THE MINADEPT CLUSTERING APPROACH FOR DISCOVERING REFERENCE PROCESS MODELS OUT OF PROCESS VARIANTS

CHEN LI *
Information Systems Group, University of Twente, The Netherlands,
lic@cs.utwente.nl (contact author)

MANFRED REICHERT
Institute of Databases and Information Systems, University of Ulm, Germany,
manfred.reichert@uni-ulm.de

ANDREAS WOMBACHER
Database Group, University of Twente, The Netherlands,
a.wombacher@utwente.nl

During the last years a new generation of adaptive Process-Aware Information Systems (PAIS) has emerged, which enables dynamic process changes at runtime, while preserving PAIS robustness and consistency. Such adaptive PAIS allow authorized users to add new process activities, to delete existing activities, or to change pre-defined activity sequences during runtime. Both this runtime flexibility and process configurations at build-time, lead to a large number of process variants being derived from the same process model, but slightly differing in structure due to the applied changes. Generally, process variants are expensive to configure and difficult to maintain. This paper presents selected results from our MinAdept project. In particular, we provide a clustering algorithm that fosters learning from past process changes by mining a collection of process variants. As mining result we obtain a process model for which average distance to the process variant models becomes minimal. By adopting this process model as reference model in the PAIS, need for future process configuration and adaptation decreases. We have validated our clustering algorithm by means of a case study as well as comprehensive simulations. Altogether, our vision is to enable full process lifecycle support in adaptive PAIS.

Keywords: process-aware information system; process change; process variants; process mining; process learning

1. Introduction

Economic success of an enterprise increasingly depends on its ability to react to changes in its environment (e.g., market changes or changes of legal regulations) in a quick, flexible and cost-effective way [39,55,40,9]. Along this trend a variety of process support paradigms as well as corresponding process specification and execution

* This work was done in the MinAdept project, which has been supported by the Netherlands Organization for Scientific Research (NWO) under contract number 612.066.512.

languages have emerged. Using WS-BPEL[4], for example, a process can be composed out of existing services. At runtime, the execution of these services is orchestrated by the *Process-Aware Information System* (PAIS) according to the defined process logic.

Generally, different scenarios for adaptive and configurable processes exist.[70] Process adaptations are not only needed for configuration purposes at build-time [16,52], but also become necessary during runtime to deal with exceptional situations and changing needs [41,69,24]; i.e., for single process instances, it should be possible to dynamically adapt their structure, by inserting, deleting or moving process activities and process fragments respectively.

In response to this need adaptive process management technology has emerged.[68,70] Basically, it enables adaptation and configuration of process models at different levels. This, in turn, results in large collections of process model variants (*process variants* for short), which are created from the same process model, but slightly differ in structure from each other. Generally, a large number of process variants may exist in a PAIS [31,70,17]. For example, according to a case study we performed in healthcare domain (details are discussed in Section 5), we have identified more than 90 process variants for one particular healthcare procedure.

In most approaches which allow to adapt and configure process models, the related process variants have to be maintained separately.[16] Then even simple changes in process behavior (e.g. due to new laws) might require manual re-editing of a large number of related process variant models. Over time this leads to degeneration and divergence of these models, which aggravates PAIS maintenance significantly.

1.1. *Problem Statement*

Though considerable efforts have been made to ease process configuration and adaptation [16,41,52], we have not utilized the knowledge resulting from these process model changes yet.[69] Fig. 1 describes the goal of our paper. We aim at learning from past process changes by "merging" existing process variants into one generic process model, which "covers" these variants best. By adopting this generic model as *reference process model* within the PAIS, cost of change and need for future process adaptations will decrease. Based on the two assumptions that (1) process models are well-formed (i.e., block-structured like in WS-BPEL) and (2) all activities in a process model have unique labels[a], we deal with the following fundamental research question:

Given a collection of process variants (i.e., process models), how to derive a reference process model out of them such that the average distance between the discovered reference model and the process variants becomes minimal?

The distance between reference process model and process variant is measured in

[a]The block-structure constraint is discussed in detail in Section 2. Regarding the constraint in respect to unique labeling, we refer to [11] for an approach matching activities with different labels in different process variants.

