Discovering Hierarchical Consolidated Models from Process Families

Nour Assy(✉), Boudewijn F. van Dongen, and Wil M.P. van der Aalst

Department of Mathematics and Computer Science,
Eindhoven University of Technology, Eindhoven, The Netherlands
{n.assy,b.v.f.dongen,w.m.p.v.d.aalst}@tue.nl

Abstract. Process families consist of different related variants that represent the same process. This might include, for example, processes executed similarly by different organizations or different versions of a same process with varying features. Motivated by the need to manage variability in process families, recent advances in process mining make it possible to discover, from a collection of event logs, a generic process model that explicitly describes the commonalities and differences across variants. However, existing approaches often result in flat complex models where it is hard to obtain a comparative insight into the common and different parts, especially when the family consists of a large number of process variants. This paper presents a decomposition-driven approach to discover hierarchical consolidated process models from collections of event logs. The discovered hierarchy consists of nested process fragments and allows to browse the variability at different levels of abstraction. The approach has been implemented as a plugin in ProM and was evaluated using synthetic and real-life event logs.

Keywords: Process mining · Consolidated process families · Hierarchical configurable models · Decomposed discovery · Configurable fragments

1 Introduction

As event data are becoming omnipresent, the importance of process mining is becoming more and more significant. Process mining allows to automatically discover, analyse and improve business processes from execution data referred to as *event logs* [1]. Traditionally, event logs are assumed to describe the execution of static and homogeneous processes. However, business requirements and regulations are continuously changing, and so are the processes. Municipalities, banks, telecommunication service providers and many others execute the same processes but with personalized features. For example, [2] reports on about 100 process variants executed by an asset management company to handle assets for institutional clients and fund distributors. This results in a family of related event logs that can be mined to discover their underlying process variants.

Discovering a collection of disconnected variants creates redundancy and turns the management and maintenance of the process family a difficult task [3].

Instead, organizations need to efficiently analyze and track changes in their processes in a unified way. Recent advances in process mining make it possible to mine process variants and to discover a generic consolidated process model (e.g. [4,5]). However, as the number of process variants increases, it becomes more common to observe partly shared behavior between a subset of the variants instead of one global behavior shared between all the variants. As a result, the discovered consolidated models may quickly become large and complex [6].

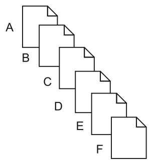
To remedy this problem, we propose an approach to discover hierarchical consolidated process models from collections of event logs. The discovered hierarchy helps in taming the complexity of consolidated models in two ways: (i) by modeling variability at different levels of abstraction (i.e. variability is shown/hidden according to the desired level of abstraction) and (ii) by expressing variability in a coarse-grained way (i.e. commonalities and differences between process fragments instead of individual elements).

Figure 1 describes the problem addressed and the desired output. Given a process family consisting of a collection of events logs, state of the art discovery approaches produce flat consolidated models as shown in Fig. 1b. The nodes' sizes give an indication about the number of variants in which the activities appear. Some edges are annotated with the logs' identifiers from which they are discovered. Clearly, the flat structure of the model and the increasing number of variants make it hard to compare variants and to track where do they agree or disagree unless applying some filtering techniques (e.g. filtering on digests [7]).

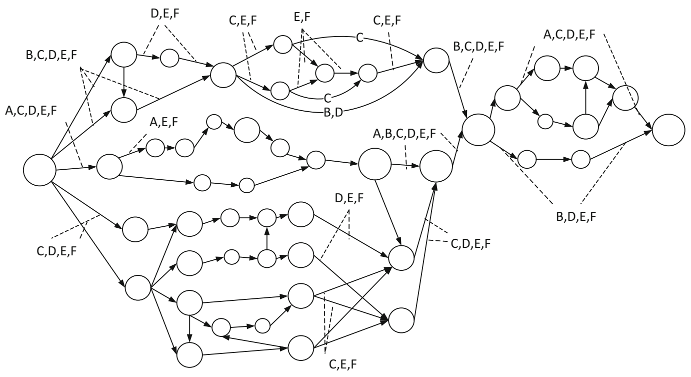
The hierarchical model discovered by our approach is shown in Fig. 1c. The hierarchical structure allows to browse the process variability at different levels of abstraction. The elements in the hierarchy refer to abstracted process fragments shared between a (sub)set of process variants. To describe fragments, we introduce the concept of *SHared-Entry SHared-Exit (SHESHE)* which is inspired from the well-know concept of Single-Entry Single-Exit (SESE) [8]. SHESHEs are independent subprocesses that are entered and exited via *shared paths*. They have well-defined interfaces through which they interact with the rest of the process. Their internal behavior encloses the variability between the entering and exiting variants. The variability abstraction is achieved by hiding the internal behavior of each fragment and by keeping the interaction of its boundaries with the rest of the process visible. Therefore, going down in the hierarchy corresponds to expanding the internal behavior of abstracted fragments. For example, in the process model shown at `Level 2`, the internal behavior of the parent at `Level 1` is expanded and the abstracted nested fragments at `Level 2` become visible.

The approach has been implemented as a plugin in ProM and was evaluated using both synthetic and real-life event logs. Experimental results show that the hierarchical structure does not only reduce the structural complexity of consolidated processes but also can improve their behavioral quality.

