Genetic Process Mining: Alignment-based Process Model Mutation

M.L. van Eck, J.C.A.M. Buijs, and B.F. van Dongen

Eindhoven University of Technology, The Netherlands
{m.l.v.eck,j.c.a.m.buijs,b.f.v.dongen}@tue.nl

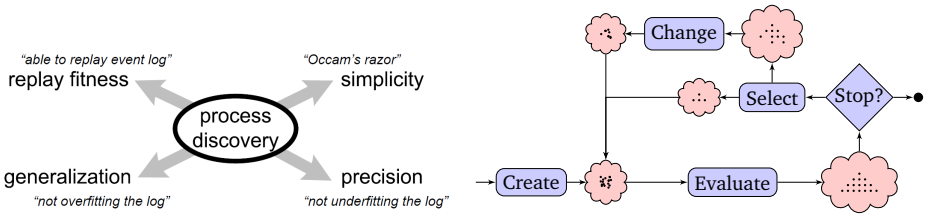
Abstract. The Evolutionary Tree Miner (ETM) is a genetic process discovery algorithm that enables the user to guide the discovery process based on preferences with respect to four process model quality dimensions: replay fitness, precision, generalization and simplicity.
Traditionally, the ETM algorithm uses random creation of process models for the initial population, as well as random mutation and crossover techniques for the evolution of generations. In this paper, we present an approach that improves the performance of the ETM algorithm by enabling it to make guided changes to process models, in order to obtain higher quality models in fewer generations. The two parts of this approach are: (1) creating an initial population of process models with a reasonable quality; (2) using information from the alignment between an event log and a process model to identify quality issues in a given part of a model, and resolving those issues using guided mutation operations.

1 Introduction

Over the years, more and more data is recorded and stored in information systems. Process mining aims to *discover, monitor and improve real processes by extracting knowledge from event logs* readily available in today's information systems [1]. Process *discovery* techniques can be used to automatically produce process models from such event logs and different techniques return different process models, with different properties and qualities.

There are four basic quality dimensions, shown in Fig. 1a, that are generally considered when discussing the quality of a process model [1,3]. The *replay fitness* dimension quantifies the extent to which the model can accurately replay the cases recorded in the log. The *precision* dimension measures whether the model prohibits behavior which is not seen in the event log. The *generalization* dimension assesses the extent to which the model will be able to reproduce possible future, yet unseen, behavior of the process. The complexity of the model is captured by the *simplicity* dimension, which operationalises Occam's Razor.

A general approach for genetic process discovery is shown in Fig. 1b. First, a *population* of models is created that are then evaluated according to the four quality dimensions described above. The models in the population are repeatedly changed using random mutation and crossover operations, and then re-evaluated until a stop criteria is satisfied. Finally, the best scoring model will be returned.



(a) The four quality dimensions used to calculate process model fitness (from [1]).

(b) The different phases of a genetic process discovery algorithm, as given in [3].

Fig. 1: The four quality dimensions and the genetic process discovery approach.

The quality of the models is measured as a weighted average over the four dimensions. Although all quality dimensions are of importance, obtaining a good quality in the dimension of replay fitness is crucial since it relates the observed behavior in the log to the model [3]. The replay fitness is computed using the alignment-based fitness computation defined in [2]. Alignments are explained in more detail in Subsec. 2.2, but basically they provide insights into where exactly the differences are between the behavior observed in the log and the model under consideration.

Unfortunately, genetic algorithms cannot discover models from large logs very efficiently [1]. This is because it is computationally complex to obtain the alignments needed to determine replay fitness [2, 6] and alignment calculation is already heavily optimized [2]. Moreover, in every generation of the algorithm, new models have to be aligned with the event log. Therefore, in this paper, our goal is to improve genetic process discovery by evaluating fewer models, i.e. we aim to discover higher quality models in fewer generations.