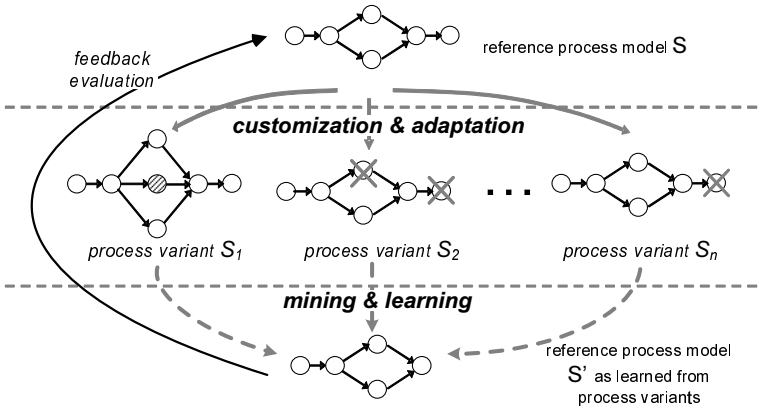


Fig. 1. Mining a new reference model

terms of the number of high-level change operations (e.g., to insert, delete or move activities [41]) needed to transform the reference model into the model of the respective variant. Furthermore, *change distance* directly represents the efforts needed for process adaptation and customization, and average distance between a reference model and a collection of process variants directly measures the configuration efforts for particular reference process model. Obviously, the challenge is to find the "best" reference model, i.e., the one with minimal average distance to the known variants. Note that we only need a collection of process variants as input of our analysis. We do not need a *change log*, which specifically documents all change operations performed during the configuration of process variants [15]. In fact, even the original reference process model from which the variants are derived is not strictly required. In the following we present a clustering technique to deal with these challenges.

1.2. *Contribution*

This paper significantly extends our work previously presented in [25] and provides more technical details and validation results. For example, we relax the constraint of requiring a unique activity set, i.e., we provide an approach to cope with process variants having different activity sets. Further, we consider more workflow patterns (e.g., loop structures) when compared to previous work. For practical validation of our approach, a case study performed in the healthcare domain is added. Finally, this paper includes a detailed description of the implemented proof-of-concept prototype and conducts a simulation to examine scalability of our mining algorithm.

The clustering algorithm presented in this paper is completely different from the heuristics algorithm introduced in [27]: it has less rigid requirements regarding input data (e.g., an original reference process model is not required), and it can provide more detailed information on the discovered model (e.g., to what degree a certain part of the discovered model matches to the variants). Complexity of our clustering algorithm is polynomial, which is significantly lower than the $\mathcal{NP}$-*hard* algorithm

presented in [27].

The remainder of this paper is organized as follows. Section 2 gives background information needed for understanding this paper. In Section 3 we describe fundamental goals for mining process variants and discuss why we need an approach which differs from traditional process mining techniques. Section 4 introduces a method to represent process variant models in a way such that they can be mined effectively. We discuss a case study which we performed in healthcare filed in Section 5. Section 6 presents our basic clustering algorithm for mining process variants. Section 7 extends it such that variants with different activity sets can be considered as well. We validate our algorithm in Section 8 by comparing its performance with existing process mining techniques. We formally specify our algorithm and sketch a proof-of-concept prototype in Section 9. Finally, Section 10 discusses related work and Section 11 concludes with a summary and outlook.

2. Backgrounds

We first introduce basic notions needed in the following:

Process Model: Let $\mathcal{P}$ denote the set of all sound (i.e., correct) process models. We denote a process model as sound if there are no deadlocks or unreachable activities in the process model [41,62]. In our context, a particular *process model* $S = (N, E, \ldots) \in \mathcal{P}$ is defined in terms of an Activity Net [41]: N constitutes the set of activities $\{a_1, \ldots, a_n\}$ and E the set of control edges (i.e., precedence relations) linking them.[b] More precisely, Activity Nets cover the following fundamental process patterns: Sequence, AND-split, AND-join, XOR-split, XOR-join, and Loop [60].[c] These patterns constitute the core set of any workflow specification language (e.g., WS-BPEL [4] and BPMN [5]) and cover most of the process models we can find in practice [75,33]. Furthermore, based on these patterns we are able to compose more complex ones if required (e.g., an OR-split can be mapped to AND- and XOR-splits [37]). Finally, when restricting process modeling to these basic process patterns, we obtain models that are better understandable and less erroneous [36,34]. A simple example of an Activity Net is depicted in Fig. 3a. For a detailed description of Activity Nets and relating correctness issues we refer to [41].

Block Structuring: To limit the scope, we assume Activity Nets to be block-structured, i.e., sequences, branchings (based on the aforementioned split and join patterns), and loops are represented as blocks with well-defined *start* and *end* nodes. These blocks may be nested, but must not overlap; i.e., the nesting must be regular [41,22]. In a process model S, a block may be a single activity, a self-contained part of S, or S itself. As example consider process model S from Fig. 3. Here {`A`}, {`A,B`}, {`C,F`}, and {`A,B,C,D,E,F,G`} describe possible blocks contained in

[b] An Activity Net contains more elements than node set N and edge set E, which can be factored out in the context of this paper.

[c] These patterns can be mapped to other languages as well. For example, in WS-BPEL (Business Process Execution Language), XOR-split / -join can be represented using 'If' or 'Pick'. Furthermore, AND-split / -join can be represented using 'Flow', and Loops using 'RepeatUntil' [4].

S. Note that we can represent a block B as activity set, since the block structure itself becomes clear from process model S. For example, block {A,B} corresponds to the parallel block with corresponding AND-split and AND-join nodes in S. The concept of block-structuring can be found in service composition languages like WS-BPEL and XLANG [4]. Furthermore, adaptive process management systems like AristaFlow BPM Suite [8] and CAKE2 [38] have emerged, which are applied in a variety of application domains and whose process modeling language is block-structured as well. When compared with non-block-structured process models, block-structured ones are easier understandable for users and have less chances of containing errors [45,34,35,36,7]. In a case study we conducted in another project, we investigated 214 process models expressed in different languages, like Event Process Chains, UML Activity Diagrams and WS-BPEL. More than 98% of these models were block-structured [57]. Finally, if a process model is not block-structured, in most practically relevant cases we can transform it into a block-structured one [66,36,22]. For all these reasons, we consider our approach for mining block-structured process variant models as being practically relevant.