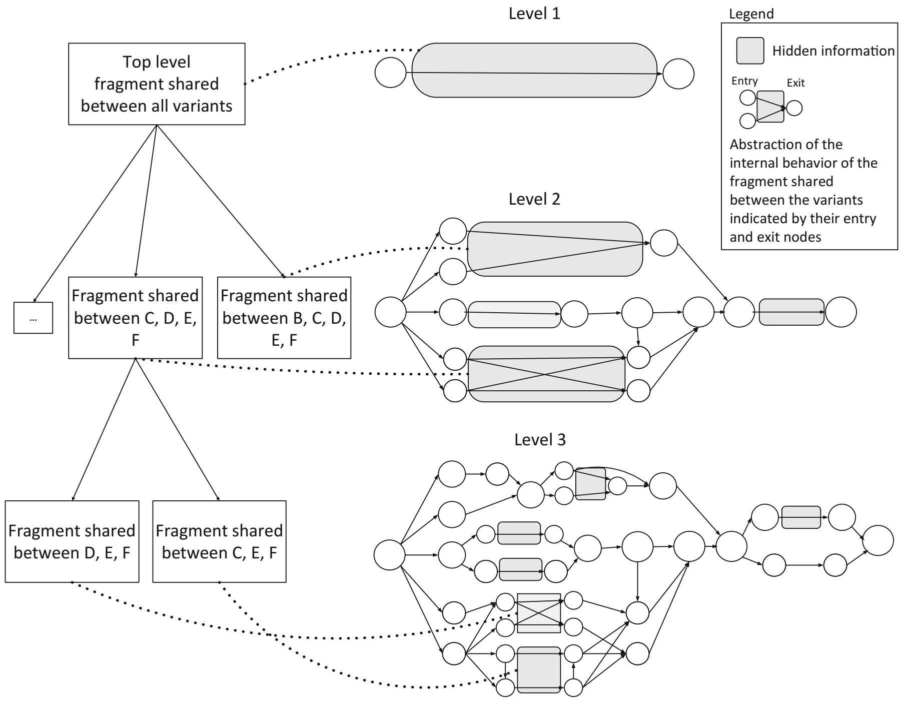
The paper is structured as follow. In Sect. 2, related work is discussed. Some basic definitions used throughout the paper are introduced in Sect. 3. The proposed approach is detailed in Sect. 4 through a running example. Section 5



(a) Input: collection of event logs recording the execution of a process family



(b) Output of traditional approaches: a flat consolidated model where variability (commonalities vs differences) is expressed at the process elements' level; For readability, some of the edges are annotates with the logs' identifiers from which they are discovered


Level 1
Legend
Top level fragment shared between all variants
Hidden information
Entry
Exit
Abstraction of the internal behavior of the fragment shared between the variants indicated by their entry and exit nodes
Level 2
...
Fragment shared between C, D, E, F
Fragment shared between B, C, D, E, F
Level 3
Fragment shared between D, E, F
Fragment shared between C, E, F

(c) Output of the proposed approach: a hierarchical consolidated model where variability is expressed at the fragment level and is shown/hidden according to the desired level of abstraction

Fig. 1. Snapshot of the problem addressed and the proposed approach

reports our experimental results. Finally, we discuss some limitations and extensions of the proposed work in Sect. 6.

2 Related Work

Process variant management is a recent research body that addresses the problem of modeling and maintaining large collections of related process variants [3]. Approaches developed in this area aim to represent the variants in a consolidated manner. For this purpose, conventional process modeling languages have been extended to explicitly support process variability modeling [9]. In this work, we adopt the configurable modeling approach [10] since it allows to explicitly represent the common and different parts in one customizable model.

To consolidate a collection of process variants, two techniques can be applied: model-merging (e.g. [2,7]) and model-mining (e.g. [4,5]). Model-merging techniques construct consolidated process models by *structurally* merging existing process variants. Model-mining techniques use process mining to discover consolidated process models from the execution behavior of process variants. All of the existing approaches produce flat models with fine-grained variability.

To overcome the complexity of consolidated models, different approaches have been proposed to abstract from the fine-grained variability expressed at the process elements' level to a coarse-grained variability expressed using domain oriented models (e.g. [11,12]). The drawback of these approaches is that they heavily rely on the domain expert knowledge. Another stream of works try to tame the complexity by localizing variability in fragments instead of entire processes (e.g. [6,13]). Subprocesses, often expressed in terms of SESE fragments, are extracted from existing process variants and are consolidated if they are similar. These techniques use model merging. The merged fragments show a local view on the variability, while in our work, we aim at providing a global view of the consolidated model and at different levels of abstraction.

Divide and conquer techniques have been also proposed to solve the complexity of discovered models. They can be grouped into two categories: case-based and activity-based. In case-based techniques, the log is split into homogeneous clusters of traces and a model is mined per cluster (e.g. [14]). The aim here is to mine a collection of simpler variants of a complex process. In this work, we deal with activity-based decomposition where traces are split into clusters of subtraces. A fragment is mined from each cluster and finally, the mined fragments are glued together into an overall model. In [15], a generic approach is presented and the principles of *correct decomposition* are discussed. The decomposition technique proposed in this paper rely on the theoretical results presented in [15].