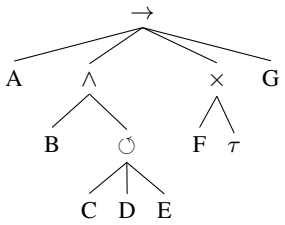
To reach this goal, we introduce two extensions that each address a particular part of the genetic process discovery lifecycle. The first extension is the guided creation of an initial population of models with a reasonable quality, such that few changes are needed to reach a model with a high quality. The second extension identifies quality issues in a given part of a model, which are then resolved using guided mutation operations instead of making completely random changes. In this paper we do not alter crossover.

We have chosen to use process trees, described in Subsec. 2.1, as the process modelling formalism for our approach. This choice was made because we want to support processes that contain duplicate activities and silent steps, and process trees are guaranteed to represent sound models. Also, we need a modelling formalism for which alignments are defined because they are used to determine the replay fitness of the models. Therefore, our approach has been developed in the context of the Evolutionary Tree Miner (ETM) [3], a genetic process discovery algorithm for process trees.

We explain our approach using a small event log from an example process shown in Fig. 2. The process starts with an A activity, followed by the parallel execution of B with a loop of C and D activities that is stopped after executing E. After this parallel execution, there is the choice to either execute or skip activity F and the process ends after activity G. The event log reflects this behavior, but it also contains traces that do not conform to the process.

ABCEG	×11	ABCEFG	×10
ACBEG	×7	ACBEFG	×10
ACEBG	×9	ACEBFG	×8
ACBDCEG	×5	ACDCEBG	×5
ACBCDCEFG	×1	ACDBCEG	×4
ACDBECDEG	×4	*ABCHIG*	×3
ACDBECDEFG	×2	*ACBCEG*	×2
ACDBCDCDEG	×3	ABCDCEFG	×3
ACDCBDCDCDEFG	×2	ACBDCEFG	×4
ACDCDCDBCDCDEG	×1	ACDCBEFG	×6

(a) The example event log. The red italic traces do not fit the modelled process completely.



(b) A process tree of the example process.

Fig. 2: The event log and a model of the running example process.

The structure of this paper is as follows: Sec. 2 contains an introduction on process trees and alignments. In Sec. 3 we explain how we create an initial population of good quality process trees. In Sec. 4 we show how to identify the parts of a process tree with a low quality and how to modify those parts. The experimental evaluation of our approach is presented in Sec. 5. Related work is discussed in Sec. 6 and we conclude this paper with Sec. 7.

2 Preliminaries

This section contains a short introduction to the process tree model notation and alignments which are used in the rest of the paper.

2.1 Process Trees

There exist many different process model notations and in this paper we use process trees to describe our process models. Process trees are used by the ETM algorithm internally because traditional modelling languages allow the creation of models that are not *sound* [3], i.e. these models contain deadlocks, livelocks and other anomalies. Process trees, however, are guaranteed to represent sound process models, which reduces the ETM algorithm's search space since unsound models do not have to be considered [3].

Fig. 3 shows the possible operators that process trees can be composed of, and their translations to BPMN constructs. *Operator* nodes specify the relation between their children. The five available operator types are: sequential execution ($\rightarrow$), parallel execution ($\wedge$), exclusive choice ($\times$), non-exclusive choice ($\vee$) and repeated execution ($\circlearrowleft$). Children of an operator node can again be operator nodes or they can be *leaf* nodes that represent the execution of an activity. The order of the children matters for the sequence and loop operators. The order of the children of a sequence operator specifies the order in which the children are executed (from left to right). Nodes can have an arbitrary number of children, except for loop nodes ($\circlearrowleft$) that always have three children. The left child is the 'do' part of the loop and after its execution either the middle child, the 'redo' part, may be executed or the right child, the 'exit' part, may be executed. After the execution of the 'redo' part the 'do part is again enabled and the 'exit' part is disabled. Process trees can also contain unobservable activities, indicated with a τ.

