Improving Business Process Models using Observed Behavior

J.C.A.M. Buijs[1,2], M. La Rosa[2,3], H.A. Reijers[1], B.F. van Dongen[1], and W.M.P. van der Aalst[1]

[1] Eindhoven University of Technology, The Netherlands
{j.c.a.m.buijs,h.a.reijers,b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl
[2] Queensland University of Technology, Australia
m.larosa@qut.edu.au
[3] NICTA Queensland Research Lab, Australia

Abstract. Process-aware information systems (PAISs) can be configured using a *reference* process model, which is typically obtained via expert interviews. Over time, however, contextual factors and system requirements may cause the operational process to start deviating from this reference model. While a reference model should ideally be updated to remain aligned with such changes, this is a costly and often neglected activity. We present a new process mining technique that automatically improves the reference model on the basis of the observed behavior as recorded in the event logs of a PAIS. We discuss how to balance the four basic quality dimensions for process mining (fitness, precision, simplicity and generalization) and a new dimension, namely the structural similarity between the reference model and the discovered model. We demonstrate the applicability of this technique using a real-life scenario from a Dutch municipality.

1 Introduction

Within the area of *process mining* several algorithms are available to automatically discover process models. By only considering an organization's records of its operational processes, models can be derived that accurately describe the operational business processes. Organizations often use a *reference* process model, obtained via expert interviews, to initially configure a process. During execution however the operational process typically starts deviating from this reference model, for example, due to new regulations that have not been incorporated into the reference model yet, or simply because the reference model is not accurate enough.

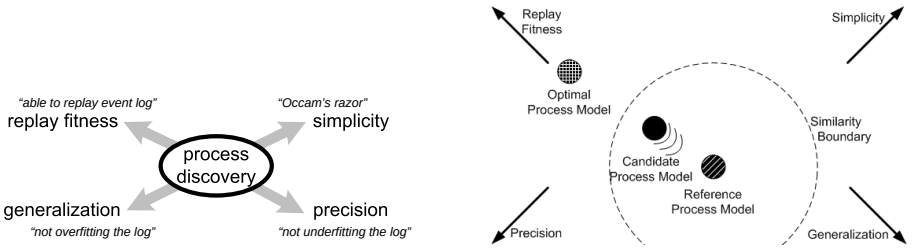
Process mining techniques can identify where reality deviates from the original reference model and especially how the latter can be adapted to better fit reality. Not updating the reference model to reflect new or changed behavior has several disadvantages. First of all, such a practice will overtime drastically diminish the reference model's value in providing a factual, recognizable view

on how work is accomplished within an organization. Second, a misaligned reference model cannot be used to provide operational support in the form of, e.g., predictions or recommendations during the execution of a business process.

A straightforward approach to fix the misalignment between a reference model and reality is to simply discover a new process model from scratch, using automated process discovery techniques from process mining [1]. The resulting model may reflect reality better but may also be very different from the initial reference model. Business analysts, process owners and other process stakeholders, may heavily rely on the initial reference model to understand how a particular process functions. Confronting them with an entirely new model may make it difficult for them to recognize its original, familiar ingredients and understand the changes in the actual situation. As a result, a freshly discovered process model may actually be useless in practice.

In this paper, we propose to use process mining to discover a process model that accurately describes an existing process yet is *very similar* to the initial reference process model. To explain our approach, it is useful to reflect on the four basic quality dimensions of the process model with respect to the observed behavior [1,2] (cf. Figure 1a). The *replay fitness* dimension quantifies the extent to which the discovered model can accurately replay the cases recorded in the log. The *precision* dimension measures whether the discovered model prohibits behavior which is not seen in the event log. The *generalization* dimension assesses the extent to which the resulting model will be able to reproduce possible future, yet unseen, behavior of the process. The complexity of the discovery process model is captured by the *simplicity* dimension, which operationalizes Occam's Razor.

Following up on the idea to use process mining for aligning reference process models to observed behaviors, we propose to add a fifth quality dimension to this spectrum: *similarity* to a given process model. By incorporating this dimension, we can present a discovered model that maximizes the four dimensions while remaining aligned, as far as possible, with the intuitions and familiar notions modeled in a reference model.



(a) Different quality dimensions for Process Model Discovery [1]

(b) Incorporating similarity

Fig. 1: Adding similarity as a process model quality dimension.

Figure 1b illustrates the effects of introducing this additional dimension. By setting a *similarity boundary*, the search for a model that balances the initial four quality dimensions is restrained. In this way, a new version of the reference model can be found that is similar to the initial reference model yet is improved with respect to its fit with actual behavior. Clearly, if the similarity boundary is being relaxed sufficiently (i.e. the discovered model is allowed to deviate strongly from the reference model), it is possible to discover the *optimal* process model. Such an optimal model, as explained, may not be desirable to use for process analysts and end users as a reference point, since they may find it difficult to recognize the original process set-up within it.