Finally, our work can be also related to **process model abstraction** (e.g. [16,17]) and **hierarchical process discovery** (e.g. [18]). In model abstraction, the aim is to create simpler views of the process by abstracting from process details. The focus of existing works has been on defining aggregation and hiding operators that preserve some correctness criteria. The focus of our work is not on defining new abstraction operators. Instead, we aim at automatically creating different abstraction levels of the variability in a process family. For this purpose, we introduce the notion of SHESHE fragments which, by definition, allows us to achieve our goal. On the other hand, the works on hierarchical process discovery explore the information recorded in event logs in order to infer a hierarchical structure that can explain the flow of the process. In our work, we discover a hierarchical structure that explains *the flow of the variability* in a process family.

3 Preliminaries

Set, Multiset. Let S be a finite set. The multiset $B(S)$ over S is a set where elements may appear multiple times. The elements in the multiset are listed between square brackets. For example, $B = [\ ]$ is the empty multiset, $B(S) = [a, a, b, c, b] = [a^2, b^2, c]$ is a multiset over $S = \{a, b, c\}$. $\mathcal{B}(S)$ is the set of all multisets over S.

Sequence, Projection. A sequence $\sigma = \langle s_1, s_2, ..., s_n \rangle \in S^*$ is an ordered list of elements. The empty sequence is denoted as $\langle\ \rangle$. The projection of σ on a subset $S' \subseteq S$ denoted as $\sigma_{\upharpoonright S'}$ is a subsequence of σ containing only the elements of S'. For example $\langle a, a, b, d \rangle_{\upharpoonright \{a,d\}} = \langle a, a, d \rangle$. Projection is also defined for multisets. For example, $[a^3, b^2, c]_{\upharpoonright \{a,b\}} = [a^3, b^2]$.

Event Log. An event log is a multiset of traces. A trace is a sequence of activities describing the lifecycle of a particular process instance. Let $\mathcal{A} \subseteq \mathcal{U}_A$ be a set of activities in some universe of activities. A trace $\sigma \in \mathcal{A}^*$ is a sequence of activities. $L \in \mathcal{B}(\mathcal{A}^*)$ is an event log. We denote by $A_L = \{a \in \sigma \mid \sigma \in L\}$ the set of activities occurring in L.

Causal Graph. A causal graph $C_L = (A_L, E_L)$ constructed from an event log L is a graph showing the causal relations between the log activities. Most process mining algorithms build such a graph in a preprocessing step by scanning the event log to see how many times an activity a_1 is followed by another activity a_2. If this occurs above a certain threshold, then it is assumed that a_1 causally precedes a_2 (i.e. $(a_1, a_2) \in E_L$). The selection of an appropriate threshold is out of scope of this paper. In this work, we assume that C_L is constructed using an existing algorithm. We also assume that C_L is connected.

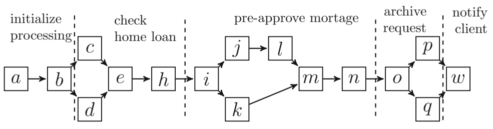
Configurable Process Models. Configurable process models allow to explicitly represent the common and different parts in one customizable process model. They need to be configured to specific requirements by (de)selecting (ir)relevant parts. The essence of configuration can be captured in terms of two operators, *hiding* and *blocking* [10]. Hiding an activity corresponds to skipping it. In other words, the activity is either removed or renamed to a silent step. Blocking an activity corresponds to disabling it, i.e. the path from the activity cannot be taken anymore. There exist several extensions to existing process modeling notations. They all share the same configuration basis but differ according to the language notation. In this work, we use Petri nets as a process modeling notation since a great number of the process mining techniques assume or generate Petri nets. However the results are not restricted to this notation.

4 Proposed Approach

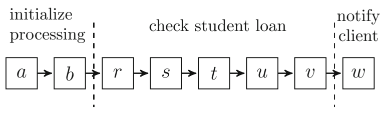
In this section, we first present a running example (Sect. 4.1) that will be used to illustrate the three steps of our discovery approach (Sects. 4.2, 4.3 and 4.4).

4.1 Running Example

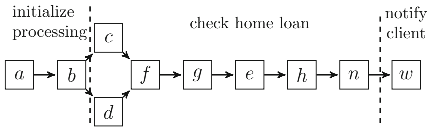
We consider a scenario of four different variants of a loan process: home, student, business and small. Example event logs corresponding to these variants are as follow: $L_1 = [\langle a, b, c, d, e, h, i, j, l, k, m, n, o, p, w\rangle^{47}, \langle a, b, d, c, e, h, i, j, k, l, m, n, o, q, w\rangle^{28}, \langle a, b, d, c, e, h, i, k, j, l, m, n, o, p\rangle^{25}]$; $L_2 = [\langle a, b, r, s, t, u, v, w\rangle^{50}]$; $L_3 = [\langle a, b, c, d, f, g, e, h, n, w\rangle^{48}, \langle a, b, d, c, f, g, e, h, n, w\rangle^{52}]$; $L_4 = [\langle a, b, r, s, t, v, s, u, v, o, p, q, w\rangle^{48}, \langle a, b, r, s, u, v, o, q, p, w\rangle^{52}]$. Figure 2 shows the causal graphs of the four event logs. To ease the understanding of the processes, we split and annotate them with the names of different phases. There are in total 22 distinct activities. Three activities appear in all the four event logs and 13 activities appear in different subsets of the logs.



