Mining Configurable Process Models from Collections of Event Logs

J.C.A.M. Buijs, B.F. van Dongen, and W.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. Existing process mining techniques are able to discover a *specific* process model for a given event log. In this paper, we aim to discover a *configurable* process model from a *collection* of event logs, i.e., the model should describe a *family of process variants* rather than one specific process. Consider for example the handling of building permits in different municipalities. Instead of discovering a process model per municipality, we want to discover one configurable process model showing commonalities and differences among the different variants. Although there are various techniques that merge individual process models into a configurable process model, there are no techniques that construct a configurable process model based on a collection of event logs. By extending our ETM genetic algorithm, we propose and compare four novel approaches to learn configurable process models from collections of event logs. We evaluate these four approaches using both a running example and a collection of real event logs.

1 Introduction

Different organizations or units within a larger organization may need to execute similar business processes. Municipalities for instance all provide similar services while being bound by government regulations [6]. Large car rental companies like Hertz, Avis and Sixt have offices in different cities and airports all over the globe. Often there are subtle (but sometimes also striking) differences between the processes handled by these offices, even though they belong to the same car rental company. To be able to share development efforts, analyze differences, and learn best practices across organizations, we need *configurable process models* that are able to describe families of process variants rather than one specific process [8, 16].

Given a collection of event logs that describe similar processes we can discover a process model using existing process mining techniques [1]. However, existing techniques are not tailored towards the discovery of a *configurable* process model based on a *collection* of event logs. In this paper, we compare four approaches to mine configurable models. The first two approaches use a combination of existing process discovery and process merging techniques. The third approach uses a two-phase approach where the fourth approach uses a new, integrated approach. All four approaches have been implemented in the ProM framework [18].

The remainder of the paper is organized as follows. In Section 2, we discuss related work on process discovery, configurable process models and current model merging techniques. In Section 3 we describe the four different approaches to mine configurable

process models in more detail. There we also describe how our genetic process discovery algorithm (ETM) has been extended to perform each of the four different approaches. Then we apply each of the approaches on a running example in Section 4. In Section 5 we apply the four approaches on a real-life event log collection to demonstrate the applicability of each of the approaches in practice. Section 6 concludes the paper and suggests directions for future work.

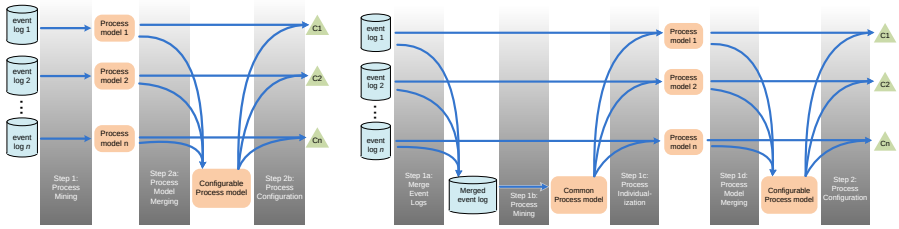
2 Related Work

The goal of *process discovery* in the area of process mining is to automatically discover process models that accurately describe processes by considering only an organization's records of its operational processes [1]. 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. For a complete overview we refer to [1]. However, until now, no process mining technique exists that is able to discover a single, configurable, process model that is able to describe the behavior of a *collection of event logs*.

A *configurable process model* describes a family of process models, i.e., variants of the same process. A configuration of a configurable process model restricts its behavior, for example by *hiding* or *blocking* activities. Hiding means that an activity can be skipped. Blocking means that a path cannot be taken anymore. Most formalisms allow operators to be made more restrictive (e.g., an OR-split is changed into an XOR-split). By configuring the configurable process model a (regular) process model is obtained. A configurable process model aims to show commonalities and differences among different variants. This facilitates reuse and comparison. Moreover, development efforts can be shared without enforcing a very particular process. Different notations and approaches for process configuration have been suggested in literature [4, 8, 11, 15–17]. In this paper we use a representation based on [17].

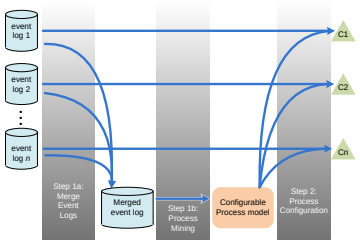
Configurable process models can be constructed in different ways. They can be designed from scratch, but if a collection of existing process models already exist, a configurable process model can be derived by merging the different variants. The original models used as input correspond to configurations of the configurable process model.

Different approaches exist to merge a collection of existing process models into a configurable process model. A collection of EPCs can be merged using the technique presented in [9]. The resulting configurable EPC may allow for additional behavior, not possible in the original EPCs. La Rosa et al. [12] describe an alternative approach that allows merging process models into a configurable process model, even if the input process models are in different formalisms. In such merging approaches, some configurations may correspond to an unsound process model. Li et al. [13] discuss an approach where an existing reference process model is improved by analyzing the different variants derived from it. However, the result is not a configurable process model but an improved reference process model, i.e., variants are obtained by modifying the reference model rather than by process configuration. The CoSeNet [17] approach has been designed for merging a collection of block structured process models. This approach always results in sound and reversible configurable process models.