The remainder of the paper is structured as follows. In Section 2 we present related work in the area of process model improvement and process model repair. In Section 3 we present our approach using a genetic algorithm to balance the different quality dimensions while in Section 4 we show how to incorporate the similarity dimension in our approach. In Section 5 we show the results of applying our technique to a small example. In Section 6 the technique is applied to a real life case. Finally, Section 7 concludes the paper.

2 Related Work

Automatically improving or correcting process models using different sources of information is an active research area. Li et. al. [15] discuss how a reference process model can be discovered from a collection of process model variants. In their heuristic approach they consider the structural distance of the discovered reference model to the original reference model as well as the structural distance to the process variants. By balancing these two forces they make certain changes to the original reference model to make it more similar to the collection of process model variants. Compared to our approach, here the starting point is a collection of process variants, rather than a log.

An approach aimed to automatically correct errors in an *unsound* process model (a process model affected by behavioral anomalies) is presented by Gambini et. al. [11]. Their approach considers three dimensions: the structural distance, behavioral distance and 'badness' of a solution w.r.t. the unsound process model, whereby 'badness' indicates the ability of a solution to produce traces that lead to unsound behavior. The approach uses simulated annealing to simultaneously minimize all three dimensions. The edits applied to the process model are aimed to correct the model rather than to balance the five different forces.

Detecting deviations of a process model from the observed behavior has been researched, among others, by Adriansyah et. al. [2,4]. Given a process model and an event log, deviations are expressed in the form of *skipped* activities (activities that should be performed according to the model, but do not occur in the log) and *inserted* activities (activities that are not supposed to happen according to the model, but that occur in the log). A cost is attributed to these operations based on the particular activity being skipped/inserted. Based on this information an alignment can be computed between the process model and the log,

which indicates how well the process model can describe the recorded behavior. While this approach provides an effective measure for the replay fitness quality dimension of Figure 1a, the approach per se does not suggest any corrections to rectify the process model's behavior.

The work of Fahland et. al. [10] provides a first attempt at repairing process models based on observed behavior. In their notion, a process model needs repair if the observed behavior cannot be replayed by the process model. This is detected using the alignment between the process model and the observed behavior of [2, 4]. The detected deviations are then repaired by extending the process model with sub-processes nested in a loop block. These fixes are applied repeatedly until a process model is obtained that can perfectly replay the observed behavior. This approach extends the original process model's behavior by adding new fragments that enable the model to replay the observed behavior (no existing fragments are removed). The main disadvantage of this approach is that only one aspect of deviation, namely that of not being able to replay the observed behavior, is considered. Moreover, since repairs add transitions to the model, by definition, the model can only become more complex and less precise. It is unclear how to balance all five quality dimensions by extending the work in [10].

3 Our Mining Technique

In this section we briefly introduce our flexible evolutionary algorithm first presented in [6]. This algorithm can seamlessly balance four process model quality dimensions during process discovery.

3.1 Process Trees

Our approach internally uses a tree structure to represent process models. Because of this, we only consider sound process models. This drastically reduces the search space thus improving the performance of the algorithm. Moreover, we can apply standard tree change operations on the process trees to evolve them further, such as adding, removing and updating nodes.

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. An operator node specifies 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 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). For a loop, the left child is the 'do' part of the loop. After the execution of this part the right child, the 'redo' part, might be executed. After this execution the 'do' part is again enabled. The loop in Figure 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. Existing process models can be translated to the process tree notation, possibly by duplicating activities.

3.2 Quality Dimensions

To measure the quality of a process tree, we consider one metric for each of the four quality dimensions, as we proposed in [6]. We base these metrics on existing work in each of the four areas [2, 4] and we adapt them for process trees, as discussed below. For the formalization of these metrics on process trees we refer to [6].

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 total costs for the penalties are then normalized, using information on the maximum possible costs for this event log and process model combination, to obtain a value between 1 (perfect) and 0 (bad).

Precision compares the state space of the tree execution while replaying the log. Our metric is inspired by [5] 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 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 in the model according to the alignment.

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.

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

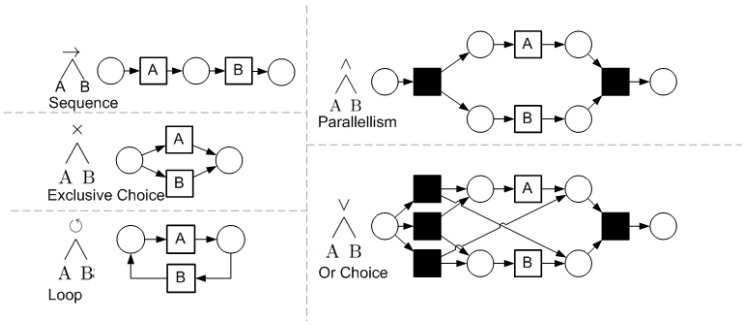


Fig. 2: Relation between process trees and block-structured Petri nets.

for perceived complexity and introduction of errors in process models [16]. 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, the tree in which each activity is represented exactly once is considered to be as simple as possible.