Process change: A process change is accomplished by applying a sequence of high-level change operations to a given process model S over time [41]. Such operations structurally modify the initial process model by altering its set of activities and/or their order relations. Thus, each application of a change operation results in a new process model. We define *process change* and *process variant* as follows:

Definition 2.1. (Process Change and Process Variant) *Let $\mathcal{P}$ denote the set of possible process models and $\mathcal{C}$ be the set of possible process changes. Let $S, S' \in \mathcal{P}$ be two process models, let $\Delta \in \mathcal{C}$ be a process change, and let $\sigma = \langle \Delta_1, \Delta_2, \ldots \Delta_n \rangle \in \mathcal{C}^*$ be a sequence of changes performed on initial model S. Then:*

- *$S[\Delta\rangle S'$ iff Δ is applicable to S and S' is the (sound) process model resulting from the application of Δ to S.*
- *$S[\sigma\rangle S'$ iff $\exists\ S_1, S_2, \ldots S_{n+1} \in \mathcal{P}$ with $S = S_1$, $S' = S_{n+1}$, and $S_i[\Delta_i\rangle S_{i+1}$ for $i \in \{1, \ldots n\}$. We also denote S' as **process variant** of S.*

Examples of high-level change operations include *insert activity*, *delete activity*, and *move activity*, but also more complex adaptations like move process fragment (i.e., a whole block) or surround process fragment with a loop structure as implemented in the ADEPT change framework [41,8]. While *insert* and *delete* enable modifying the set of activities in a process model, *move* changes activity positions and thus the structure of the process model. A formal semantics of these change patterns can be found in [50]. For example, change operation $move(S, \texttt{A,B,C})$ moves activity A from its current position within process model S to the position after activity B and before activity C. Operation $delete(S, \texttt{A})$, in turn, deletes activity A from process model S. Finally, change operation $insert(S, \texttt{A,B,C})$ adds activity A to the position after activity B and before activity C. Issues concerning the correct use of these change operations, their generalization, and formal pre-/post-conditions are

described in [41]. Though the depicted change operations are discussed in relation to our ADEPT change framework, they are generic in the sense that they can be easily applied in connection with other process meta models as well [50,68]. For example, a process change as realized in the ADEPT framework can be mapped to the concept of life-cycle inheritance known from Petri Nets [58]. We refer to ADEPT since it covers by far most high-level change patterns and change support features when compared to other adaptive PAIS [68]. Furthermore, with AristaFlow BPM Suite [8], an industrial-strength version of the ADEPT technology emerged, which has been already applied in a variety of application domains.[d]

Based on the given set of change operations, we define the notions of *distance* and *bias* as follows:

Definition 2.2. (Bias and Distance) *Let S, $S' \in \mathcal{P}$ be two process models. Then:* ***Distance*** *$d_{(S,S')}$ between S and S' corresponds to the minimal number of high-level change operations needed to transform S into S'; i.e., we define*

$$d_{(S,S')} = \min\{|\sigma| \mid \sigma \in \mathcal{C}^* \wedge S[\sigma\rangle S'\} \quad (1)$$

Furthermore, a sequence of change operations σ with $S[\sigma\rangle S'$ and $|\sigma| = d_{(S,S')}$ is denoted as ***bias*** *between S and S'.*

The *distance* between process models S and S' corresponds to the minimal number of high-level change operations needed for transforming S into S'. The corresponding sequence of change operations is denoted as *bias* $B_{S,S'}$ between S and S'.[e] Usually, such distance measures the complexity for model transformation (i.e., model configuration). As example consider Example 1 in Fig. 2. Here, distance between reference process model S and process variant S_1 is *one*, since we only need to perform change operation $move(S, \texttt{B}, \texttt{A}, \texttt{C})$ to transform S into S_1 [26]. In general, determining bias and distance between two process models has complexity at *$\mathcal{NP}$-hard* level [26,59]. We consider high-level change operations instead of change primitives (i.e., elementary changes like adding or removing nodes or edges in a process graph) to measure the distance between process models. Amongst others, this helps us to guarantee soundness of process models and further provides a more meaningful measure for distance [26,68].

Finally, we define the notion of trace:

Definition 2.3. (Trace) *Let $S = (N, E, \ldots) \in \mathcal{P}$ be a process model. We define t as a trace of S iff:*

- *$t \equiv < a_1, a_2, \ldots, a_k >$ (with $a_i \in N$) constitutes a valid and complete execution sequence of activities considering the control flow defined by S. We*

[d] Visit www.aristaflow-forum.de for details.