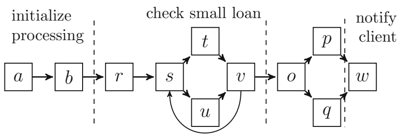
(a) C_1: home loan process



(b) C_2: student loan process



(c) C_3: business loan process



(d) C_4: small loan process

a: receive application	m: loan type approved	g: check revenue	s: check amount
b: start processing	n: request approval	h: end checking	t: check system funds
c: register client	o: check income	i: verify mortage type	u: check eligibility (small)
d: check client credit	p: archive request	j: check property information	v:end checking phase
e: contact bank	q: contact client	k: request mortage underwriter	
f: check company funds	r: check profile	l: check eligibility (big)	

(e) Activities' names

Fig. 2. The causal graphs of four event logs of different loan process variants

Our approach for discovering a hierarchical consolidated model consists of three main steps:

1. An integrated representation of the causal graphs is created and is referred to as *multi causal graph* (Sect. 4.2);

2. The multi causal graph is decomposed into a hierarchical structure of nested SHESHE fragments. For this, we formally introduce the concept of SHESHE and their induced hierarchical structure (Sect. 4.3);
3. The abstraction of SHESHE fragments is generated and the event logs are mined accordingly to discover a hierarchical consolidated model (Sect. 4.4).

4.2 Multi Causal Graph Construction

Given a collection of event logs $L_1, \ldots, L_n \in \mathcal{B}(\mathcal{A}^*)$, we first unify the start and end of all the event logs by adding start (ST) and end (ET) activities to all the traces. Then, we construct a causal graph C_i for each event log L_i. The causal graphs are merged into a multi-causal graph. Edges in the multi causal graph have different identities according to the causal graph from which they originate. In this work, we assume that activities having the same labels are identical.

Definition 1 (Multi causal graph). *Let $\mathfrak{L} = \{L_i \mid i \geq 2\}$ be a collection of events logs with unique start (ST) and end activities (ET) and $C_i = (A_i, E_i)$ a causal graph constructed for each $L_i \in \mathfrak{L}$. A multi causal graph $C^\circ = (A^\circ, E^\circ)$ is the graph resulting from merging the causal graphs such that $A^\circ = \bigcup_{i \geq 2} A_i$ and $E^\circ = \bigcup_{i \geq 2} \{((a_1, a_2), i) \mid (a_1, a_2) \in E_i\}$.*

Figure 3 shows an example of the multi causal graph resulting from merging the causal graphs in Fig. 2 (for now ignore the dashed rectangles).

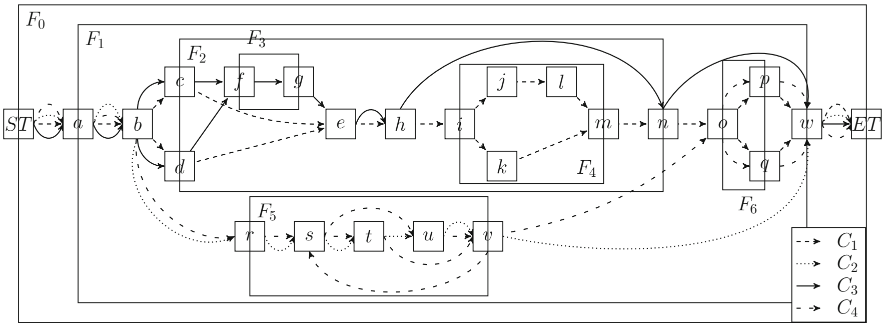


Fig. 3. Maximal SHESHE decomposition on the multi causal graph resulting from merging the causal graphs in Fig. 2

4.3 SHESHE Decomposition

The second step of our approach is to find a hierarchical decomposition of the multi causal graph in terms of fragments organized in a *hierarchical containment* relationship. To define fragments, we introduce the concept of *SHared-Entry SHared-Exit (SHESHE)*. SHESHE fragments are subprocesses with well-defined interfaces through which they interact with the rest of the process. They represent parts in the process where variants *enter and exit through the same interfaces*. In the following, we give some definitions to introduce a SHESHE.

Let $\mathfrak{L} = \{L_i \mid i \geq 2\}$ be a collection of event logs with unique start (ST) and end (ET) activities; $C_1, \ldots, C_n$ be the casual graphs constructed for each $L_i \in \mathfrak{L}$ and $C^\circ = (A^\circ, E^\circ)$ be the multi causal graph constructed from $C_1, \ldots, C_n$.

Definition 2. *Let $F \subseteq E^\circ$ be a non empty set of weakly connected edges. We define the following notations:*

- *The shareability level of F: $id(F) = \{i \mid ((a_1, a_2), i) \in F\}$;*
- *The edges with the identifier i in F: $F_{\restriction i} = \{((a_1, a_2), i') \in F \mid i' = i\}$;*
- *The set of activities: $act(F) = \bigcup_{((a_1,a_2),i) \in F} \{a_1, a_2\}$;*
- *The complement of F in E°: $\overline{F} = E^\circ \setminus F$;*
- *For an activity $a \in act(F)$, the following is defined:*
  - *The incoming edges that belong to F: $in(a, F) = \{((a_1, a), i) \in F\}$;*
  - *The incoming edges that do not belong to F: $\overline{in}(a, F) = in(a, \overline{F}) = \{((a_1, a), i) \in \overline{F}\}$;*
  - *The outgoing edges that belong to F: $out(a, F) = \{((a, a_1), i) \in F\}$;*
  - *The outgoing edges that do not belong to F: $\overline{out}(a, F) = out(a, \overline{F}) = \{((a, a_1), i) \in \overline{F}\}$;*
  - *The edges connected to a: $conn(a, F) = in(a, F) \cup \overline{in}(a, F) \cup out(a, F) \cup \overline{out}(a, F)$.*

The activities in F can be split into three categories: local if the activity is connected only to activities in F, boundary if it is connected to elements not in F, and shared boundary if, in addition to being boundary, the activity is connected only to edges having the identifier included in the shareability level of F. This latter category allows us to introduce the concept of SHESHE.