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 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.

3.3 The ETM Algorithm

In order to be able to seamlessly balance the different quality dimensions we implemented the ETM algorithm (which stands for *Evolutionary Tree Miner*). In general, this genetic algorithm follows the process shown in Figure 3. The input of the algorithm is an event log describing the observed behavior and, optionally, one or more reference process models. First, the different quality dimensions for each candidate currently in the population are calculated, and 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, or exceeding a time limit. 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 best candidate (highest overall fitness) is then returned.

The genetic algorithm has been implemented as a plug-in for the ProM framework [18]. We used this implementation for all experiments presented in this paper. The algorithm stops after $1,000$ generations or sooner if a candidate with perfect overall fitness is found before. In [7] we empirically showed that $1,000$ generations are typically enough to find the optimal solution, especially

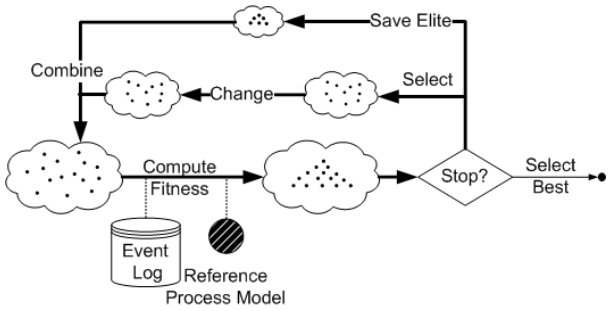


Fig. 3: The different phases of the genetic algorithm.

for processes with few activities. All other settings were selected according to the optimal values presented in [7].

4 Similarity as the 5th Dimension

In order to extend our ETM algorithm for process model improvement we need to add a metric to measure the similarity of the candidate process model to the reference process model. Similarity of business process models is an active area of research [3, 8, 9, 12–15, 19]. We distinguish two types of similarity: i) *behavioral similarity* and ii) *structural similarity*. Approaches focusing on behavioral similarity, e.g. [3, 8, 9, 13, 19], encode the behavior described in the two process models to compare using different relations. Examples are causal footprints [9], transition adjacency relations [19], or behavioral profiles [13]. By comparing two process models using such relations, it is possible to quantify behavioral similarity in different ways.

Approaches focusing on structural similarity only consider the graph structure of models and abstract from the actual behavior, e.g., heuristic approaches like [15], only focus on the number of common activities ignoring the connecting arcs, or vice versa, ignore the actual activities to only consider the arcs. Most approaches [8, 12, 14] provide a similarity metric based on the minimal number of edit operations required to transform one model into the other model, where an edit is either a node or an arc insertion/removal.

Both behavioral and structural similarity approaches first require a suitable *mapping* of nodes between the two models. This mapping can be best achieved by combining techniques for syntactic similarity (e.g. using string-edit distance) with techniques for linguistic similarity (e.g. using synonyms) [8].

Our algorithm only needs to consider the structural similarity, since the event log already captures the behavior that the process model should describe. Recall that the behavior of the reference model w.r.t. the logs is already measured by means of the four mining dimensions (Fig. 3.2). Hence, we use structural similarity to quantify the fifth dimension.

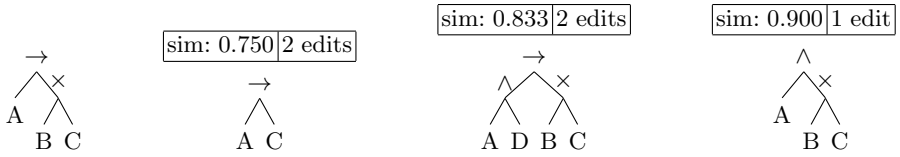
4.1 Tree Edit Distance as a Metric for Similarity

Since we use process trees as our internal representation, similarity between two process trees can be expressed by the tree edit distance for ordered trees. The tree edit distance dimension indicates how many simple edit operations (add, remove and change) need to be made to nodes in one tree in order to obtain the other tree. Since the other four quality metrics are normalized to values between 0 and 1, we need to do the same for the edit distance. This is easily done by making the number of edits relative to the sum of the size of both trees. The similarity score finally is calculated as 1 minus the edit distance ratio. Hence, a similarity score of 1.000 means that the process model is the same as the reference model.