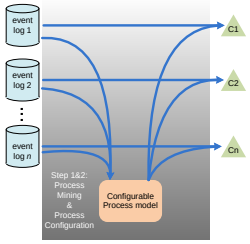


(a) Approach 1: Merge individually discovered process models

(b) Approach 2: Merge similar process models



(c) Approach 3: First discover a single process model then discover configurations



(d) Approach 4: Discover process model and configurations at the same time

Fig. 1: Creating a configurable process model from a collection of event logs.

Another way of obtaining a configurable process model is not by merging process models but by applying process mining techniques on a collection of event logs. This idea was first proposed in [10], where two different approaches were discussed, but these were not supported by concrete discovery algorithms. The first approach merges process models discovered for each event log using existing process model merge techniques. In the second approach the event logs are first merged and then a combined process model is discovered and individualized for each event log.

3 Mining a Configurable Process Model

In this section we present different approaches to mine a configurable process model from a collection of event logs. We also present the algorithm used for the discovery of process models.

3.1 Approaches

As mentioned in Section 1, we consider four approaches to discover a configurable process model from a collection of event logs.

The first approach, as is shown in Figure 1a, applies process discovery on each input event log to obtain the corresponding process model. Then these processes models are merged using model merge techniques. This approach was first proposed in [10].

Since the process models of the first approach are discovered independently of each other, they might differ significantly hence merging them correctly becomes more difficult. Therefore we propose a second approach as an improvement of the previous approach. The overall idea is shown in Figure 1b. From the input event logs first one process model is discovered that describes the behavior recorded in all event logs. Then the single process model is taken and individualized for each event log. For this we use the work presented in [7] to improve a process model within a certain edit distance. In the next step these individual process models are merged into a configurable process model using the approach of [17]. By making the individual process models more similar, merging them into a configurable process model should be easier.

The third approach, as shown in Figure 1c, is an extension of the second approach presented by Gottschalk et al. in [10]. A single process model is discovered that describes the behavior of all event logs. Then, using each individual event log, configurations are discovered for this single process model. In this approach the common process model should be less precise than other process models since we can only restrict the behavior using configurations, but not extend it. Therefore the process discovery algorithm applied needs to put less emphasis on precision.

The fourth approach is a new approach where the discovery of the process model and the configuration is combined, see Figure 1d. This approach is added to overcome the disadvantages of the other three approaches. By providing an integrated approach, where both the process model and the configuration options are discovered simultaneously, better trade-offs can be made.

The third and fourth approaches require an algorithm that is able to balance trade-offs in control flow, and optionally in configuration options. In previous work we presented the ETM-algorithm [5] that is able to seamlessly balance different quality dimensions. Therefore, in this paper the ETM-algorithm is extended such that it can discover a single process tree using a collection of event logs. Together with the process tree a configuration for each of the event logs is also discovered. In order to be able to compare the results of the different approaches, the ETM-algorithm is used as the process discovery algorithm in all four approaches.

3.2 The ETM Algorithm

In this section we briefly introduce our evolutionary algorithm first presented in [5]. The ETM (*Evolutionary Tree Miner*) algorithm is able to discover tree-like process models that are sound and block-structured. The fitness function used by this genetic algorithm can be used to seamlessly balance different quality dimensions. In the remainder of this section we only discuss the ETM-algorithm on a high-level together with the extensions made, to prevent repetition. All details of the ETM-algorithm can be found in [5].

Overall the ETM algorithm follows the genetic process shown in Figure 2. The input of the algorithm is one or more event logs describing the observed behavior and, optionally, one or more reference process models. First, different quality dimensions for each candidate currently in the population are calculated, and using the weight given to each quality 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 ETM-algorithm works on process trees, which are a tree-like representation of a process model. The leafs are activities and the other nodes represent one of several predefined control-flow constructs.

To measure the quality of a process tree, we consider one metric for each of the four main quality dimensions described in literature [1–3] (see Fig. 3). We have shown in [5] that the *replay fitness* dimension is the most important of the four in process discovery. The replay fitness dimension expresses how much of the observed behavior in the event log can be replayed in the process model. The *precision* dimension indicates how much additional behavior is possible in the process model but is not observed in the event log. The *simplicity* dimension assesses how simple the process model description of the behavior is. The *generalization* dimension is added to penalize "overfitting", i.e., the model should allow for unseen but very likely behaviors.