Definition 3 (Local, boundary, shared boundary). *Let $F \subseteq E^\circ$ be a non empty set of weakly connected edges such that $ST, ET \notin act(F)$. The local, boundary and shared boundary nodes in F are defined as following:*

- *$lact(F) = \{a \in act(F) \mid \overline{in}(a, F) \cup \overline{out}(a, F) = \emptyset\}$;*
- *$bact(F) = \{a \in act(F) \mid a \notin lact(F)\}$;*
- *$sact(F) = \{a \in bact(F) \mid id(conn(a, F)) = id(F)\}$.*

For example, in Fig. 3, the set of edges represented by the fragment F_5 has: $lact(F_5) = \{s, t, u\}$, $bact(F_5) = \{r, v\}$ and $sact(F_5) = \{r, v\}$. The boundary activities in F can be further classified into entry and exit activities. In [8], SESEs are defined as sets of edges having *single entry* and *single exit* activities. In this work, we define SHESHEs as sets of edges having *shared entry* and *shared exit* nodes. Roughly speaking, a shared entry is a shared boundary through which it is possible for every variant to enter inside the region represented by F. The same holds for a shared exit. For example, in Fig. 3, c and d are shared entries for F_2 since both variant 1 (which corresponds to L_1) and variant 3 can enter F_2 through them; n is a shared exit, since both variants can exit F_2 through it. The formal definition of shared entry and exit nodes is given in Definition 4.

Definition 4 (Shared entry and exit nodes). *Let $F \subseteq E^{\circ}$ be a non empty set of weakly connected edges such that $ST, ET \notin act(F)$. A node $a \in sact(F)$ is a shared entry iff:*

1. $id(\overline{in}(a, F)) = id(out(a, F)) = id(F)$ *and*
2. $[(\forall_{i \in id(F)}\ \overline{out}(a, F)_{\restriction i} = \emptyset \ \vee \ in(a, F)_{\restriction i} = \emptyset)$ *or*
3. $(id(\overline{out}(a, F)) = id(in(a, F)) = id(F))]$.

$a \in sact(F)$ is a shared exit iff:

1. $id(\overline{out}(a, F)) = id(in(a, F)) = id(F)$ *and*
2. $[(\forall_{i \in id(F)}\ \overline{in}(a, F)_{\restriction i} = \emptyset \ \vee \ out(a, F)_{\restriction i} = \emptyset)$ *or*
3. $(id(\overline{in}(a, F)) = id(out(a, F)) = id(F))]$.

We denote by $entry(F)$ and $exit(F)$ the set of entry and exit nodes respectively.

The first requirement of a shared entry states that the set of identifiers of the incoming edges that do not belong to F should be equal to the set of identifiers of the outgoing edges that belong to F, which in turn, should be equal to the shareability level of F. This requirement ensures that all variants can enter F through the entry node. Since we do not impose single entry nodes, the second and third requirement ensure that the entry node can either be only an entry (i.e. no variant can exit through this node) or a shared exit (i.e. all variants have the possibility to exit the node).

Definition 5 (Shared-Entry Shared-Exit). *Let $F \subseteq E^{\circ}$ be a non empty set of weakly connected edges such that $ST, ET \notin act(F)$. F is a shared-entry shared-exit iff: $act(F) = lact(F) \cup entry(F) \cup exit(F)$.*

A SHESHE fragment allows to localize the variability between a subset of variants. As all involved variants can enter and exit through its entry and exit nodes, the fragment can be treated as a black box (similar to a macro activity) and the variability can be expressed at the fragment level. Blocking a fragment corresponds to blocking the subprocess executed by the corresponding variants. On the process elements' level, this requires only to blocking the fragment interfaces (i.e. entry and exit activities). On the other hand, any local configuration inside the fragment does not affect its interfaces and therefore is independent from outside. Examples of SHESHE fragments are shown in Fig. 3.

The edges formed by E° define a specific type of a SHESHE with no incoming or outgoing edges. This fragment is called the root and is not considered in any operation on SHESHEs. In this work, we are interested in a hierarchical representation of SHESHE fragments where a child fragment restricts the shareability level of its parent. This requires that SHESHE fragments do not overlap. In the following, we give some definitions to derive non-overlapping SHESHEs.

Lemma 1 (SHESHE inclusion). *Let $F_1, F_2 \subseteq E^{\circ}$ be two SHESHEs such that $F_1 \subseteq F_2$. The following hold: (i) $id(F_1) \subseteq id(F_2)$ and (ii) if $entry(F_1) \cap entry(F_2) \neq \emptyset$ or $exit(F_1) \cap exit(F_2) \neq \emptyset$ then $id(F_1) = id(F_2)$.*

Lemma 2 (SHESHE union). *Let $F_1, F_2 \subseteq E^\circ$ be two SHESHEs and $F = F_1 \cup F_2$ be their union such that F is weakly connected. F is a SHESHE iff $id(F_1) = id(F_2)$ or $F_1 \subseteq F_2$.*

Many of the SHESHEs in a multi causal graph are *less informative*. For instance, in Fig. 3, $F'_1 = \{((e,h),1),((e,h),2)\} \subset F_1$ [1] has the same shareability level $\{1,2\}$ as F_1. Given F_1, F'_1 does not provide any additional information. This does not hold for $F_8 \subset F_1$ as F_8 has a new restricted shareability level. In this work, we are interested in *maximal* SHESHE fragments that are the largest fragments having a specific shareability level.