Figure 4 shows examples of each of the three edit operations. The reference tree is shown in Figure 4a. Figure 4b shows the result after deleting activity `B` from the tree. Our trees are binary trees, meaning that each non-leaf node has exactly 2 children. Therefore, the $\times$ operator node is also removed. The removal of `B` from the tree results in an edit distance of 2. The similarity is $1 - \frac{2}{5+3} = 0.75$.

The process tree shown in Figure 4c has activity `D` added in parallel to activity `A`. This also results in 2 edits since a new $\wedge$ operator node needs to be added, including a leaf for activity `D`. Since the resulting tree has grown, the relative edit distance is less than when part of the tree is deleted. Finally, changing a node as shown in Figure 4d, where the root $\rightarrow$ operator is changed into an $\wedge$ operator, only requires 1 edit operation.

We use the Robust Tree Edit Distance (RTED) algorithm [17] to calculate the edit distance between two ordered trees. The RTED approach first computes the optimal strategy to use for calculating the edit distance. It then calculates the edit distance using that strategy. Since the overhead of determining the optimal strategy is minimal, this ensures best performance and memory consumption, especially for larger trees. However, it is important to realize that our approach is not limited to the RTED algorithm. Furthermore, although in this paper we only consider a single distance metric, it is possible to incorporate multiple metrics (for example looking at both structural and behavioral similarity).



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

Fig. 4: Examples of possible edits on a tree (a) and respective similarities.

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

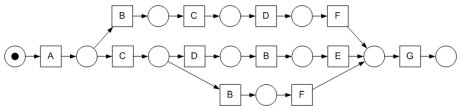


Fig. 5: Petri net of a loan application process. (`A` = send e-mail, `B` = check credit, `C` = calculate capacity, `D` = check system, `E` = accept, `F` = reject, `G` = send e-mail)

Table 2: Different weight combinations and the resulting fitness values for the simple example.

Weights					**Quality**					
Sim	**f**	**p**	**g**	**s**	**Sim**	**edits**	**f**	**p**	**g**	**s**
100	10	1	1	1	**1.000**	**0**	0.880	1.000	0.668	0.737
10	10	1	1	1	**0.935**	**3**	1.000	0.885	0.851	0.737
1	10	1	1	1	**0.667**	**12**	1.000	0.912	0.889	1.000
0.1	10	1	1	1	**0.639**	**13**	1.000	0.923	0.889	1.000
10	**0**	1	1	1	1.000	0	**0.880**	1.000	0.668	0.737
10	10	**0**	1	1	0.935	3	1.000	**0.849**	0.851	0.737
10	10	1	**0**	1	0.978	1	0.951	0.992	**0.632**	0.737
10	10	1	1	**0**	0.935	3	1.000	0.885	0.851	**0.737**

5 Experimental Evaluation

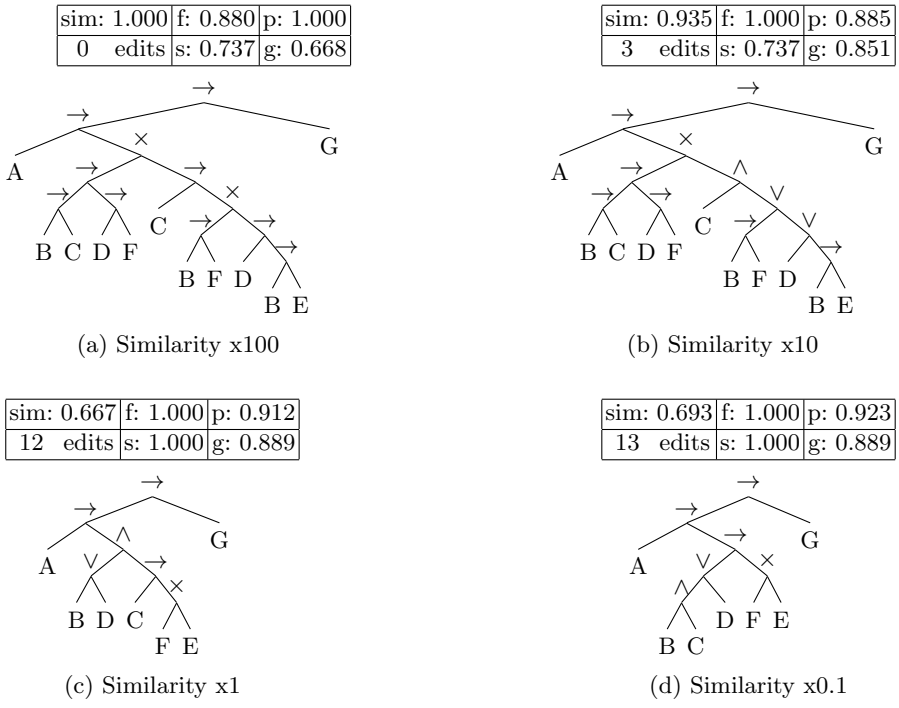
Throughout this section we use a small example to explain the application and use of our approach. Figure 5 describes a simple loan application process of a financial institute which provides small consumer credits through a webpage. The figure shows the process as it is known within the company. When a potential customer fills in a form and submits the request from the website, the process starts by executing activity `A` which notifies the customer with the receipt of the request. Next, according to the process model, there are two ways to proceed. The first option is to start with checking the credit (activity `B`) followed by calculating the capacity (activity `C`), checking the system (activity `D`) and rejecting the application by executing activity `F`. The other option is to start with calculating the capacity (activity `C`) after which another choice is possible. If the credit is checked (activity `B`) then finally the application is rejected (activity `F`). Another option is the only one resulting in executing `E`, concerned with accepting the application. Here activity `D` follows activity `C`, after which activity `B` is executed, and finally activity `E` follows. In all three cases the process ends with activity `G`, which notifies the customer of the decision made.

