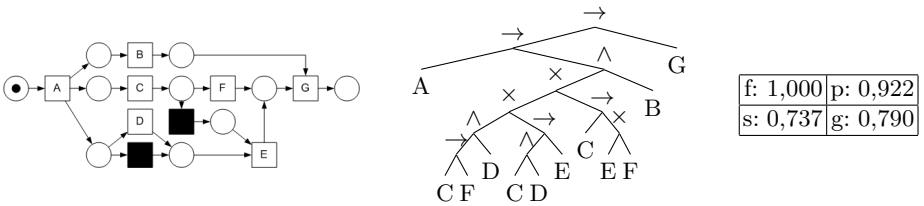


Fig. 10. Result of the genetic miner [14] (Unsound, tokens left behind.)

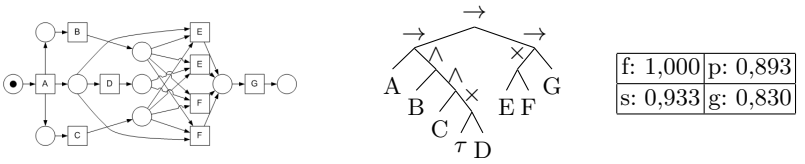


Fig. 11. Result of the state-based region theory [21]

under consideration. Next, in Section 6, we show that only our ETM algorithm is able to balance all quality dimensions for real life event logs.

4 Ignoring Quality Dimensions

The examples shown in figures 5 to 11 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.

4.1 Considering Only One Quality Dimension

Fig. 12a shows an example process tree that was discovered when focussing solely on the *replay fitness* dimension. Although the tree is able to replay the event log perfectly, the tree 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. 12a) 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).

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. 12b, which has a perfect precision because it can only generate the trace $\langle C, B, C \rangle$. A process tree will have perfect precision if each trace it can generate is used in an alignment. Since the tree of Fig. 12b can only generate one trace, each alignment between event log and the tree, will use this path of execution. However, the low replay fitness score indicates that the tree in Fig. 12b 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. 12c, where each activity is included exactly once. However, the tree 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.

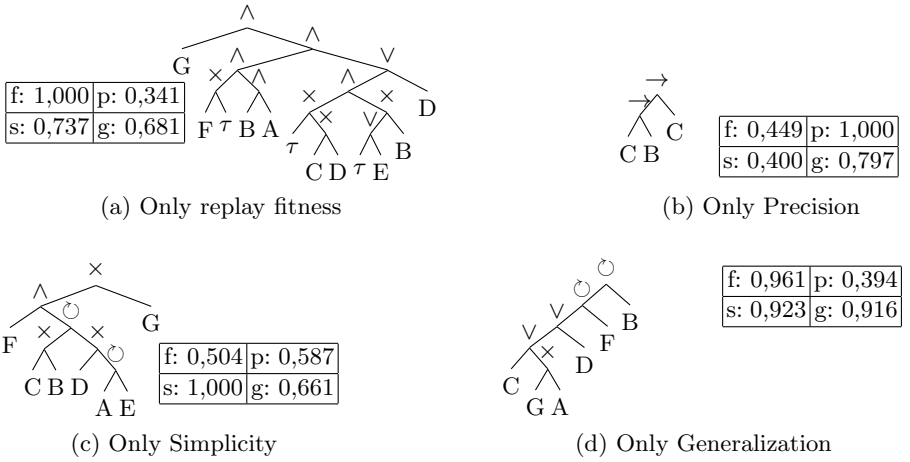
Generalization measures the likeliness that a model contains future, not yet observed behavior. When focussing solely on generalization, Fig. 12d shows the

process tree that has the best generalization score. 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 ↻ 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.

4.2 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 focussing 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 Figure 6, 7 and 11. 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 as shown in Fig. 13a also has has 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,



(a) Only replay fitness (b) Only Precision

(c) Only Simplicity (d) Only Generalization

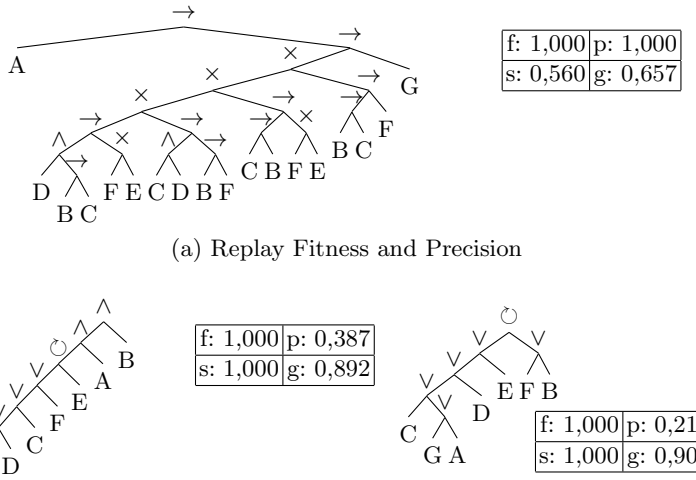
Fig. 12. Process trees discovered when considering each of the quality dimensions separately

since the tree is 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. 13b. When considering only replay fitness, we obtained fairly large models, while simplicity should keep the models compact. The process tree shown in Fig. 13b contains 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 $\wedge$, one $\circlearrowleft$ and three $\vee$ 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 that is found when focussing on the combination of *replay fitness and generalization* is shown in Fig. 13c. 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 $\circlearrowleft$ 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 4.1, 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 4.3, we consider three out of four dimensions, while never ignoring replay fitness.