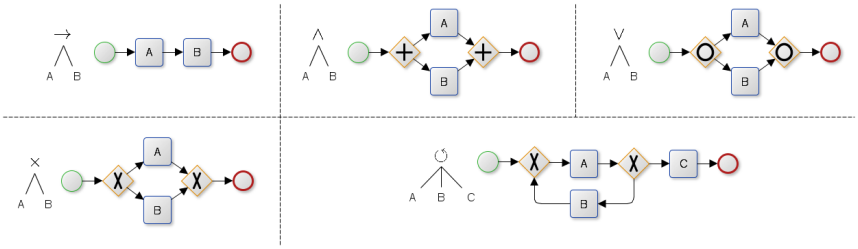
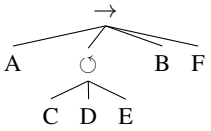


Fig. 3: The process tree operators and their relation to BPMN constructs.

Move	1	2	3	4	5	6	7	8	9
l	A	C	B	D	C	E	$\gg$	$\gg$	G
σ	A	C	$\gg$	D	C	E	B	F	$\gg$

(a) An alignment between the trace $l = \langle A, C, B, D, C, E, G \rangle$ and an execution of the process tree in Fig. 4b.



(b) A process tree discovered for the running example process.

Fig. 4: An alignment between a trace and a process tree.

2.2 Alignments

Conformance checking techniques can be used to identify exactly where process executions that are stored in an event log deviate from the behavior allowed by a process model [1]. The most robust and state-of-the-art conformance checking technique *aligns* an event log and a process model by relating events in the event log to model elements and vice versa [2]. It does this by constructing an alignment that contains a minimal amount of deviations between the event log and the process model.

Fig. 4a shows such an alignment, represented as a sequence of moves relating the trace $l = \langle A, C, B, D, C, E, G \rangle$ from our running example with an execution sequence σ of the model in Fig. 4b. Occurrences of events in l that are matched to activity executions in σ are called *synchronous moves*, e.g. move $\frac{|A|}{|A|}$. Events in l that cannot be executed at that point in σ are called *log moves*, e.g. activity B cannot be executed in the model during move $\frac{|B|}{|\gg|}$. Similarly, *model moves* are activities in σ for which there is no matching event in l at that point, e.g. moves $\frac{|\gg|}{|B|}$ and $\frac{|\gg|}{|F|}$. This information can be used to identify which activities should be skipped or inserted, and at what position in the model, to make it conform to the event log, as we show in Subsec. 4.1.

3 Guided Population Creation

The first step in any evolutionary algorithm is the generation of an initial population, or in our case, the generation of an initial set of process trees. Since process trees are

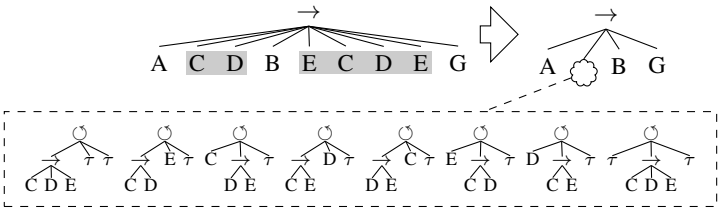


Fig. 5: Trace-model creation for the trace $\langle A, C, D, B, E, C, D, E, G\rangle$. The cloud in the right model is replaced with one of the possible ↺-subtrees.

structured, sound process models, the generation of a random collection of process trees given a set of activities is rather straightforward. However, the quality of such a population typically leaves much to be desired.

In this section we present an approach that is used to create an initial population of process trees that describe part of the event log. This improves the ETM algorithm's process discovery performance because fewer changes are needed from the initial population to reach high quality models.

Our approach is based on the idea that the main behavior of a process is often captured in the most frequently occurring traces in an event log. Therefore, process trees describing the main behavior of a process can be created by first creating simple process trees that describe a small number of randomly selected individual traces, and then merging these simple trees.