For the simplicity dimension we use a slightly different metric than in previous work. Simplicity is based on Occam's razor, i.e., the principle that says that when all other things are equal the simplest answer is to be preferred. Size is one of the simplest measures of complexity [14] since bigger process models are in general more complex to understand. Unfortunately, the ideal size of the process tree cannot directly be calculated, as control flow nodes can have multiple children. Furthermore, it might be beneficial for other quality dimensions, such as replay fitness or precision, to duplicate certain parts. Therefore, in the genetic algorithm, we use the fraction of the process tree that consists of 'useless' nodes as a simplicity metric since it does not influence the other quality dimensions. A node is useless if it can be removed without changing the behavior of the tree. Useless nodes are operators with only one child, τ leafs in a $\rightarrow$ or $\wedge$ construct, non-first τ's in an $\vee$ construct and $\circlearrowleft$'s consisting of one $\circlearrowleft$ as a child and two τ's as other children.

Each of the four metrics is 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 nodes are visited, the closer the value gets to 1.

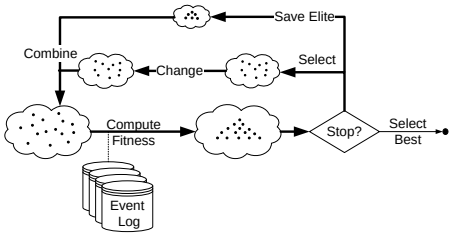


Fig. 2: The phases of the genetic algorithm.

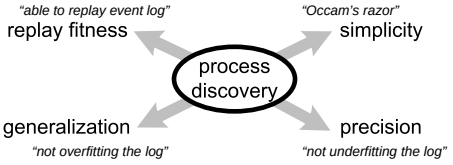


Fig. 3: Quality dimensions for Discovery [1, 2].

3.3 Configuring Process Trees

In this paper, we extend process trees [5] with configuration options. A node in a process tree can be not configured, blocked, hidden or 'downgraded' for each of the input event logs. The blocking and hiding operations are as specified in existing configuration languages [8, 16, 17] (see Section 2) and either *block* a path of execution or *hide* a part of the process. However, we add the configuration option that operators can be *downgraded*. By downgrading an operator, the behavior of the operator is restricted to a subset of the initially possible behavior. The $\circlearrowleft$ operator for instance can be downgraded to a $\rightarrow$. This is done by removing the 'redo' part of the $\circlearrowleft$ operator and putting the 'do' and 'exit' children of the loop in a sequence.

Another operator that can be downgraded is the $\vee$ operator which can be downgraded to an $\wedge$ (forcing all children to be executed), $\times$ (only allowing for one child to be executed), and a $\rightarrow$ (executing all its children in a particular order). However, since in one configuration the order of the children might be different than in another, we also added the $\leftarrow$ operator, representing a *reversed sequence*, which simply executes the children in the reversed order, i.e. from right to left. Finally, also the $\wedge$ can be downgraded to an $\rightarrow$ or $\leftarrow$ operator.

The *quality of the configuration perspective* should also be incorporated in the fitness function of the ETM-algorithm. This is partly done by applying the configuration options on the overall (i.e. configurable) process tree before evaluating the main four quality dimensions. For instance, when an activity that is not present in an event log is hidden from the process tree, this is reflected by replay fitness. The four quality dimensions are calculated for each individual event log and then a weighted average is calculated using the size of each event log. However, as part of the quality of the configuration, the number of nodes that have a configuration option set should be considered (otherwise all nodes can be made configurable without any penalty). Therefore, we add a new quality dimension for configuration that simply measures the fraction of nodes in the process tree for which no configuration option exist. The other four quality dimensions are more important than the configuration fitness, but if a configurable process tree exists with fewer configuration options and the same quality in the other dimensions, then the latter process tree is preferred.

4 Running Example

Our running example [7] is based on four variants of the same process describing a simple loan application process of a financial institute, providing small consumer credit through a webpage. The four BPMN process models describing the variants are shown in Figure 4. The event logs that were obtained through simulation are shown in 1.

In the *first variant* the process works as follows: when a potential customer fills in a form and submits the request on the website, the process is started by activity `A` which is sending an e-mail to the applicant to confirm the receipt of the request. Next, three activities are executed in parallel. Activity `B` is a check of the customer's credit history with a registration agency. Activity `C` is a computation of the customer's loan capacity and activity `D` is a check whether the customer is already in the system. This check is

Table 1: Four event logs for the four different variants of the loan application process of Figure 4.

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 C B F G	4
A C D B F G	2
A D B C F G	1
A D B C E G	1
A C B F G	1

(a) Event log for variant 1

Trace	#
A B1 B2 C D2 E G	20
A B1 B2 C D2 F G	50

(b) Event log for variant 2

Trace	#
A C B E	120
A C B F	80

(c) Event log for variant 3

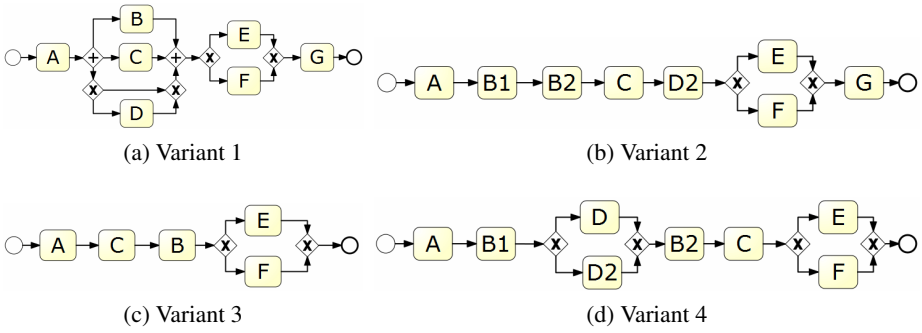
Trace	#
A B1 D B2 C E	45
A B1 D2 B2 C F	60

(d) Event log for variant 4

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, a decision is made and communicated to the client. The loan is accepted (activity `E`, covering about 20% of the cases) or rejected (activity `F`, covering about 80% of the cases). Finally, activity `G` (archiving the request) is performed.