However, the observed behavior, as is recorded in the event log shown in Table 1, deviates from this process model. The event log contains 11 different traces whereas the original process model only allows for 3 traces, i.e., modeled and observed behavior differ markedly. To demonstrate the effects of incorporating the similarity between process trees, we run the extended ETM algorithm on the example data of Table 1.

In [6] we showed that, on this data set, the optimal weights are 10 for replay fitness and 1 for precision, generalization and simplicity. In the first experiment (Section 5.1), we only change the similarity weight to vary the amount of change we allow. In the second experiment (Section 5.2) we fix the weight for similarity and ignore each of the other four quality dimensions, one at a time. The experiment settings and their results are shown in Table 2.

5.1 Varying the similarity weight

Figure 6a shows the process tree that is discovered when giving the similarity a weight of 100. The similarity ratio is 1.000, indicating that no change has taken place. Apparently no change in the tree would improve the other dimensions enough to be beneficial.



(a) Similarity x100

(b) Similarity x10

(c) Similarity x1

(d) Similarity x0.1

Fig. 6: Varying similarity weight

If we reduce the similarity weight to 10, the process tree as shown in Figure 6b is discovered. Three edits have been applied: in the bottom-right part of the tree two $\rightarrow$ and an $\times$ operator have been changed to $\wedge$ and $\vee$. This allows for more behavior, as is indicated by the increase in replay fitness of 0.220. Also, generalization increased by 0.183, at the cost of a decrease in precision of 0.115.

If we lower the weight of the similarity to 1, we get the process tree as shown in Figure 6c. This process tree requires 12 edits starting from the original tree and is very different from the process tree we started with. However, compared to the previous process tree, the other 4 quality dimensions have improved overall. Replay fitness has now reached a value of 1.000 since this process tree allows skipping activity `D`. Also, simplicity reached 1.000 since no activities are duplicated or missing.

Table 3: Different weight combinations and the resulting fitness values for the practice application.

Weights					Quality					
Sim	**f**	**p**	**g**	**s**	**Sim**	**edits**	**f**	**p**	**g**	**s**
1000	10	1	1	1	**1.000**	**0**	0.744	0.785	0.528	0.755
100	10	1	1	1	**0.990**	**1**	0.858	0.799	0.566	0.792
10	10	1	1	1	**0.942**	**6**	0.960	0.770	0.685	0.815
1	10	1	1	1	**0.650**	**42**	0.974	0.933	0.747	0.613
0.1	10	1	1	1	**0.447**	**83**	0.977	0.862	0.721	0.519

Finally, reducing the similarity weight to 0.1 provides us with the process tree shown in Figure 6d, which is also the process tree that would be found when no initial process tree has been provided, i.e., pure discovery. The only improvement w.r.t. the previous tree is the slight increase in precision. However, the tree looks significantly different. The resemblance to the original tree is little as is indicated by a similarity of 0.693, caused by the 13 edits required to the original model.