The resulting models will in general have a higher quality than randomly generated process models. Their replay fitness may not be very high, because they only explain a part of the event log, but they will fully describe the selected traces. This means that they score very high on precision and simplicity, and also high on the current generalization metric. This balance of the different quality dimensions is desirable because the guided mutation operations described in Sec. 4 focus mainly on improving replay fitness.

3.1 Trace-Model Creation

Creating a process tree that describes a single trace, i.e. a *trace-model*, is relatively straightforward. The root is a →-operator with all the activities from the trace as its children, arranged in the same order as they occur in the trace, as shown for the trace $\langle A, C, D, B, E, C, D, E, G\rangle$ in the left model in Fig. 5.

When there are duplicate activities in a trace we modify the trace-model slightly to remove the duplicate activities, as shown in Fig. 5, to make it easier to merge the trace-models in the next step. Duplicate activities are detected by counting the number of occurrences of each activity in a trace. If there are duplicate activities, such as activities C, D and E marked in gray, then the trace-model is parsed and a ↺-operator is inserted at the point where the first activity that occurs multiple times is encountered, which is between A and B in our example. The duplicate activities are removed from the trace-model and divided randomly over the 'do' and 'redo' children of the ↺-operator, while the order of the activities in these children is maintained, resulting in one of the 8 possible ↺-subtrees shown in the dashed rectangle.

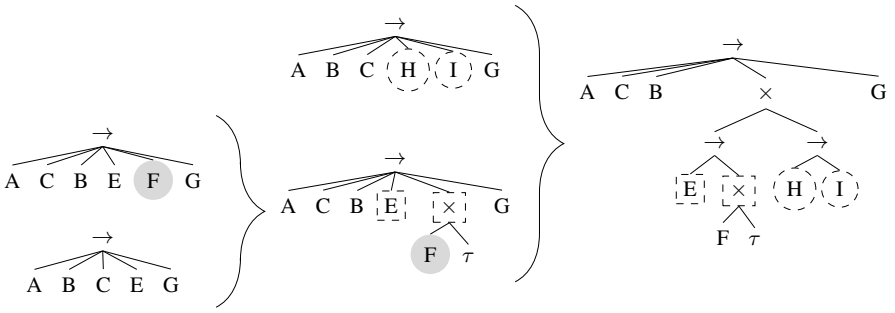


Fig. 6: Merging of trace models with nodes in the original and merged models marked for clarity.

Note that the duplicate elimination may result in a model that is unable to fully replay the original trace used to create the trace-model. Activities that are interleaved with duplicate activities, like activity B here, will be executed out of order. However, mutation operations can easily resolve this issue during evolution by putting B in parallel with the $\circlearrowleft$-operator. The random choice for the $\circlearrowleft$-subtree structure may also cause incorrect replay, but we want to introduce variation in the initial population here because the ETM's mutation operations have difficulties moving activities between the children of $\circlearrowleft$-operators.

3.2 Merging Trace-Models

The second step in the guided population creation is merging several trace-models into a process tree that describes multiple traces. This is done iteratively by merging two trace-models and then merging the result with the remaining trace-models. This merging process continues until all randomly selected traces and their trace-models have been merged or until duplicate activities are introduced. Duplicate activities make it difficult to create a mapping of activities to merge two process trees, so if duplicate activities are introduced then the resulting process model is used and the remaining trace-models are discarded.

To merge two process trees without duplicate activities we create a process tree that contains the common behavior from both process trees and a choice between the behavior that is different, which is shown in Fig. 6 using examples. To identify the common behavior and the differences, we create a mapping between the nodes of the process trees being merged. Such a mapping is easily created since each activity occurs at most once in each process tree. The order of activities that can be mapped is ignored because this can be improved later by the mutation operations introduced in Sec. 4. Therefore, the only important difference between the two leftmost models in Fig. 6 is activity F. The resulting merged model contains the mapped nodes from one process tree and a choice to skip the unmapped activity F. Similarly, merging the resulting model and the top right model introduces a choice between the unmapped activity E and $\times$-subtree from the first model, and the unmapped activities H and I from the second model.