Definition 6 (Maximal SHESHE). *Let $F \subseteq E^\circ$ be a SHESHE. F is maximal iff $\nexists F' \subseteq E^\circ, F' \neq F$ where $F \cup F'$ is a SHESHE and $id(F) = id(F')$.*

Proposition 1 (Non overlapping SHESHE). *Let $F, F' \subseteq E^\circ$ be two maximal SHESHEs. One of the three statements holds: (i) $F \subseteq F'$, (ii) $F' \subseteq F$ or (iii) $F \cap F' = \emptyset$.*

Proposition 1 allows to derive a hierarchical topology of the SHESHE fragments organized in a tree-like structure (Fig. 3 shows an example).

Definition 7 (SHESHE decomposition tree). *Let C° be a multi causal graph and $\mathcal{F}$ be the set of its maximal SHESHEs including E°. $D = (\mathcal{F}, \prec)$ is a tree of maximal SHESHE fragments where $\mathcal{F}$ is the set of tree nodes and $\prec \subseteq \mathcal{F} \times \mathcal{F}$ is the set of parent-child relations such that $(F_1, F_2) \in \prec$ iff $F_2 \subset F_1$ and F_1 is the smallest SHESHE containing F_2.*

4.4 Hierarchical Discovery

Given a SHESHE decomposition tree, we aim at discovering a hierarchical consolidated model that shows the fragments at different levels of abstraction. The abstraction of fragments allows to show the *most shared behavior* and to *hide the local variability* between the involved variants. For instance, given F_2 in Fig. 3, we would like to see that the variants of both L_1 and L_3 start with executing c and d; both execute the fragment represented by e and h and end with the execution of n; the fragments F_3 and F_4 are hidden since they are executed by L_3 and L_1 respectively and therefore depict a local variability. In terms of event logs, this abstraction can be achieved by projecting the traces of the logs L_1 and L_3 on the activities c, d, e, h and n and discovering the corresponding fragment. The fragment is discovered by merging the projected traces of L_1 and L_3 into one log and using existing discovery techniques (e.g. Inductive Miner).

In addition to the shared behavior, we should be able to link the discovered abstracted fragments with their children in order to create the entire process. For example, the abstracted fragment of F_2 includes the activities c, d, e, h and n; the abstracted fragment of F_3 includes f and g and the abstracted fragment

[1] This SHESHE is not shown because it is not maximal according to Definition 6.

of F_4 includes i, j, l, k and m. In order to construct the whole behavior of F_2 (i.e. non abstracted version), we should be able to link the discovered abstracted fragment of F_2 with its children. In this case two solutions are possible. The trivial one is to restart from scratch by projecting the traces of L_1 and L_3 on all the activities of F_2 and discovering their corresponding fragment. However, this solution requires an excessive and repeated work of discovery which may be expensive in case of a large number of parent-child relations in the tree.

Another more convenient solution is to include the boundary nodes of children fragments in their abstracted parent fragment. In this way the children boundary nodes act as a glue and allow to create the entire process by combining the discovered abstracted parents with their discovered abstracted children. This solution corresponds to the output of the approach shown in Fig. 4.

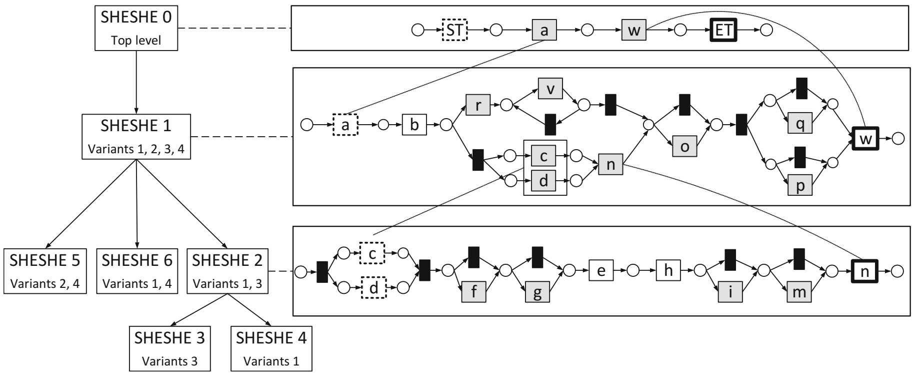


Fig. 4. Output of the discovery approach taking as input the logs of the variants in Fig. 2. Transitions with dashed line represent the entry of the SHESHE fragment; those with bold line represent the exit; gray transitions are the boundaries of the SHESHE children. The fragments are discovered using Inductive Miner

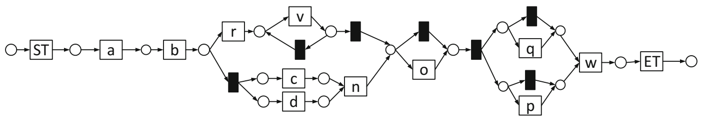


Fig. 5. A model provided at the second level of the tree in Fig. 4 by glueing the discovered fragment of SHESHE 0 and its children (in this case SHESHE 1)