[e] Generally, it is possible to have more than one minimal set of change operations to transform S into S', i.e., given process models S and S' their bias does not need to be unique. A detailed discussion of this issue can be found in [58,26].

define $\mathcal{T}_S$ *as the set of all traces that can be produced by process instances running on process model* S.

- $t(a \prec b)$ *is denoted as precedence relationship between activities* a *and* b *in trace* $t \equiv < a_1, a_2, \ldots, a_k >$ *iff* $\exists i < j : a_i = a \wedge a_j = b$.

We only consider traces composing 'real' activities, but no events related to *silent ones*, i.e., nodes within a process model having no associated action and only existing for control flow purpose [26]. Fig. 4 depicts some examples. At this stage, we consider two process models as being the same if they are *trace equivalent*, i.e., $S \equiv S'$ iff $\mathcal{T}_S \equiv \mathcal{T}_{S'}$. Like in most process mining approaches, the stronger notion of bi-similarity [18] is not considered in our context.

3. Mining Process Variants: Goals and Issues

This section discusses the major goal in respect to the mining of process variants, namely to derive a *generic process model* from a collection of process variants. This shall be done in a way such that the existing variants (as well as future ones) can be efficiently configured out of the discovered generic model. We measure efforts for corresponding process configurations in terms of the number of high-level change operations needed to transform the discovered generic model into the respective model variant. The challenge is to find a generic model such that the *average* number of high-level change operations needed (i.e., the *average distance*) becomes minimal with respect to the given variant collection.

To make this more clear, we first compare process variant mining with traditional process mining.[61] Process mining has been extensively studied in literature. Its key idea is to discover a process model by analyzing the execution behavior of (completed) process instances as captured in execution logs.[61] Different mining techniques like alpha algorithm [63], heuristics mining [71] or genetic mining [10] have been proposed in this context. Obviously, input data for traditional process mining differs from the one for process variant mining. While traditional process mining operates on *execution logs*, mining of process variants is based on a collection of process model variants. Of course, such high-level consideration is insufficient to prove that existing mining techniques do not provide optimal results with respect to the aforementioned goal. In principle, existing process mining techniques [63,10] can be applied to our problem as well. For example, we could derive all traces producible by a given collection of process variants [73] and then apply existing mining algorithms to them. To make the difference between process and process variant mining more evident, in the following we consider *behavioral similarity* between two process models as well as *structural similarity* based on their bias (cf. Def. 2.2).

The behavior of a process model S can be represented by the set of traces (i.e., $\mathcal{T}_S$) it can produce. Therefore, two process models can be compared based on the difference between their trace sets.[63,73] By contrast, biases can be used to express the (structural) distance between two process models [26], i.e., the minimal number of high-level change operations needed to transform one model into the other (cf. Def.

2.2). While the mining of process variants addresses structural similarity, traditional process mining focuses on behavior. Obviously, this leads to different choices with respect to the design of mining algorithms and also suggests different mining results.

Fig. 2 depicts two very simple examples. First, consider Example 1 which shows two process variants S_1 and S_2. Assume that 55 process instances are running on S_1 and 45 instances on S_2. We want to derive a generic process model such that the efforts for configuring the 100 process instances out of the generic model become minimal. If we focus on behavior, like existing process mining algorithms do [63], the discovered process model will be S; all traces producible on S_1 and S_2, respectively, can be produced on S as well, i.e. $\mathcal{T}_{S_1} \subseteq \mathcal{T}_S$ and $\mathcal{T}_{S_2} \subseteq \mathcal{T}_S$. However, if we adopt S as reference model and relink process instances to it, all instances running on S_1 or S_2 will have a non-empty bias. More precisely, we would need to move `B` in S to either obtain S_1 or S_2; i.e., $S[\sigma_1\rangle S_1$ with $\sigma_1 = move(S, \texttt{B,A,C})$ and $S[\sigma_2\rangle S_2$ with $\sigma_2 = move(S, \texttt{B,C,D})$ (cf. Def. 2.2). Using the number of instances as weight for each variant, average weighted distance between S and S_i $(i = 1, 2)$ is *one*; i.e., for each process instance we need on average one high-level change operation to configure S into S_1 and S_2 respectively.

By contrast, if we focus on biases, we should choose S' as reference model. While no adaptations become necessary for the 55 instances running on S_1, we need to move `B` for the 45 instances based on S_2, i.e. $S'[\sigma'\rangle S_2$ with $\sigma' = move(S', \texttt{B,C,D})$. Therefore, average weighted distance between S' and variants S_i $(i = 1, 2)$ corresponds to *0.45*. Though S' does not cover all traces variants S_1 and S_2 can produce (i.e., $\mathcal{T}_{S_2} \nsubseteq \mathcal{T}_{S'}$), adapting S' rather than S as the new generic model requires per average less efforts for process configuration, since average weighted distance between S' and the instances running on both S_1 and S_2 is 55% lower than when using S.