Move	1	2	3	4	5	6	7	8	9
l	A	C	B	D	C	E	$\gg$	$\gg$	G
σ	A	C	$\gg$	D	C	E	B	F	$\gg$

The alignment from Fig. 4a

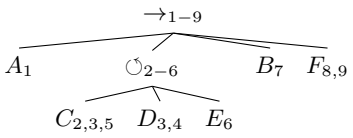


Fig. 7: A mapping of the moves of the alignment in Fig. 4a on the process tree from Fig. 4b. The subscripts show which moves are mapped onto each node.

4 Guided Mutation

The ETM algorithm aims to improve the quality of individual process trees in a population through the successive application of mutation and crossover operations that modify randomly selected nodes or subtrees, after which the quality of the resulting model is re-evaluated. We present an alternative approach to modify process trees, which reuses the replay fitness alignments to identify quality issues for a given node that are resolved using guided mutation operations.

In Subsec. 4.1, we introduce the *alignment move mapping* that shows the quality issues and what types of modifications may improve the quality. In Subsec. 4.2, 4.3 and 4.4 we introduce new operations that mutate process trees in a guided way using the mapping. Additional details can be found in [5].

Our approach focusses on replay fitness because it is generally considered to be the most important quality dimension [3, 6]. However, while changing process trees we aim to improve the replay fitness without reducing precision. We also take into account generalization and simplicity, by inserting structured process model parts and by relying on the ETM algorithm to balance all four dimensions.

4.1 Identifying Quality Issues

The alignments described in Subsec. 2.2 provide *local* information on the parts of the model that are not conform the event log. Model moves indicate activities that should be able to be skipped in the model, while log moves indicate activities that should be inserted in the model. In order to identify where activities should be inserted or made skippable in the tree, we create a mapping between alignment moves and nodes in the tree for each trace in the event log.

An alignment move mapping, as shown in Fig. 7, is created as follows. Synchronous and model moves in an alignment can be easily mapped onto nodes in the process tree because it is known which node was executed during such a move, e.g. move 1 $\left|\frac{\text{A}}{\text{A}}\right|$ is mapped onto node A and move 7 $\left|\frac{\gg}{\text{B}}\right|$ is mapped onto node B. However, no node in the process tree is executed during a log move, so therefore they are mapped onto the node related to the preceding and succeeding synchronous or model move. For example, move 3 $\left|\frac{\text{B}}{\gg}\right|$ is mapped onto activities C and D, while move 9 $\left|\frac{\text{G}}{\gg}\right|$ is mapped onto activity F. The argument for mapping log moves like this is that there are usually multiple suitable locations to insert activities to eliminate a log move, but including all possibilities makes modifying the process tree too complex.

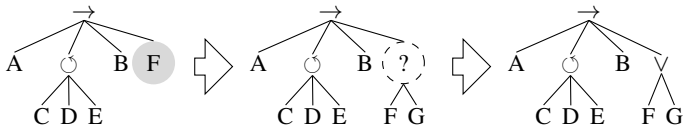


Fig. 8: Adding activity G next to node F in the process tree from Fig. 7.

All moves mapped to the children of a node are also mapped to the node itself. For example, the moves mapped to nodes C, D and E are also mapped to the ↻-operator and all moves are mapped to the root of the process tree. In this way, the alignment move mappings provide an indication of the behavior and the problems of each subtree within a process tree.

We use the information from the alignment move mapping to determine how to incrementally improve a process tree by mutating a given node. Behavior is removed if the given node is a leaf node onto which model moves, but no log moves, are mapped (Subsec. 4.2), behavior is added if the given node is a leaf node onto which log moves are mapped (Subsec. 4.3), and behavior is changed if the given node is an operator node (Subsec. 4.4).