The second loan application variant is simpler than the first process. Most notable is the absence of parallelism. Furthermore, activity `B` has been split into the activities `B1` (send credit history request to registration agency) and `B2` (process response of registration agency). Activity `D` of the original process has been replaced by `D2` which is checking the paper archive.

The third variant of the loan application process is even simpler where after sending the confirmation of receipt (activity `A`) the capacity is calculated (activity `C`) and the credit is checked (activity `B`). Then the decision is made to accept (activity `E`) or reject (activity `F`) the application. The application is not archived; hence no activity `G` is performed.



(a) Variant 1

(b) Variant 2

(c) Variant 3

(d) Variant 4

Fig. 4: Four variants of a loan application process. (`A` = send e-mail, `B` = check credit, `B1` = send check credit request, `B2` = process check credit request response, `C` = calculate capacity, `D` = check system, `D2` = check paper archive, `E` = accept, `F` = reject, `G` = send e-mail).

In the fourth and final variant of this process, after sending the confirmation of receipt (activity A), the request for the credit history is sent to the agency (activity B1). Then either the system archive (activity D) or paper archive (activity D2) is checked. Next the response of the credit history check is processed (activity B2) and next the capacity is calculated (activity C). Then the decision is made to accept (activity E) or reject (activity F) the application. The application is not archived (i.e., no activity G in model).

Although the four variations of the loan application process seem similar, automatically discovering a configurable process model is far from trivial.

4.1 Experimental Setup

In the remainder of this section we use the ETM algorithm as our discovery technique to construct a process model, in the form of a process tree, from an event log. We ran the experiments for $20,000$ generations on each individual event log for approaches 1 and 2. Because in approaches 3 and 4 we consider all four event logs at once, we increased the number of generations to $80,000$ to get a stable result. Each generation contained a population of 20 trees out of which the best six were kept unchanged between generations, i.e. the elite. The quality dimensions of replay fitness and simplicity were given a weight of ten, since we want a small process model with a good relation to the event log. A weight of five for precision makes sure the model does not allow for too much additional behavior and a weight of one-tenth for generalization makes the models more general.

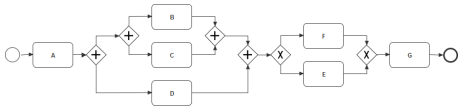
4.2 Approach 1: Merge Individually Discovered Process Models

The results of applying the first approach on the running example are shown in Figure 5. Each of the individual process models (see Figures 5a through 5d) clearly resemble each of the individual event logs. The combined configurable process model however is nothing more than a choice between each of the individual input process models. In this configurable process model those nodes that are configured have a grey 'callout' added, indicating for each configuration whether that node is not configured ('-'), hidden ('H') or blocked ('B'). The table shown in Figure 5f shows the different quality scores for both the configurable process models as well as for each of the configurations. Moreover, the simplicity statistics of size, number of configuration points (#C.P.) and similarity of the configured process model w.r.t. the configurable process model is shown. The fact that the four configuration options block a big part of the process model is reflected in the low similarity of the configured process models with the configurable process model. This is also shown by the relatively large size of the process tree.

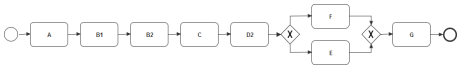
4.3 Approach 2: Merge Similar Discovered Process Models

In the second approach we try to increase similarity by discovering a common process model from all event logs combined, of which the result is shown in Figure 6a. This process model has difficulties to describe the combined behavior of the four variants.

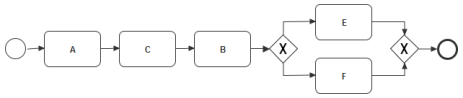
The four individual process models derived from this common process model are shown in Figures 6b through 6e. Each individual process model has a high similarity with the common process model, while some changes are made to improve the overall fitness for that particular event log. For the first three variants the discovered process models are identical to the one of approach 1. The process of the fourth variant however differs too much from the common model, hence the similar process model is not as good as the one found in approach 1. The combined process tree is shown in Figure 6f. Despite the similarity of the individual process models, the combined configurable process model is still a choice of the four input process models. The overall fitness of this model is slightly worse than that of approach 1, mainly due to the process model of variant 4. Similar to the previous approach, the number of configuration points is low. Unfortunately, also the similarity between the configured process model and the configurable process model is low.