Regarding Example 2 from the bottom of Fig. 2, activity `X` is only present in S_2, but not in S_1. When applying traditional process mining, we obtain model S (with X being contained in a conditional branch). If we want to minimize average change distance, in turn, we need to choose S' as reference model. Note that we only consider very simple process models in Fig. 2 to illustrate basic ideas. As we show in the following, our approach works for process models with more complex structure (e.g., AND- XOR-branching and Loops) as well.

Our discussions on the difference between behavioral and structural similarity also demonstrate that current process mining algorithms do not consider structural similarity based on bias and change distance. (We quantitatively compare our mining approach with existing algorithms in Section 8.) First, a fundamental requirement for traditional process mining concerns the availability of a critical number of instance traces. An alternative method is to enumerate all the traces the process variants can produce (if it is finite) to represent the process model, and to use these traces as input source (i.e., logs) for traditional process mining algorithms. Unfortunately, this does also not satisfy our need to minimize average

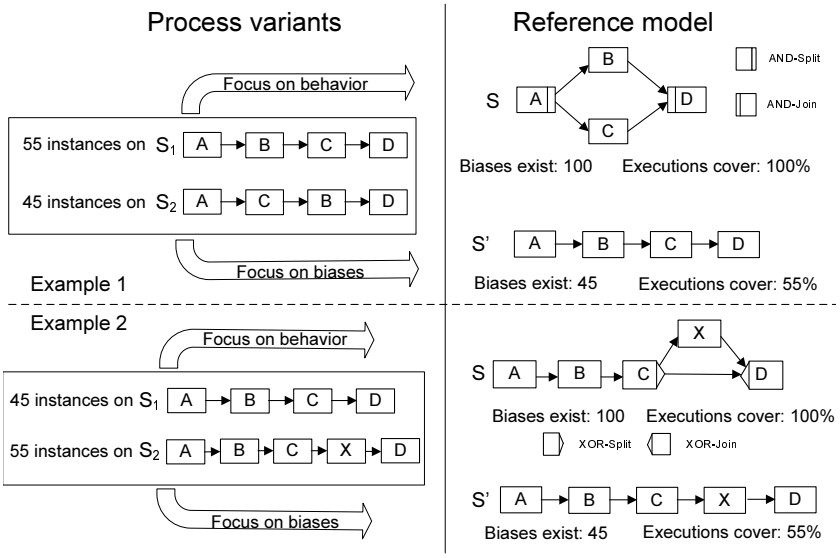


Fig. 2. Mining focusing either on Behavior or on Minimization of Biases

distances since it focuses on covering behavior as captured in execution logs (see Examples 1 and 2). Clearly, enumerating all the traces would be also a tedious and expensive task. For example, if a parallel branching block contains five branches and each branch contains five activities, the number of traces for such structure will be $(5 \times 5)!/(5!)^5 = 623360743125120$.

4. Representing Block-structured Processes as Order Matrices

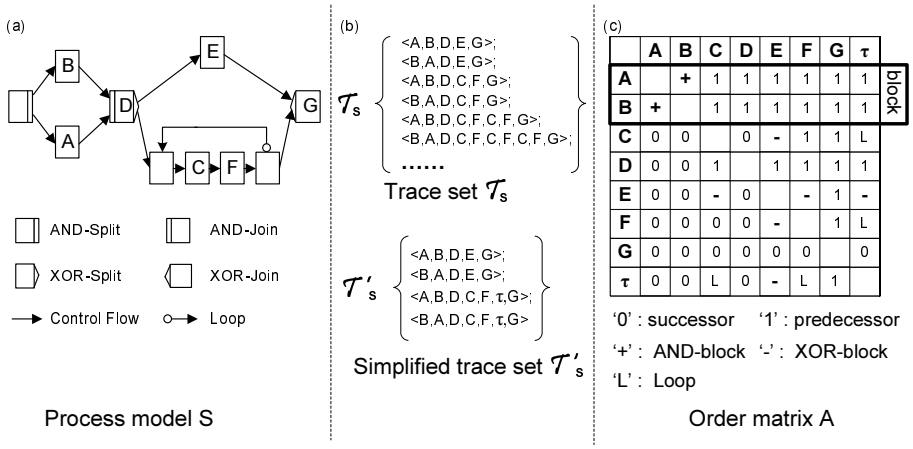
One key feature of our ADEPT change framework is to maintain the structure of the unchanged parts of a process model [41,8,70]. For example, if we delete an activity from a process model, the remaining process model will still be valid and the order relations (e.g., predecessor or successor) between other activities will remain the same [50,44]. To incorporate this feature in our approach, rather than only looking at direct predecessor-successor relationships between activities (i.e. control edges), we consider the transitive control dependencies for each pair of activities; i.e., for a given process model $S = (N, E, \ldots) \in \mathcal{P}$, for activities $a_i, a_j \in N$, $a_i \neq a_j$ we examine their structural order relations (including transitive one). Logically, we determine order relations by considering all traces in trace set $\mathcal{T}_S$ producible on model S (cf. Def. 2.3).