4.2 Removing Behavior

Behavior can be removed from a process tree in two ways: by making activity nodes skippable or by removing them. Activity nodes are only removed from a process tree by the mutation operation if they are never executed as a synchronous move, otherwise they are made skippable.

Activity nodes are made skippable by replacing them with a choice between that node and a τ-node, as shown in Fig. 6 where node F is made skippable. Removing an activity node from a process tree is trivial, and an operator node without non-τ children can be removed as well. However, if a node's parent node is a ↻-operator then the node cannot be deleted because loop operators need to have three children, so the node is then replaced by a τ-node instead.

4.3 Adding Behavior

Behavior can be added to a process tree by inserting additional activity nodes into the tree, as shown in Fig. 8 where an activity is added to node F. First, an activity is randomly chosen from the set of log moves mapped onto the selected node. In this case, G is the only log move activity mapped to F. The old node is then replaced with an operator node joining both activities.

The choice for the operator node's type depends on the relation between the two activities in the event log. For each non-↻-operator type, we check what percentage of the trace alignments where at least one of the two activities is executed, can be correctly replayed by the operator. In our example log, $\vee$ can replay all traces, while $\times$ can replay 54%, and $\wedge$ and $\rightarrow$ can replay only 46% of the traces because activity F is not always executed while G is.

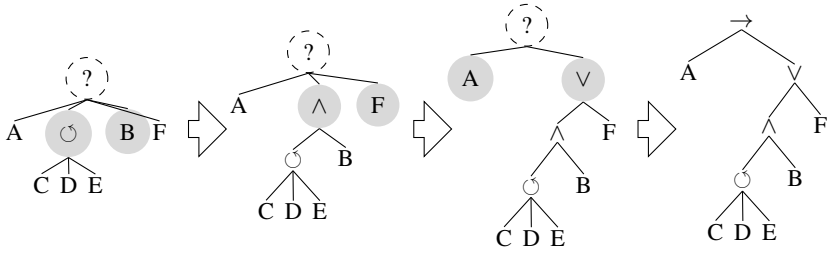


Fig. 9: Changing the behavior of the $\rightarrow$-operator in the model from Fig. 7.

We then choose the most restrictive operator that enables the correct replay in at least a certain percentage of all traces, in order to prevent unnecessary losses in precision. In Fig. 8 the $\vee$-operator was chosen, but if a replay percentage over 50% was good enough then the more precise $\times$-operator would have been chosen. Note that the $\times$-operator and $\wedge$-operator are more restrictive than the $\vee$-operator, and the $\rightarrow$-operator is more restrictive than the $\wedge$-operator.

An exception to the approach above occurs if the two activities that we want to join together are identical, in which case we have identified a self-loop and instead of the above approach, the leaf node in the process tree is simply replaced with a $\circlearrowleft$-operator that can repeatedly execute the activity.

4.4 Changing Behavior

The behavior of a process tree can be changed by adding and removing activities, but it can also be changed by modifying operator nodes. We use an iterative approach to change the type of an operator node, as shown in Fig. 9.

The first step in this approach is the random selection of two children of the operator node being changed, which are joined together with a new operator type chosen using the method discussed in Subsec. 4.3. In this example, the $\circlearrowleft$-operator and node B are first joined with an $\wedge$-operator because the activities below the $\circlearrowleft$ are interleaved in the traces with B. The next step is randomly taking one of the remaining children and joining this child to the subtree created in the previous step. This is repeated until all children of the original operator are added with an operator to the new subtree. Here, F is randomly selected in the second step and joined to the $\wedge$-subtree with an $\vee$-operator because F is often missing in traces. Finally, A is selected as the last remaining child and joined with a $\rightarrow$-operator because it is always the first activity.

5 Evaluation

The guided population creation and the guided mutations described in the previous sections have been evaluated using the running example log and a real-life event log [4]. The ETM algorithm and the extensions presented in this paper have been implemented in the ETM plug-in for the process mining framework ProM 6 [11]. Additional experimental evaluation can be found in [5].