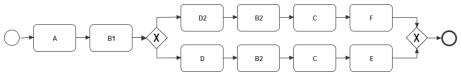
(a) Process model mined on event log 1



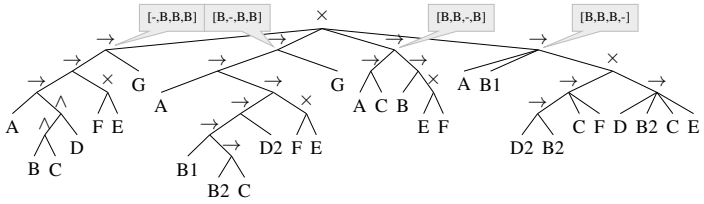
(b) Process model mined on event log 2



(c) Process model mined on event log 3



(d) Process model mined on event log 4



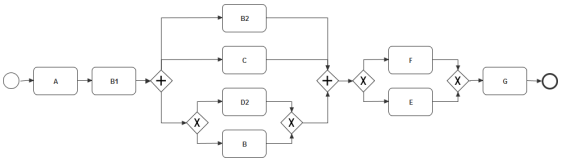
(e) Configurable process model obtained after merging models (a) through (d)

	Overall	Fitness	Precision	Simplicity	Generalization
Combined	0.989	0.999	0.999	0.981	0.220
Variant 0	0.986	0.995	0.995	0.981	0.235
Variant 1	0.989	1.000	1.000	0.981	0.263
Variant 2	0.989	1.000	1.000	0.981	0.174
Variant 3	0.989	1.000	1.000	0.981	0.264

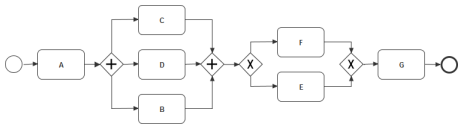
Size	#C.P.	Similarity
53	4	-
14	3	0.418
16	3	0.464
10	3	0.317
16	3	0.464

(f) Quality statistics of the configurable process model of (e)

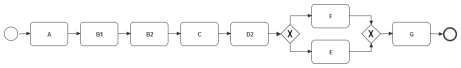
Fig. 5: Results of merging seperate discovered process models on the running example



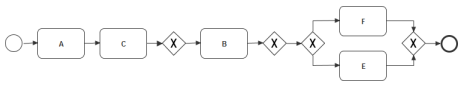
(a) Process model discovered from combined event log



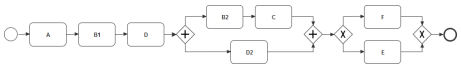
(b) Process model individualized for event log 1



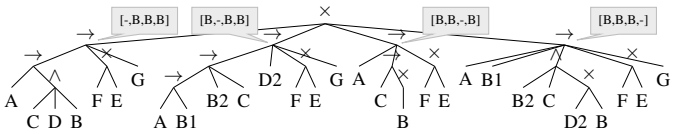
(c) Process model individualized for event log 2



(d) Process model individualized for event log 3



(e) Process model individualized for event log 4



(f) Configurable process model obtained after merging models (b) through (e)

	Overall	Fitness	Precision	Simplicity	Generalization
Combined	0.958	0.974	0.921	0.968	0.212
Variant 0	0.981	0.995	0.995	0.968	0.232
Variant 1	0.984	1.000	1.000	0.968	0.246
Variant 2	0.984	1.000	1.000	0.968	0.180
Variant 3	0.869	0.886	0.649	0.968	0.232

Size	#C.P.	Similarity
46	4	-
12	3	0.414
13	3	0.441
10	3	0.357
14	3	0.467

(g) Quality statistics of the configurable process model of (f)

Fig. 6: Results of merging the similar process models on the running example

4.4 Approach 3: First Discover a Single Process Model then Discover Configurations

The resulting configurable process model is shown in Figure 7. From this model it can be seen that we relaxed the precision weight, in order to discover an 'overly fitting' process model. Then, by applying configurations, the behavior is restricted in such a way that the model precisely describes each of the variants, as is indicated by the perfect replay fitness. This process model also scores relatively high for precision and simplicity. The process tree however has a similar large size as the two previous approaches. Nonetheless, the similarity of each of the individual process models to the configurable process model is higher than in the previous two approaches since only small parts are configured.

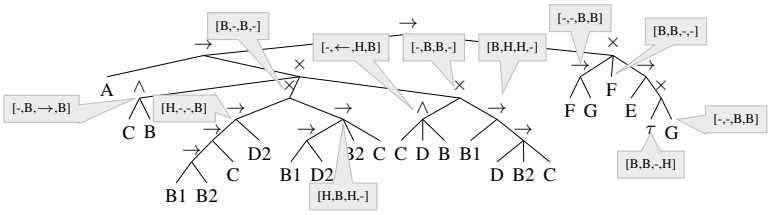
4.5 Approach 4: Discover Process Model and Configurations at the Same Time

The result of applying the fourth, integrated approach is shown in Figure 8. This process model is smaller and therefore simpler than previous models, a result of the weight of ten for the simplicity dimension. Moreover, it clearly includes the common parts of all variants only once, e.g. always start with `A` and end with a choice between `E` and `F`, sometimes followed by `G`. This process model correctly hides activities that do not occur in certain variants, for instance `G` for variants 3 and 4 and the `B`, `B1` and `B2` activities. Moreover, it correctly discovered the parallelism present in variant one, where the other variants are configured to be sequential. As a trade-off, it did not include activity `D2`, which is the least occurring activity in the event logs, and occurs in different locations in the process.