5.2 Ignoring One Quality Dimension

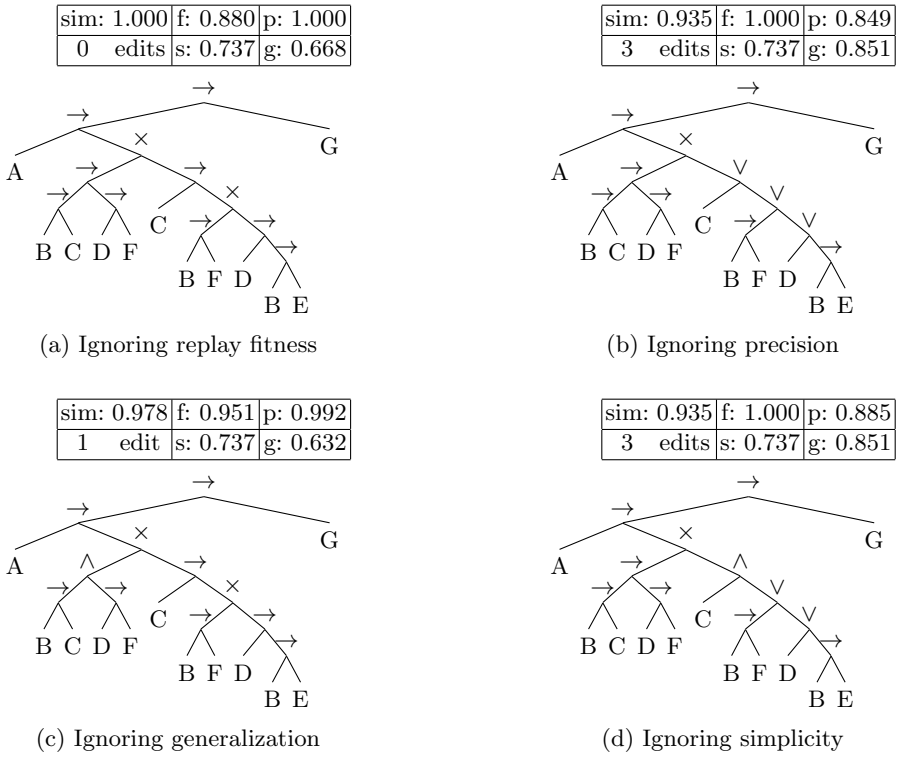
In [6] we showed that ignoring one of the four quality dimensions in general does not produce meaningful process models. However, many of these undesirable and extreme models are avoided by considering similarity. To demonstrate this we set the similarity weight to 10. The other weights are the same as in the previous experiment: 10 for fitness, 1 for the rest. We then ignore one dimension in each experiment. The results are shown in Figure 7.

Ignoring the fitness dimension results in the process tree as shown in Figure 7a. No changes were made, demonstrating that no improvement could be made on the other three dimensions that was worth the edit.

If precision is ignored, the result is the process tree as shown in Figure 7b. Replay fitness and generalization improved by applying 3 edits. The tree of Figure 6b, where we used the same similarity weight but included precision, only 1 edit was allowed. By removing the restriction on precision, it is worth to apply more edits to improve replay fitness and generalization.

We do not see this effect as strongly when we ignore generalization or simplicity. The resulting process trees, shown in Figure 7c and Figure 7d, are very similar to the original one with only 1 edit.

This experiment shows that considering similarity to a reference process model avoids the extreme cases encountered in [6].



(a) Ignoring replay fitness

(b) Ignoring precision

(c) Ignoring generalization

(d) Ignoring simplicity

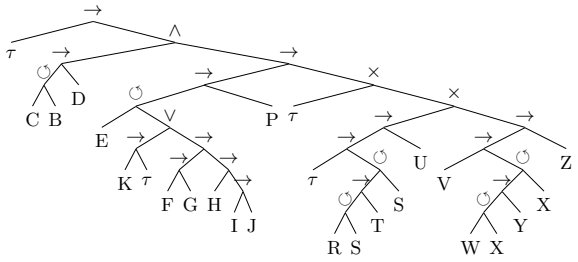
Fig. 7: Ignoring one dimension.

6 Application in Practice

Within the context of the CoSeLoG project, we are collaborating with ten Dutch municipalities that are facing the problem addressed in this paper.[1] The municipalities have implemented case management support, using a particular reference model. Now they are deriving new, possibly shared, reference models because they want to align each model with their own real process and the real processes in the municipalities they are collaborating with.

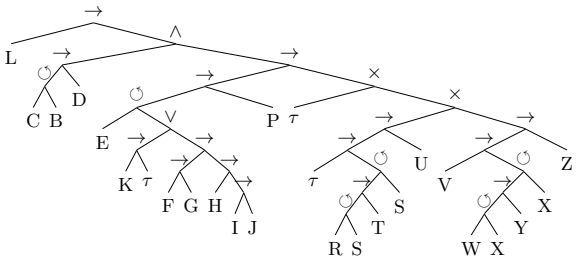
One of the municipalities participating in the CoSeLoG project recently started looking at one of their permit processes. The reference model used in the implementation was very detailed, with many checks that the employees in practice did not always do (usually with good reasons). Therefore, they were interested in a model that looks similar to the original reference model, but still shows most of the behavior actually observed. For this we applied our technique to discover different variants of the process model, focusing on different quality combinations, while maintaining the desired similarity to the reference model.