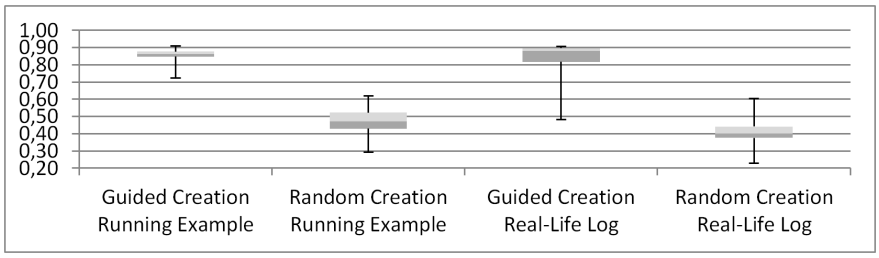


Fig. 10: Evaluation of the results of the initial population creation.

5.1 Guided Population Creation

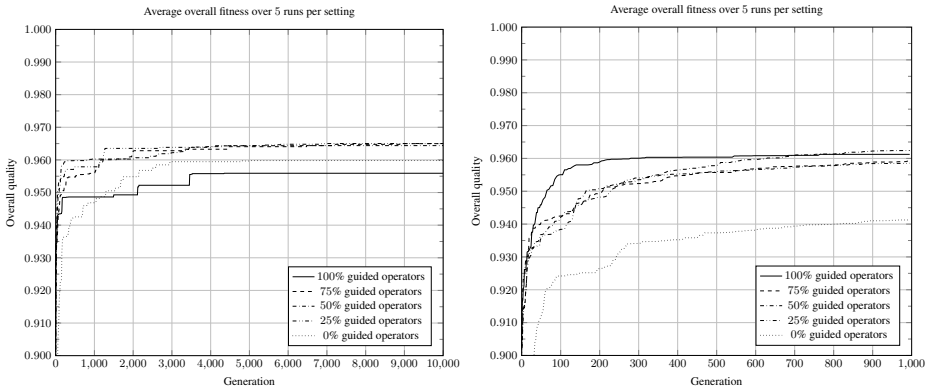
The results of the evaluation of the ETM's random initial population creation and the new approach from Sec. 3 are shown in Fig. 10. Both approaches were used to create 100 models that were evaluated in terms of an overall fitness score, calculated as a weighted average of the replay fitness $(\times 10)$, precision $(\times 5)$, simplicity $(\times 1)$ and generalization $(\times 0.1)$, which generally gives good results [3]. The resulting quality scores are presented as boxplots with the minimal and maximal values as the endpoints.

The results clearly show that guided population creation produces process trees with a higher quality than randomly generated models. For both logs, more than 75% of the guided creation models are significantly better than the best randomly created model. Guided population creation produces models that have high scores for precision, simplicity and generalization [5], so they are well suited to be improved using guided mutation, as it focusses on improving replay fitness.

5.2 Guided Mutation

The guided mutation operator evaluation consisted of multiple runs of 10,000 generations on the example log and 1,000 generations on the real-life event log. In each generation, the quality of the best scoring model was reported. The time required to mutate process trees is negligible compared to the time required to calculate their fitness. Therefore, we measure performance as the number of generations needed to reach a certain overall fitness score, defined as above. There was a 0, 25, 50, 75 or 100% probability to apply a guided mutation operator instead of a random mutation when modifying process trees, with five runs for each setting. These settings also determined the probability that guided tree creation was used when creating a model for the initial population.

The results of the evaluation on the artificial event log are shown in Fig. 11a. The guided mutation operators are able to quickly improve the high quality initial models to reach quality scores that the original ETM needs many more generations to reach. The 25%, 50% and 75% guided mutation runs also reached a quality score that the original ETM could not. An important observation is that the quality score of the runs with 100% guided mutation is eventually surpassed by the original ETM. The cause of this is likely the limited exploration of the solution search space when random mutations are



(a) Artificial event log

(b) Real-life event log

Fig. 11: The results of the guided mutation evaluation

not used to modify process trees. Random mutation allows for every possible process tree to be created eventually in an infinite run, but this is not true for guided mutation.