The discovered configurable process model can be further improved by increasing the replay fitness, making sure that all behavior can be replayed. This results in the configurable process model as shown in 9. This process model is able to replay all behavior, something that only was achieved in the two-phase mining approach. However, this results in a process model with a lot of ⟲ and ∨ constructs, which are then blocked for particular configurations. Moreover, the resulting process model is rather large, contains many configuration points and has mediocre similarity scores. This is a clear trade-off of aiming for a higher replay fitness value. However, the two-phase approach produced a better model with perfect replay fitness.

4.6 Comparison of the four Approaches

The results of applying the four different approaches on the running example are very different. All discovered models have similar scores for replay fitness and precision and there are (almost) no useless nodes. However, there are noticeable differences in generalization, size and similarity between the configurable and the configured models. The



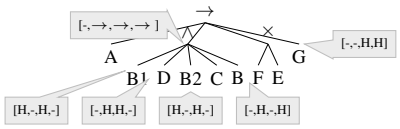
(a) Configurable process model discovered using the two-phase approach

	Overall	Fitness	Precision	Simplicity	Generalization
Combined	0.988	1.000	0.981	0.986	0.374
Variant 0	0.990	1.000	0.990	0.986	0.400
Variant 1	0.992	1.000	1.000	0.986	0.408
Variant 2	0.992	1.000	1.000	0.986	0.285
Variant 3	0.977	1.000	0.922	0.986	0.496

Size	#C.P.	Similarity
42	11	-
20	6	0.645
20	7	0.645
13	8	0.473
24	6	0.727

(b) Quality statistics of the configurable process model of (a)

Fig. 7: Results of the two-phase mining approach on the running example



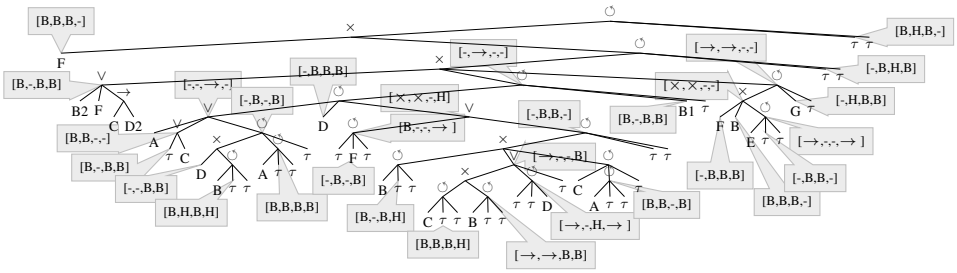
(a) Configurable process model discovered using the integrated discovery approach

	Overall	Fitness	Precision	Simplicity	Generalization
Combined	0.983	0.962	0.999	1.000	0.684
Variant 0	0.996	0.995	0.994	1.000	0.738
Variant 1	0.957	0.894	1.000	1.000	0.723
Variant 2	0.998	1.000	1.000	1.000	0.614
Variant 3	0.961	0.905	1.000	1.000	0.741

Size	#C.P.	Similarity
12	6	-
10	2	0.909
10	3	0.909
8	5	0.800
10	3	0.909

(b) Quality statistics of the configurable process model of (a)

Fig. 8: Results of the integrated mining approach on the running example



(a) Configurable process model discovered with more weight on replay fitness

	Overall	Fitness	Precision	Simplicity	Generalization
Combined	0.980	1.000	0.767	0.993	0.345
Variant 0	0.985	1.000	0.826	0.993	0.534
Variant 1	0.974	1.000	0.696	0.993	0.306
Variant 2	0.986	1.000	0.850	0.993	0.349
Variant 3	0.973	1.000	0.680	0.993	0.261

Size	#C.P.	Similarity
82	31	-
54	20	0.794
30	21	0.527
31	20	0.549
24	21	0.453

(b) Quality statistics of the configurable process model of (a)

Fig. 9: Result of the integrated mining approach when improving replay fitness

first two approaches score relatively poor on generalization, because the merge operator used introduces specific submodels for each log, which limits the number of visits per node during replay. Also, due to duplication, the models are significantly larger, and because in the configuration large parts are blocked, the configured models are dissimilar to the configurable one. Mining a process model and then mining configurations improves the similarity, but still the configurable model remains larger than necessary. Furthermore, the number of configuration points is very high. However, it is easier to aim for higher replay fitness values The final approach, where the configurations are discovered simultaneously with the configurable model reduces the size significantly,