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



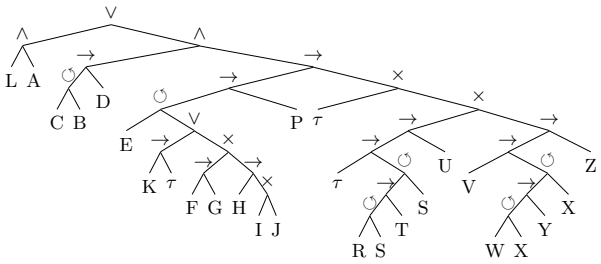
sim: 1.000	0 edits
f: 0.744	p: 0.785
s: 0.755	g: 0.528

(a) Similarity x1000



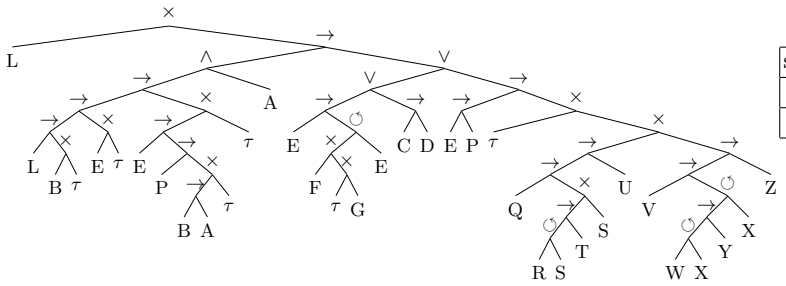
sim: 0.990	1 edit
f: 0.858	p: 0.799
s: 0.792	g: 0.566

(b) Similarity x100



sim: 0.942	6 edits
f: 0.960	p: 0.770
s: 0.815	g: 0.685

(c) Similarity x10



sim: 0.650	42 edits
f: 0.974	p: 0.933
s: 0.613	g: 0.747

(d) Similarity x1

Fig. 8: Process trees for the municipality process.

For this application we only used the first part of the process which contains a total of 27 different activity labels, which we anonymized using letters from `A` to `Z` plus `AA`.

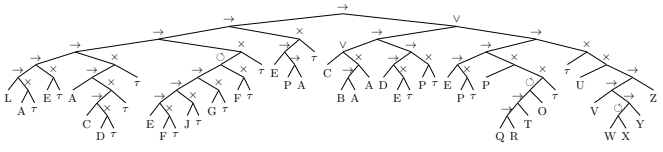
The experiment settings and their results are shown in Table 3. We experimented with fixed weights for the original four quality dimensions, and only changed the weight for the similarity to the reference model. The results confirm the intuition that reducing the weight of the similarity dimension allows more edits to be made (cf. the quality part of the table). In general, we also see that by allowing more edits, most of the quality dimensions improve. Of course, there is a trade-off between different dimensions. Since we weight the replay fitness dimensions 10 times more than the others, we see that this dimension always improves, sometimes at the cost of the other dimensions.

Figure 8a shows the process tree that is discovered using a similarity weight of $1,000$. This is the same process tree as created from the process model provided by the municipality. All four quality dimensions are relatively bad and many improvements are possible. However, none of these improvements were applied since one change needs to drastically improve the process tree to be worth it.

If we set the similarity weight to 100 we obtain the process tree of Figure 8b. Here one edit has been made, namely the left-most activity leaf node has been changed from τ to `L`. This single edit causes all four quality dimensions to improve, especially replay fitness. The original process model used a significantly different activity name than the one present in the event log, which was translated to a τ in the process tree.

If we further relax the importance of similarity by using a weight of 10, we obtain the process tree of Figure 8c. Here 6 edits have been made from the original model. The root node now changed to an $\vee$ to allow more behavior. The left branch of the root node also changed to allow more behavior, better suiting the recorded behavior. Also the operator node of activities `I` and `J` changed as well as the operator of their grandparent node. It appears that the event log contains a lot of short traces, only containing activities from the first part of the process. Some traces even contain activity `L` only. All the dimensions have improved after these changes, except precision which has slightly decreased.

Further relaxing the similarity we obtain the process trees of Figure 8d (weight of $1 - 42$ changes) and Figure 8e (weight of $0.1 - 83$ changes). Both these models have little to do with the original reference model. At the same time, the quality of these two process trees with respect to the log did not improve



sim: 0.447	83 edits
f: 0.977	p: 0.862
s: 0.519	g: 0.721

(e) Similarity x0.1

Fig. 8: Process trees for the municipality process (cont'ed).

much, while their appearance did. Therefore, for this experiment, we propose the process tree of Figure 8c as the improved version of the reference model. By applying only 6 edits the process model has improved significantly, mainly on replay fitness (from 0.744 to 0.960), while still showing a great resemblance to the original reference model.