The ETM was also tested with 1,000 generations per run on the real-life event log and the results are shown in Fig. 11b. Again, the results show that the initial population has a higher quality when using guided population creation. The guided mutation operators also improve these models quickly to reach an overall fitness score that takes the original ETM algorithm much longer to reach.

6 Related Work

The guided mutation operators presented in this paper are similar to methods that aim to improve the quality of a model by repairing deviations. There exist approaches that repair deviations identified with respect to a known reference model [8], while other approaches use edit distance metrics to find improved models that are most similar to the model being repaired [7, 9, 10]. However, there are not many approaches that can automatically repair a model based on the deviations identified between a model and an event log [6].

One such repair approach by Fahland et al. [6] features an algorithm that tries to achieve a similar effect as the guided mutation operators introduced here. Their work focusses on repairing deviations between the behavior allowed by a process model and the behavior observed in an event log. These deviations are also detected by the creation of alignments between the process model and traces from the event log. Based on this information, they decompose the event log into several smaller logs containing the nonfitting subtraces. A subprocess is mined for each sublog using a process discovery algorithm and these are then added to the original model at the right location in order to repair the model. A limitation is that the approach cannot handle noise in the event log.

7 Conclusion

In this paper we presented an approach that enables the ETM to make guided changes to models, in order to obtain higher quality models in less time. The approach consists of two parts: (1) creating an initial population of process models with a reasonable quality; (2) identifying quality issues in a given part of a model, and resolving those issues using guided mutation operations.

The approach was evaluated using an artificial and a real-life event log, which showed that our approach can create higher quality models in fewer generations than the original ETM. However, the guided mutation operations are not perfect and random mutation is still required for optimal results.

In future work we plan to improve our approach as follows. The guided population creation can be improved with a more robust algorithm to merge process trees, while guided mutation is lacking when mutating $\circlearrowleft$-operators. Furthermore, our approach does not indicate which nodes should be repaired first, while research in this area may help to reduce the dependency on the randomness of the ETM algorithm. Finally, we focussed on improving replay fitness, but similar approaches may be created for the other quality dimensions as well.

References

1. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
2. A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Conformance checking using cost-based fitness analysis. In *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*, pages 55–64. IEEE, 2011.
3. 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 *On the Move to Meaningful Internet Systems: OTM 2012*, pages 305–322. Springer, 2012.
4. Joos C. A. M. Buijs. Receipt phase of an environmental permit application process ('WABO'), CoSeLoG project. `http://dx.doi.org/10.4121/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6`, 6 2014.
5. M.L. van Eck. Alignment-based process model repair and its application to the Evolutionary Tree Miner. Master's thesis, Technische Universiteit Eindhoven, 2013.
6. D. Fahland and W.M.P. van der Aalst. Model repair - aligning process models to reality. *Information Systems*, (0):–, 2013.
7. Mauro Gambini, Marcello La Rosa, Sara Migliorini, and Arthur H. M. ter Hofstede. Automated error correction of business process models. In *BPM*, LNCS, pages 148–165. Springer, 2011.
8. Jochen M Küster, Jana Koehler, and Ksenia Ryndina. Improving business process models with reference models in business-driven development. In *Business Process Management Workshops*, pages 35–44. Springer, 2006.
9. Chen Li, Manfred Reichert, and Andreas Wombacher. Discovering reference models by mining process variants using a heuristic approach. In *BPM*, pages 344–362, 2009.
10. Niels Lohmann. Correcting deadlocking service choreographies using a simulation-based graph edit distance. In *Business Process Management*, pages 132–147. Springer, 2008.
11. H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES, XESame, and ProM 6. In *Information Systems Evolution*, pages 60–75. Springer, 2011.