The tree shows the abstracted fragments of the SHESHE decomposition tree in Fig. 3. The discovered fragments of some nodes are shown on the right-hand side. For example, the discovered fragment of SEHSHE 2 which corresponds to the abstracted version of F_2 includes the entry and exit of F_2, the shared behavior including the activities e and h as well as the boundary nodes of SHESHE 3 and SHESHE 4. Each level of the tree shows a local behavior on the abstracted SHESHE. To construct an entire process, parents and children can be recursively glued at each level using the approach presented in [15]. For example, the model shown in Fig. 5 corresponds to the entire process of the second level in the tree. It is obtained by glueing together the fragment of SHESHE 0 and its children (in this case SHESHE 1). The entire process of the third level can be obtained by glueing the entire process of the second level with the children of the third level (i.e. SHESHE 5, 6 and 2) and so on.

Having such hierarchical structure, it becomes easy to express variability by identifying the configurable elements. The tree nodes *shared between a subset of the variants* are configurable. They can be either blocked or allowed. On the process level, this corresponds to making the entry and exit activities configurable.

5 Evaluation

The approach has been implemented as a plugin in ProM[2](www.processmining.org). The plugin takes a collection of event logs as input and produces a hierarchical consolidated process model as output.

5.1 Synthetic Logs

We used event logs of four variants of a travel booking process[3]. The base process allows for booking a flight with the option of booking a hotel and/or a car in a subset of the variants. Figure 6a shows the flat configurable process model discovered using Inductive Miner (IM) with the default parameters in ProM. Because of the high variability in the logs, IM generates an underfitting model (i.e., a flower construct) which allows for any behavior. All the activities in the flower construct are configurable. This model is simple but scores very low on precision. With Heuristics Miner (HM), a better model is discovered. However, the number of configurable nodes is 21 which is still relatively high.

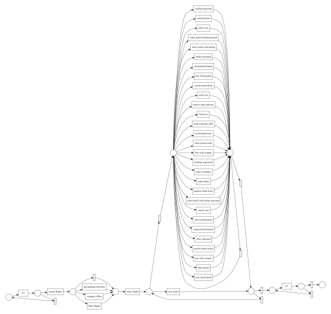
Figure 6b shows the hierarchical structure discovered using our approach. The tree hierarchy contains 5 configurable fragments (i.e. 5 out of 7 nodes are shared between a subset of the variants). Figures 6c and d show the discovered root fragment SHESHE 0 and the fragment SHESHE 4 using IM. Compared to the flower model in Fig. 6a, the flower construct in the root fragment is reduced and is completely broken in the child fragment.

5.2 Real-Life Logs

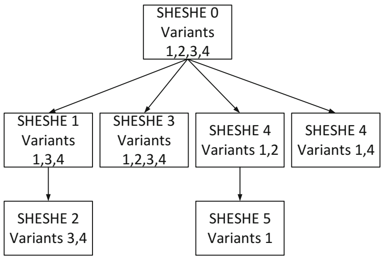
We used the dataset from BPI challenge 2015 [19] which corresponds to five process variants of building permit applications executed by five Dutch municipalities. We evaluated the quality, in terms of structural complexity and behavioral

[2] https://svn.win.tue.nl/repos/prom/Packages/NourAssy/.

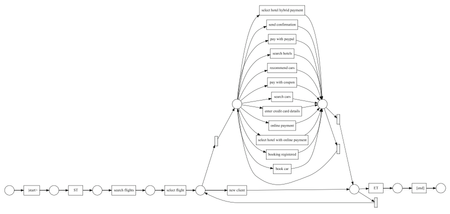
[3] The process models and logs can be downloaded from: https://svn.win.tue.nl/repos/prom/Packages/NourAssy/Trunk/artificialLogs.



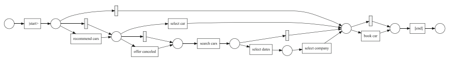
(a) IM: flower model - all activities are configurable



(b) IM: discovered hierarchy (5 configurable fragments)



(c) fragment of SHESHE 0 shared between all variants



(d) fragment of SHESHE 4 shared between variants 1 and 2

Fig. 6. (a) flat vs (b), (c) and (d) hierarchical discovered process models

accuracy, of the discovered models using traditional approaches for discovering flat models versus using our approach for discovering hierarchical models.

The structural quality of the discovered models is measured in terms of the hierarchy quality and the fragments complexity. To evaluate the quality of the hierarchy, we compute (i) the number of configurable nodes in the discovered models (# conf. nodes), (ii) the average number of configuration steps needed to derive a variant (# conf. dec.), (iii) the depth of the tree and (iv) the average number of shared variants per node (# Avg. share). Regarding the fragments' complexity, we measured their average size and compared it to the size of one flat model (Size(# arcs)). The behavioral accuracy is measured in terms of fitness, precision and generalization of the merged log against the discovered consolidated model. The results are reported in Tables 1 and 2.