Fig. 3a shows an example of a process model S. This model is based on the patterns Sequence, AND-block, XOR-block, and Loop-block [60]. Here, trace set $\mathcal{T}_S$ of S constitutes an infinite set due to the presence of the loop-block in S (cf. Fig. 3b). Such infinite number of traces precludes us to perform any detailed analysis of the trace set. Therefore we need to transform it into a finite representation before conducting further analysis.

4.1. *Simplification of Infinite Trace Sets*

One common approach to describe a string with infinite length is to represent it as finite set of n-gram lists [6]. General idea behind an n-gram list is to represent a single string by an ordered list of substrings with length n (so-called *n-grams*). In particular, only the first occurrence of an n-gram is considered, while later occurrences of same n-gram are omitted in the n-gram list. Thus, an n-gram list represents a collection of strings with different length. In particular, an infinite language can be represented as finite set of n-gram lists. For example, a string $< abababab >$ can be represented as 2-gram $< \$a, ab, ba, b\# >$, where \$ (#) represents the start (end) of the string. Such approach is commonly used for analyzing loop structures in process models [73,3] or - more generally - in the context of text indexing for substring matching [2]. Inspired by the n-gram approach, we define the notion of *Simplified Trace Set* as follows:

Definition 4.1. (Simplified Trace Set) *Let S be a process model and $\mathcal{T}_S$ denote the trace set producible on S. Let B_k, $k = (1, \ldots, K)$ be Loop-blocks in S, and $\mathcal{T}_{B_k}$ denote the set of traces producible by loop body B_k. Let further $(t_{B_k})^m$ be a sequence of $m \in \mathbb{N}$ traces $< t^1_{B_k}, t^2_{B_k}, \ldots, t^m_{B_k} >$ with $t^j_{B_k} \in \mathcal{T}_{B_k}$, $j \in \{1, \ldots, m\}$. We additionally define $(t_{B_k})^0 \equiv <>$ as empty sequence. If we only consider the activities corresponding to B_k, in any trace $t \in \mathcal{T}_S$ producible on S, t either has no entries [f] or must have structure $< t^*_{B_k}, (t_{B_k})^m >$, with $t^*_{B_k} \in \mathcal{T}_{B_k}$ representing the first loop iteration and $m \in \mathbb{N}_0$ being the number of additional iterations loop-block B_k is executed in trace t. We can simplify this structure by using $< t_{B_k}, \tau_k >$ instead, where τ_k refers to $(t_{B_k})^m$. When simplifying trace set $\mathcal{T}_S$ this way, we obtain a finite set of traces $\mathcal{T}'_S$ which we denote as* ***Simplified Trace Set*** *of process model S.*



	A	B	C	D	E	F	G	τ
A		+	1	1	1	1	1	1
B	+		1	1	1	1	1	1
C	0	0		0	-	1	1	L
D	0	0	1		1	1	1	1
E	0	0	-	0		-	1	-
F	0	0	0	0	-		1	L
G	0	0	0	0	0	0		0
τ	0	0	L	0	-	L	1	

Fig. 3. a) Process model, b) (simplified) Trace set, and c) related order matrix

[f] i.e., the loop-block B_k has not been executed at all.

In our simplified representation of a trace $t \in \mathcal{T}_S$, we only consider the first occurrence of trace $t^*_{B_k}$ producible by loop-block B_k, while omitting others that occur later within trace t. Instead, we represent such repetitive entries by a silent activity τ_k, which has no associated action, but solely exists to indicate omission of other t_{B_k} appearing later in trace t; i.e., τ_k represents the iterative execution of loop-block B_k as captured in trace t.[g] When omitting repetitive entries within trace set $\mathcal{T}_S$, we obtain a finite trace set $\mathcal{T}'_S$ that we can use for further analysis. Note that when dealing with nested loops (e.g., a loop-block B_k contains another loop-block B_j), we first need to analyze B_j and then B_k; i.e., we need to first define τ_j to represent the iterative execution of loop-block B_j as captured in trace t and then we define τ_k to represent loop-block B_k.

As example consider process model S in Fig. 3a. Loop-block $B = \{\texttt{C,F}\}$, which is surrounded by a loop-backward edge, constitutes the block that comprises activities `C` and `F`. Consequently, the trace set this block can produce corresponds to $\{<\texttt{C,F}>\}$. Therefore, when only considering activities `C` and `F`, any trace $t \in \mathcal{T}_S$ producible on S has structure $<\texttt{C,F},(\texttt{C,F})^m>$ with $m \in \mathbb{N}_0$ depending on the number of times the loop iterates. For example, $<\texttt{C,F}>$, $<\texttt{C,F,C,F}>$ and $<\texttt{C,F,C,F,C,F}>$ are all valid traces producible by the loop-block. Let us define a silent activity τ corresponding to block B. Then we can simplify these traces by $<\texttt{C,F},\tau>$ where τ refers the to the sequence of the traces producible on B. As illustrated in Fig. 3b, we can simplify infinite trace set $\mathcal{T}_S$ to finite set $\mathcal{T}'_S = \{<\texttt{A,B,D,E,G}>,<\texttt{B,A,D,E,G}>,<\texttt{A,B,D,C,F},\tau,\texttt{G}>,<\texttt{B,A,D,C,F},\tau,\texttt{G}>\}$.