Table 2: Case study event log statistics

	#traces	#events	#activities
Combined	1214	2142	28
L1	54	131	15
L2	302	586	13
L3	37	73	9
L4	340	507	9
L5	481	845	23

Table 3: Statistics of merging the separate process models on the case study event logs

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.979	0.973	0.962	0.997	0.560	1555	5	-
Variant 0	0.977	0.977	0.949	0.996	0.362	581	4	0.544
Variant 1	0.973	0.967	0.944	0.998	0.617	201	4	0.229
Variant 2	0.991	1.000	0.993	0.988	0.234	72	4	0.089
Variant 3	0.984	0.978	0.974	0.998	0.690	455	4	0.453
Variant 4	0.978	0.971	0.964	0.997	0.480	250	4	0.277

thus improving the similarity score. However, this comes at a minor cost of replay fitness.

The first two approaches seem to struggle with merging process models based on their behavior. Because they only focus on the structure of the model, the frequencies of parts of the process model being visited are not considered during the merge. The third and fourth approach both directly consider the behavior and frequencies as recorded in the event log. This seems to be beneficial for building a configurable process model since these latter two approaches outperform the first two. In the next section we apply all four approaches on a collection of real-life event logs to validate these findings.

5 Case Study

To validate our findings we use a collection of five event logs from the CoSeLoG project[1], each describing a different process variant. The main statistics of the event logs are shown in Table 2. The event logs were extracted from the IT systems of five different municipalities. The process considered deals with objections related to building permits.

The result of both the first (individually discovered process models that are then merged) and the second approach (making sure the models are similar) result in process trees with more than 200 or even $1,500$ nodes. Both process models however again consist of an $\times$ operator as the root with each of the five original models as its children that are then blocked, similar to the running example results. We therefore only show the statistics in Table 3 and 4 since the process models are unreadable.

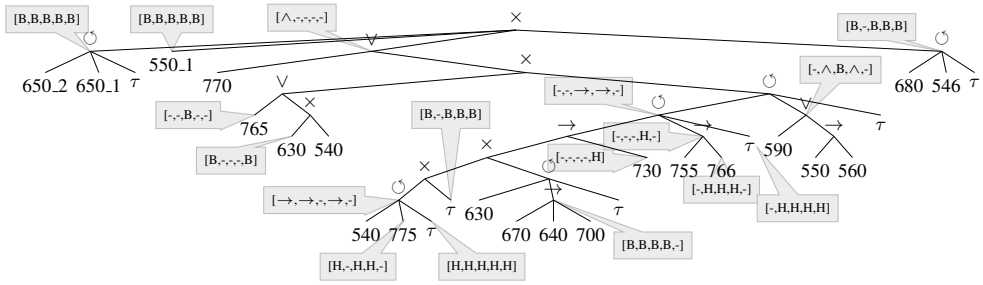
The third approach, where the ETM-algorithm first discovers a common process model that is not very precise, and then applies configuration options, results in the process tree as shown in Fig. 10a. The statistics for this process model are shown in

[1] More information can be found at `http://www.win.tue.nl/coselog/wiki/start`

Table 4: Statistics of merging the similar process models on the case study event logs

	Overall	Fitness	Precision	Simplicity	Generalization
Combined	0.980	0.989	0.936	0.998	0.540
Variant 0	0.973	0.992	0.894	0.999	0.337
Variant 1	0.977	0.969	0.958	0.998	0.598
Variant 2	0.966	0.991	0.863	0.998	0.533
Variant 3	0.991	0.999	0.968	0.999	0.461
Variant 4	0.977	0.994	0.909	0.998	0.582

Size	#C.P.	Similarity
236	5	-
65	4	0.432
34	4	0.252
24	4	0.185
48	4	0.338
69	4	0.452



(a) Results of the two-phase mining approach

	Overall	Fitness	Precision	Simplicity	Generalization
Combined	0.973	0.965	0.951	0.999	0.451
Variant 0	0.947	0.943	0.862	0.999	0.319
Variant 1	0.979	0.968	0.971	0.999	0.452
Variant 2	0.961	0.932	0.958	0.999	0.267
Variant 3	0.980	0.990	0.934	0.999	0.390
Variant 4	0.970	0.950	0.961	0.999	0.522

Size	#C.P.	Similarity
46	17	-
31	10	0.792
36	8	0.878
25	12	0.690
30	13	0.763
34	8	0.850

(b) Statistics of the two-phase mining result

Fig. 10: Results of the two-phase mining approach on the real-life event logs

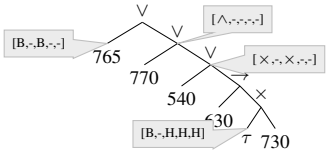
Table 10b. This process tree is rather compact and has reasonable scores for replay fitness, precision and simplicity.

The fourth approach, where the control flow and configuration points are discovered simultaneously, results in the process tree as shown in Fig. 11a. The statistics are shown in Table 11b. With only 4 configuration points, and similar quality scores as the previous result, this process tree is even smaller and hence simpler.