7 Conclusion

In this paper, we proposed a novel process mining algorithm that improves a given *reference* process model using observed behavior, as extracted from the event logs of an information system. A distinguishing feature of the algorithm is that it takes into account the structural similarity between the discovered process model and the initial reference process model. The proposed algorithm is able to improve the model with respect to the four basic quality aspects (fitness, precision, generalization and simplicity) while remaining as similar as possible to the original reference model (5th dimension). The relative weights of all five dimensions can be configured by the user, thus guiding the discovery/modification procedure. We demonstrated the feasibility of the algorithm through various experiments and illustrated its practical use within the CoSeLoG project.

A limitation of this paper is that it assumes that the deviations discovered from the logs are always rightful. Indeed, some process deviations do reflect an evolving business process due to new acceptable practices or regulations, and as such should be accommodated into the reference model. However, some other deviations may be the result of non-compliance or be caused by a sub-efficient execution. These undesirable deviations should be isolated and discarded in order to prevent bad practices form becoming a part of the reference model. In future work, we plan to implement a more fine-grained control on the different costs for edit actions on different parts of the process model. For example, edits on operators may have lower costs than edits on labels. In this way we can for instance restrict our changes to extensions of the original reference model, and prevent existing parts of the model from being changed. Also, pre-defined domain-specific constraints which the process model should adhere to can be fixed in this way. However, while these techniques may help produce better results, the identified deviations still need to be validated by a domain expert before making their way into the reference model. Only in this way we can ensure that false positives are properly identified.

Finally, we plan to conduct an empirical evaluation of the understandability of the process models discovered using our algorithm, as perceived by domain experts, and compare the results with those obtained with other process mining algorithms, which ignore the similarity dimension.

Acknowledgments NICTA is funded by the Australian Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
2. W.M.P. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
3. W.M.P. van der Aalst, A. de Medeiros, and A. Weijters. Process Equivalence: Comparing Two Process Models Based on Observed Behavior. In *Proceedings of BPM*, LNCS. Springer, 2006.
4. A. Adriansyah, B. van Dongen, and W.M.P. van der Aalst. Conformance Checking using Cost-Based Fitness Analysis. In *Proceedings of EDOC*, pages 55–64. IEEE Computer Society, 2011.
5. A. Adriansyah, J. Munoz-Gama, J. Carmona, B.F. van Dongen, and W.M.P. van der Aalst. Alignment Based Precision Checking. In *Proceedings of the 8th BPI Workshop*, LNBIP. Springer, 2012. (to appear).
6. 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 *Proceedings of CoopIS*, LNCS. Springer, 2012.
7. Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. A Genetic Algorithm for Discovering Process Trees. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.
8. R.M. Dijkman, M. Dumas, B.F. van Dongen, R. Käärik, and J. Mendling. Similarity of Business Process Models: Metrics and Evaluation. *Information Systems*, 36(2):498 – 516, 2011.
9. B.F. van Dongen, R.M. Dijkman, and J. Mendling. Measuring Similarity between Business Process Models. In *Proceedings of CAiSE*, volume 5074 of *LNCS*, pages 450–464. Springer, 2008.
10. Dirk Fahland and W.M.P. van der Aalst. Repairing Process Models to Reflect Reality. In *Proceedings of BPM*, LNCS. Springer, 2012.
11. Mauro Gambini, Marcello La Rosa, Sara Migliorini, and Arthur H. M. ter Hofstede. Automated Error Correction of Business Process Models. In *Proceedings of BPM*, LNCS. Springer, 2011.
12. Tao Jin, Jianmin Wang, and Lijie Wen. Efficient retrieval of similar business process models based on structure. In *Proceedings of CoopIS*, LNCS. Springer, 2011.
13. M. Kunze, M. Weidlich, and M. Weske. Behavioral Similarity – A Proper Metric. In *Proceedings of BPM*, volume 6896 of *LNCS*, pages 166–181. Springer, 2011.
14. M. La Rosa, M. Dumas, R. Uba, and R. Dijkman. Business Process Model Merging: An Approach to Business Process Consolidation. *ACM Transactions on Software Engineering and Methodology*, 22(2), 2013.
15. C. Li, M. Reichert, and A. Wombacher. The minadept clustering approach for discovering reference process models out of process variants. *IJCIS*, 19(3-4):159–203, 2010.
16. 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.
17. Mateusz Pawlik and Nikolaus Augsten. RTED: A Robust Algorithm for the Tree Edit Distance. *CoRR*, abs/1201.0230, 2012.

18. 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 *Proceedings of CAiSE Forum*, volume 72 of *LNBIP*, pages 60–75, 2010.
19. H. Zha, J. Wang, L. Wen, C. Wang, and J. Sun. A Workflow Net Similarity Measure based on Transition Adjacency Relations. *Computers in Industry*, 61(5):463–471, 2010.