4.2. *Representing Process Models as Order Matrices*

For process model S, the analysis results concerning its trace set $\mathcal{T}_S$ are aggregated in an order matrix A, which considers five types of order relations (cf. Def. 4.2):

Definition 4.2. (Order matrix) *Let $S = (N, E, \ldots) \in \mathcal{P}$ be a process model with activity set $N = \{a_1, a_2, \ldots, a_n\}$. Let further $\mathcal{T}_S$ denote the set of all traces producible on S and let $\mathcal{T}'_S$ be the simplified trace set of S according to Def. 4.1. Finally let B_k, $k = (1, \ldots, K)$ denote loop-blocks in S and for every B_k, let τ_k, $k = (1, \ldots, K)$ be a silent activity representing the iterative structure producible by B_k in $\mathcal{T}'_S$. Then:*

*A is called **order matrix** of S with $A_{a_i a_j}$ representing the order relation between activities a_i,$a_j \in N \bigcup \{\tau_k | k = 1, \ldots, K\}$, $i \neq j$ iff:*

- *$A_{a_i a_j}$ = '1' iff $(\forall t \in \mathcal{T}'_S$ with $a_i, a_j \in t \Rightarrow t(a_i \prec a_j))$*
  If for all producible traces containing activities a_i and a_j, a_i always appears BEFORE *a_j, we set $A_{a_i a_j}$ to '**1**', i.e., a_i always precedes a_j in the*

[g] Though this approach has been inspired by n-gram, it is somewhat different from n-gram representation of a string. In n-gram the length of the sub-string is a fixed number n, while in our approach we use τ_k to represent traces producible by loop-block B_k. Obviously, traces producible by B_k do not need to have same length.

flow of control.

- $A_{a_i a_j}$ = *'0' iff* $(\forall t \in \mathcal{T}'_{\mathcal{S}}$ *with* $a_i, a_j \in t \Rightarrow t(a_j \prec a_i))$
  If for all producible traces containing activities a_i *and* a_j*,* a_i *always appears* AFTER a_j*, we set* $A_{a_i a_j}$ *to* ***'0'****, i.e.* a_i *always succeeds* a_j *in the flow of control.*
- $A_{a_i a_j}$ = *'+' iff* $(\exists t_1 \in \mathcal{T}'_{\mathcal{S}}$ *with* $a_i, a_j \in t_1 \wedge t_1(a_i \prec a_j)) \wedge$ $(\exists t_2 \in \mathcal{T}'_{\mathcal{S}}$ *with* $a_i, a_j \in t_2 \wedge t_2(a_j \prec a_i))$
  If there exists at least one producible trace in which a_i *appears before* a_j *and another one in which* a_i *appears after* a_j*, we set* $A_{a_i a_j}$ *to* ***'+'****; i.e.,* a_i *and* a_j *are contained in different parallel branches.*
- $A_{a_i a_j}$ = *'-' iff* $(\neg\exists t \in \mathcal{T}'_{\mathcal{S}} : a_i \in t \wedge a_j \in t)$
  If there is no producible trace containing both activity a_i *and* a_j*, we set* $A_{a_i a_j}$ *to* ***'-'****, i.e.* a_i *and* a_j *are contained in different branches of a conditional branching.*
- $A_{a_i a_j}$ = *'L', iff* $((a_i \in B_k \wedge a_j = \tau_k) \vee (a_j \in B_k \wedge a_i = \tau_k))$
  For any activity a_i *in a Loop-block* B_k*, we define order relation* $A_{a_i \tau_k}$ *between it and the corresponding silent activity* τ_k *as* ***'L'****.*

The first four order relations {1,0,+,-} specify the precedence relations between activities as captured in the trace set, while the last order relation 'L' indicates loop structures within the trace set. As example consider Fig. 3c which depicts the order matrix of process model S. Since S contains one loop-block, a silent activity τ is added to this order matrix as well. Note that the order matrix contains all five order relations as described in Def. 4.2. For example, activities `E` and `C` will never appear in same trace of the simplified trace set, since they are contained in different branches of an XOR block. Therefore, we assign '-' to matrix element $A_{\mathtt{EC}}$. Further, since in all producible traces, which contain both activity `B` and activity `G`, `B` always appears before `G`, we obtain order relations $A_{\mathtt{BG}}$ = '1' and $A_{\mathtt{GB}}$ = '0' respectively. Special attention should be paid to the order relations between silent activity τ and the other activities. The order relation between τ and activities `C` and `F` is set to 'L', since both `C` and `F` are contained in the loop-block; with all remaining activities τ has same order relations as `C` (or `F`) have. Note that the main diagonal of an order matrix is empty since we do not compare an activity with itself.