The results show that, with the hierarchical discovery, the sizes of fragments in the hierarchy are greatly reduced compared to the size of one flat model. Instead of looking to one *big* process model, one can scan the hierarchy and inspect smaller fragments. On the other hand, the depth of the tree is small. This means that the decomposition was not able to find a hierarchy of nested fragments. Because of the small depth, the difference in the number of configurable nodes between the hierarchical model and the flat model is not in our favour. In the hierarchical model, the fragments shared between different variants contain shared activities but exhibit different behavior (this is explained by the high number of configurable nodes in the flat model). The reasons for this can be explained by the fact that we assumed that activities with the same labels are the same across variants. By doing so, we reduce the chance to find valid SHESHEs. Regarding the behavioral quality, we noticed that our decomposition approach was able to improve the precision of hierarchical models with a small decrease in the fitness. The decrease in the fitness is not caused by the SHESHE decomposition itself, but because of the glueing step presented in Sect. 4.4.

Table 1. Structural quality of the discovered flat vs hierarchical model

Dataset	Flat			Hierarchical				
	# conf. nodes	# conf. dec	Size	Depth	Avg. share	#conf. nodes	# conf. dec	Size frag
BPI 15	29	16.6	102	2	2.92	7	2	45.53

Table 2. Behavioral quality of the discovered flat model vs hierarchical model

		Fitness	Precision	Generalization
IM	Flat	1	0.1	1
	Hierarchical	0.86	0.57	0.97
	Hierarchical (fragment avg.)	0.99	0.84	0.44
HM	Flat	0.93	0.76	0.94
	Hierarchical	0.96	0.8	0.95
	Hierarchical (fragment avg.)	0.99	0.99	0.4

6 Conclusion

In this paper, we presented an approach for mining hierarchical consolidated models from process families. The hierarchy allows to (i) browse the variability at different levels of abstraction and to (ii) model it in a coarse-grained way. Through experimental evaluation, we showed that our decomposition approach is suitable to tame the complexity of consolidated models.

As already shown in Sect. 5, the quality of the discovered hierarchy highly depends on the way the equivalence class is defined over the logs' activities. In the present work, we simply assumed that activities with common labels are equal. However, one interesting feature of our shared-entry shared-exit fragments is that they allow to *bring structure to unstructured variability*. Therefore, in our future work, we will investigate the problem of discovering consolidated process models with structured variability. This calls for the problem of finding an equivalence class over the logs' activities that optimizes the SHESHE decomposition quality.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd edn. Springer, Heidelberg (2016)
2. Reijers, H.A., Mans, R.S., van der Toorn, R.A.: Improved model management with aggregated business process models. Data Knowl. Eng. **68**(2), 221–243 (2009)
3. Dijkman, R.M., La Rosa, M., Reijers, H.A.: Managing large collections of business process models - current techniques and challenges. Comput. Ind. **63**(2), 91–97 (2012)
4. Buijs, J.C.A.M., Dongen, B.F., van der Aalst, W.M.P.: Mining configurable process models from collections of event logs. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 33–48. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40176-3_5
5. Li, C., Reichert, M., Wombacher, A.: Mining business process variants: challenges, scenarios, algorithms. Data Knowl. Eng. **70**(5), 409–434 (2011)
6. Milani, F., Dumas, M., Ahmed, N., Matulevicius, R.: Modelling families of business process variants: a decomposition driven method. Inf. Syst. **56**, 55–72 (2016)
7. La Rosa, M., Dumas, M., Uba, R., Dijkman, R.: Business process model merging: an approach to business process consolidation. ACM Trans. Softw. Eng. Methodol. **22**(2), 11:1–11:42 (2013)
8. Johnson, R., Pearson, D., Pingali, K.: The program structure tree: computing control regions in linear time. In: Proceedings of the ACM SIGPLAN 1994 Conference on Programming Language Design and Implementation (PLDI), pp. 171–185 (1994)
9. La Rosa, M., van der Aalst, W.M.P., Dumas, M., Milani, F.P.: Business process variability modeling: a survey. ACM Comput. Surv. **50**(1), 2:1–2:45 (2017)
10. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H., La Rosa, M.: Configurable workflow models. Int. J. Coop. Inf. Syst. **17**(2), 177–221 (2008)
11. La Rosa, M., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Questionnaire-based variability modeling for system configuration. Softw. Syst. Model. **8**(2), 251–274 (2009)
12. Gröner, G., Boskovic, M., Parreiras, F.S., Gasevic, D.: Modeling and validation of business process families. Inf. Syst. **38**(5), 709–726 (2013)
13. La Rosa, M., Dumas, M., Ekanayake, C.C., García-Bañuelos, L., Recker, J., ter Hofstede, A.H.M.: Detecting approximate clones in business process model repositories. Inf. Syst. **49**, 102–125 (2015)
14. García-Bañuelos, L., Dumas, M., La Rosa, M., De Weerdt, J., Ekanayake, C.C.: Controlled automated discovery of collections of business process models. Inf. Syst. **46**, 85–101 (2014)
15. van der Aalst, W.M.P.: Decomposing Petri nets for process mining: a generic approach. Distributed and Parallel Databases **31**(4), 471–507 (2013)
16. Polyvyanyy, A., Smirnov, S., Weske, M.: The triconnected abstraction of process models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 229–244. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03848-8_16
17. Reichert, M., Kolb, J., Bobrik, R., Bauer, T.: Enabling personalized visualization of large business processes through parameterizable views. In: Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, 26–30 March 2012, pp. 1653–1660 (2012)

18. Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: BPMN miner: automated discovery of BPMN process models with hierarchical structure. Inf. Syst. **56**, 284–303 (2016)
19. van Dongen, B.F.: BPI challenge 2015 (2015). http://dx.doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1