The application of the different approaches on the real-life event logs show similar results as on the running example. The first two approaches seem to have difficulties in merging the process models based on the behavior of the process model.

6 Conclusion

In this paper we presented and compared four approaches to construct a *configurable* process model from a *collection* of event logs. We applied all four approaches on both a running example and a real-life collection of event logs. Our results show that the naive approach of first discovering a process model for each event log separately and



(a) Result of the integrated mining approach

	Overall	Fitness	Precision	Simplicity	Generalization
Combined	0.966	0.952	0.929	1.000	0.839
Variant 0	0.945	0.866	1.000	1.000	0.622
Variant 1	0.970	0.958	0.935	1.000	0.861
Variant 2	0.955	0.920	0.942	1.000	0.651
Variant 3	0.974	0.975	0.923	1.000	0.845
Variant 4	0.962	0.945	0.921	1.000	0.860

Size	#C.P.	Similarity
11	4	-
9	4	0.900
11	0	1.000
10	3	0.952
11	1	1.000
11	1	1.000

(b) Statistics result

Fig. 11: Results of the integrated mining approach on the real-life event logs

then merging the discovered models yields large configurable models to which the individual configurations are not very similar. It is slightly better to first discover a process model on the combination of the event logs and then configure this model for each log. However, both of these approaches struggle with merging the modeled behavior of the input process models into a configurable process model. The other two approaches that directly discover a configurable process model from the event log seem to be able to use the recorded behavior to better generalize the behavior into a configurable process model. The approach where both the control flow and the configuration options are changed together seems to have more flexibility than the approach where first a control flow is discovered which is then configured.

Using the results presented in this paper we can improve model merging techniques by considering the actual intended behavior instead of the process model structure. We also plan to develop more sophisticated techniques for the ETM-algorithm to directly mine configurable models from collections of event logs. For example, we plan to add configuration-specific mutation operators and learn good parameter settings (using large collections of real-life event logs from the CoSeLoG project). Moreover, we plan to consider other perspectives (e.g., data-, resource- and time-related aspects) and further develop the new area of cross-organizational mining [6]. The ultimate goal is to support organizations in selecting a suitable configuration based on their recorded behavior.

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. 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.
4. Jörg Becker, Patrick Delfmann, Alexander Dreiling, Ralf Knackstedt, and Dominik Kuropka. Configurative Process Modeling–Outlining an Approach to increased Business Process Model Usability. In *Proceedings of the 15th IRMA International Conference*, 2004.
5. 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.
6. J.C.A.M Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Towards Cross-Organizational Process Mining in Collections of Process Models and their Executions. In *Business Process Management Workshops*, pages 2–13. Springer, 2012.
7. J.C.A.M. Buijs, M. La Rosa, H.A. Reijers, B.F. Dongen, and W.M.P. van der Aalst. Improving Business Process Models using Observed Behavior. In *Proceedings of the Second International Symposium on Data-Driven Process Discovery and Analysis*, LNBIP. Springer, 2013 (to appear).
8. F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable Workflow Models. *International Journal of Cooperative Information Systems (IJCIS)*, 17(2), 2008.
9. F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Merging Event-driven Process Chains. In *OTM 2008, Part I, CoopIS 2008*, volume 5331 of *Lecture Notes in Computer Science*, pages 418–426, Berlin Heidelberg, 2008. Springer Verlag.
10. F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Mining Reference Process Models and their Configurations. In *OTM 2008 Workshops*, volume 5333 of *Lecture Notes in Computer Science*, pages 263–272, Berlin Heidelberg, 2008. Springer Verlag.
11. Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Capturing Variability in Business Process Models: The Provop Approach. *Journal of Software Maintenance*, 22(6-7):519–546, 2010.
12. 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), 2012.
13. C. Li, M. Reichert, and A. Wombacher. The MINADEPT Clustering Approach for Discovering Reference Process Models Out of Process Variants. *International Journal of Cooperative Information Systems*, 19(3-4):159–203, 2010.
14. 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.
15. M. La Rosa, F. Gottschalk, M. Dumas, and W.M.P. van der Aalst. Linking Domain Models and Process Models for Reference Model Configuration. In J. Becker and P. Delfmann, editors, *Informal Proceedings of the 10th International Workshop on Reference Modeling (RefMod 2007)*, pages 13–24. QUT, Brisbane, Australia, 2007.
16. M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modeling Language. *Information Systems*, 32(1):1–23, 2007.
17. D. Schunselaar, E. Verbeek, W.M.P. van der Aalst, and H. Reijers. Creating Sound and Reversible Configurable Process Models Using CoSeNets. In W. Abramowicz, D. Kriksciuniene, and V. Sakalauskas, editors, *Business Information Systems (BIS 2012)*, volume 117 of *LNBIP*, pages 24–35. Springer, 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 *Information System Evolution*, volume 72, pages 60–75. Springer, 2011.