(a) Replay Fitness and Precision

(b) Replay Fitness and Simplicity

(c) Replay Fitness and Generalization

Fig. 13. Process trees discovered when considering replay fitness and one of the other quality dimensions

4.3 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 first consider 3-out-of-4 dimensions.

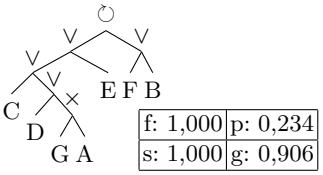
Fig. 14 shows the three process trees that are discovered when ignoring one of the quality dimensions, but always including replay fitness. Fig. 14a shows the process tree found when ignoring precision. The resulting process tree is similar to the one in Fig. 13c, which was based on replay fitness and generalization only. The only difference is that the parent of `A` and `G` has changed from $\vee$ to $\times$. Since this only influences precision, the other metrics have the same values and both trees have the same overall fitness when ignoring precision.

The process tree which is discovered when ignoring generalization is the same a when simplicity is ignored and is shown in Fig. 14b. 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 describe the process.

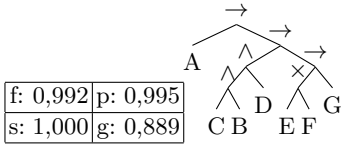
Interestingly, this tree is the same as the result of the α-algorithm (Fig. 5). However, as mentioned earlier, the α-algorithm is not very robust. This will also be demonstrated in Section 6 using real life event logs.

5 Weighing Dimensions

The process trees shown in Fig. 14 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. 15 shows the process tree 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.



(a) no precision



(b) no generalization or no simplicity

Fig. 14. Considering 3 of the 4 quality dimensions

Let us compare this process tree with the process tree of Fig. 14b, which is also the tree 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 Figure 15 performs better than the model we originally used for simulating the event log as can be seen in Figure 16. 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 less paths are 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.

6 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 3 event logs from real information systems. Using these event logs, and the running example, we show that our ETM 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 Table 1. 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.

f: 1,000	p: 0,923
s: 1,000	g: 0,889

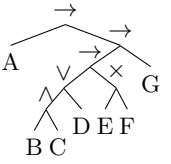


Fig. 15. Process tree discovered when replay fitness is 10 times more important than all other dimensions

f: 1,000	p: 0,893
s: 0,933	g: 0,830

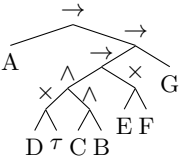


Fig. 16. Process tree of the model used for simulation (Translated manually from Figure 4)

Table 2. 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

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[1] 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 3.1 we selected four well-known algorithms: the α-algorithm [5], the ILP-miner [21], the heuristics miner [19] and the genetic algorithm by Alves de Medeiros [14]. Because we do not have enough space to show all process models we show some important characteristics of the resulting Petri nets in Table 2.

The α-algorithm and ILP Miner produce sound Petri nets for each of the 4 input logs. The Genetic Miner however never produces a sound Petri net and the Heuristics Miner produces a sound solution for 2 out of the 4 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 4 metrics, and the overall average fitness, is shown in Table 3.

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 and 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, which is shown in Figure 17a, 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

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

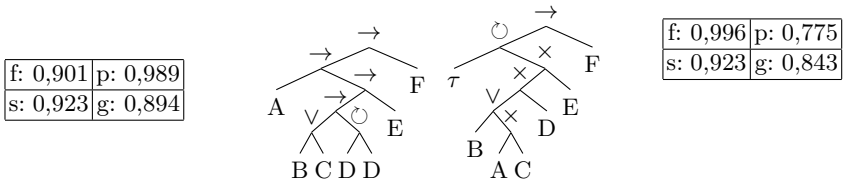
Table 3. Quality of Process Tree translations of Several Discovery Algorithms (*italic* results indicate unsound models, the best model is indicated in **bold**)

	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**	

quality dimensions. This results in the process tree as shown in Figure 17b. With an overall (unweighed) 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.

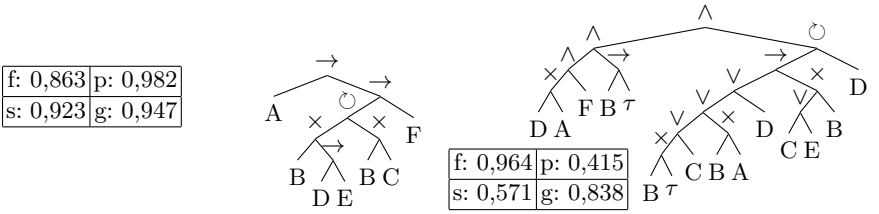
Event log L2 shows similar results: the α-algorithm and the ILP miner are able to find process models that can replay all behavior but allow for far more behavior. The heuristics miner and genetic miner again found models with several loops. The ETM algorithm was able to find a tree, which is shown in Figure 18a, 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 tree as is shown in Figure 18b. 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 tree found by the ETM algorithm. Although replay fitness is almost perfect, we can let the ETM algorithm discover a process tree with real perfect replay fitness. This requires making it 1000 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



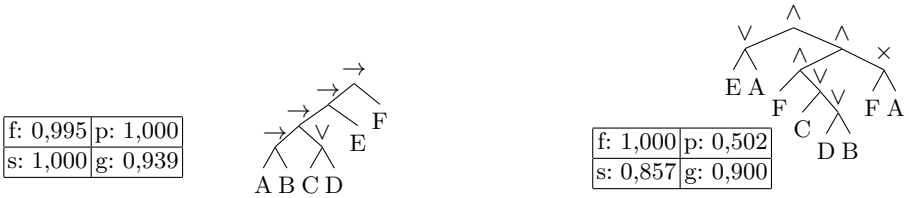
(a) All dimensions weight 1 (b) Replay Fitness weight 10, rest 1

Fig. 17. Process Trees discovered for L1



(a) All dimensions weight 1 (b) Replay fitness weight 10, rest weight 1

Fig. 18. Process Trees discovered for L2



(a) All dimensions weight 1

(b) Replay fitness weight 1000, rest weight 1

Fig. 19. Process Trees discovered for L3

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 Table 3 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 to the others. Furthermore, the ETM algorithm can be steered to improve certain dimensions of the process model as desired.

7 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 these dimensions.

In this paper, we presented the ETM algorithm to discover process trees on a log. This ETM algorithm *can be configured* to optimize for a weighted average over the four quality dimension, i.e. a model can be discovered that is optimal given the weights given to each parameter. Furthermore, the ETM algorithm is *guaranteed to produce sound process models.*

We used 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 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 demonstrated the need to consider all four quality dimensions. Moreover, our algorithm is able to balance all four dimensions is a seamless manner.

References

1. Carmona, J., van Dongen, B.F., van der Aalst, W.M.P., Adriansyah, A., Munoz-Gama, J.: Alignment Based Precision Checking. BPM Center Report BPM-12-10. BPMcenter.org (2012)
2. van der Aalst, W., Adriansyah, A., van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
3. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011)
4. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic Process Mining. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 48–69. Springer, Heidelberg (2005)
5. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)
6. Adriansyah, A., van Dongen, B., van der Aalst, W.M.P.: Conformance Checking using Cost-Based Fitness Analysis. In: IEEE International Enterprise Computing Conference (EDOC 2011), pp. 55–64. IEEE Computer Society (2011)
7. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process Mining Based on Regions of Languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 375–383. Springer, Heidelberg (2007)
8. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A Genetic Algorithm for Discovering Process Trees. In: Proceedings of the 2012 IEEE World Congress on Computational Intelligence. IEEE (to appear, 2012)
9. Calders, T., Günther, C.W., Pechenizkiy, M., Rozinat, A.: Using minimum description length for process mining. In: Proceedings of the 2009 ACM Symposium on Applied Computing, SAC 2009, pp. 1451–1455. ACM, New York (2009)

10. van Dongen, B.F.: Process Mining and Verification. Phd thesis, Eindhoven University of Technology (2007)
11. van Dongen, B.F., Alves de Medeiros, A.K., Wen, L.: Process Mining: Overview and Outlook of Petri Net Discovery Algorithms. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 225–242. Springer, Heidelberg (2009)
12. Eiben, A.E., Smith, J.E.: Introduction to evolutionary computing. Springer (2003)
13. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
14. Alves de Medeiros, A.K., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic Process Mining: An Experimental Evaluation. Data Mining and Knowledge Discovery 14(2), 245–304 (2007)
15. Mendling, J., Verbeek, H.M.W., van Dongen, B.F., van der Aalst, W.M.P., Neumann, G.: Detection and Prediction of Errors in EPCs of the SAP Reference Model. Data and Knowledge Engineering 64(1), 312–329 (2008)
16. Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems 33(1), 64–95 (2008)
17. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. Data and Knowledge Engineering 68(9), 793–818 (2009)
18. Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: XES, XESame, and ProM 6. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 60–75. Springer, Heidelberg (2011)
19. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering Workflow Models from Event-Based Data using Little Thumb. Integrated Computer-Aided Engineering 10(2), 151–162 (2003)
20. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining Process Models with Non-Free-Choice Constructs. Data Mining and Knowledge Discovery 15(2), 145–180 (2007)
21. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process Discovery using Integer Linear Programming. Fundamenta Informaticae 94, 387–412 (2010)