Generally, it is not a good idea to first enumerate all traces of a process model and then to analyze the order relations captured by them. Note that the trace set of a process model can become extremely large, particularly if the model contains multiple AND-blocks or even infinite at the presence of loop-blocks. In [29], we have introduced two algorithms for transforming a block-structured process model into its corresponding order matrix and vice verse. Complexity of these two algorithms is $\mathcal{O}(2n^2)$, where n equals the number of activities plus the number of loop-blocks contained in the process model; we have further proven that an order matrix constitutes a unique representation of a block-structured process model; i.e., if we transform a process model into an order matrix and then transform the latter back into a

process model, the two process models are *trace equivalent*; i.e., they cover same behavior [18].

Based on an order matrix representation, we can easily identify activities belonging to the same block. In particular, such activities have the same order relations with respect to activities from outside this block. As example, take the order matrix depicted in Fig. 3. If we ignore the internal relation between activities `A` and `B`, the order relations between `A` and all other activities are the same as for `B` (as marked up in Fig. 3 where the first two rows are identical when ignoring the order relation between `A` and `B`). Based on the order matrix, we can determine a process block containing `A` and `B`. Furthermore, these activities are contained in different branches of an XOR-block (as indicated by A_{AB} = '-').

5. Hospital Case and Running Example

To illustrate our mining approach along a real-world example and to also validate it in this context, we first introduce a real-world case from one of the projects we conducted in the healthcare domain.

5.1. *Case Study Description*

Context. We conduct a case study in a large clinical centre (with more than 1000 beds) in Germany. In this clinical centre the diagnostic and therapeutic processes of a patient usually involve various, organizationally more or less autonomous units. For a patient treated in a department of internal medicine or surgery, for example, tests and procedures at the laboratory and the radiology department have to be ordered. In this context, medical procedures must be planned and prepared, and appointments be made. Further, specimen or the patient himself have to be transported, physicians from other units may need to come and see the patient, and medical reports have to be written, sent, and evaluated. Thus, the cooperation between organizational units as well as the medical personnel is a vital task, with repetitive, but non-trivial character.

Data Source. We analyze several process model repositories of this clinical centre. In total, we can identify more than 90 process variants for handling medical orders and procedures respectively (e.g., X-ray inspections, cardiological examinations). Despite their similarity the different variants are captured in separate process models based on different notations (e.g., Event-driven Process Chains and UML Activity Diagrams) and modeling components (e.g., ARIS Architect, MQSeries Workflow, ADEPT). All models use standard process patterns like Sequence, AND-/XOR-Splits, AND-/XOR-Joins, and Loop, and their size ranges from 7 to 17 activities. Interestingly, for each non-block-structured variant model it is possible to transform it into a trace equivalent, block-structured representation; i.e., it is possible to map the different variant models to a representation following our process meta model. In this context, we apply simple refactorings (e.g., relabeling of activities) in order to harmonize considered variant models [67] .

Sources of Variance. Despite the structural similarity of the variant models, the latter also comprise parts only relevant for a sub-collection of the variants. For example, some of the variants require approval of a medical order by a senior physician, while this is not required in the context of other variants. Similarly, there exist medical procedures requiring complex scheduling activities, whereas in other cases no scheduling is required or the patient simply needs to be registered at the site of the care provider. Depending on the physical condition of the treated patient, in addition, a transport may have to be organized or not. Similarly, in emergency cases a short medical report is transmitted immediately after the medical examination to the requesting unit (e.g., a ward). Other variations of the analyzed models concern the preparation phases at the site of wards and examination units respectively.

We consider the most relevant 84 process variants which make up more than 95% of the identified ones. Based on the number of corresponding process instances, we assign weights to the variant models ranging from 0.1% to 8.67%. However, none of the process variants is dominant or significantly more relevant than others.

5.2. *Illustrative Example*

Due to space limitations, we cannot show all 84 process variants. Fig. 4 depicts six process variants $S_i \in \mathcal{P}$ $(i = 1, 2, \ldots 6)$ from our hospital case study (to ease presentation and later discussion we assign to each labeled activity a letter ranging from `A` to `Q`). Furthermore we assign weights to these six variant models according to their relevance. In the context of our work, we define the **weight** w_i of a process variant S_i as the percentage of process instances executed on basis of S_i. In our example, 20% of instances were executed based on S_1 and 5% of instances on S_3. If we only know process variants, but have no runtime information about related instance executions, we will assume the variants to be equally weighted; i.e., every process variant then will have weight $1/n$, where n corresponds to the number of variants in the system.

For our following considerations, first of all, we focus on these six variants which are divided into two parts: Part2 consists of activities `J,K,P,Q,L,M` and `O`. These activities exist in all six process variants S_1 - S_6, but show different order relations in these variants. On the contrary, Part1 consists of activities that do not appear in all process variants. For example, activity `E` exists only in S_1, S_2 and S_5, but not in S_3, S_4 and S_6.

In Section 6, we first assume that all process variants have same activity sets, i.e., we first consider solely Part2 of each process variants. In Section 7, we relax this constraint by also considering Part1 of the process variants. Finally, Section 7.5 summarizes mining results when applying our clustering algorithm to all 84 process variants from our healthcare case study.

6. Clustering Approach for Discovering Reference Process Models

We now present a clustering-based algorithm for mining a collection of process variants. Our goal is to derive a new reference model out of a given collection of

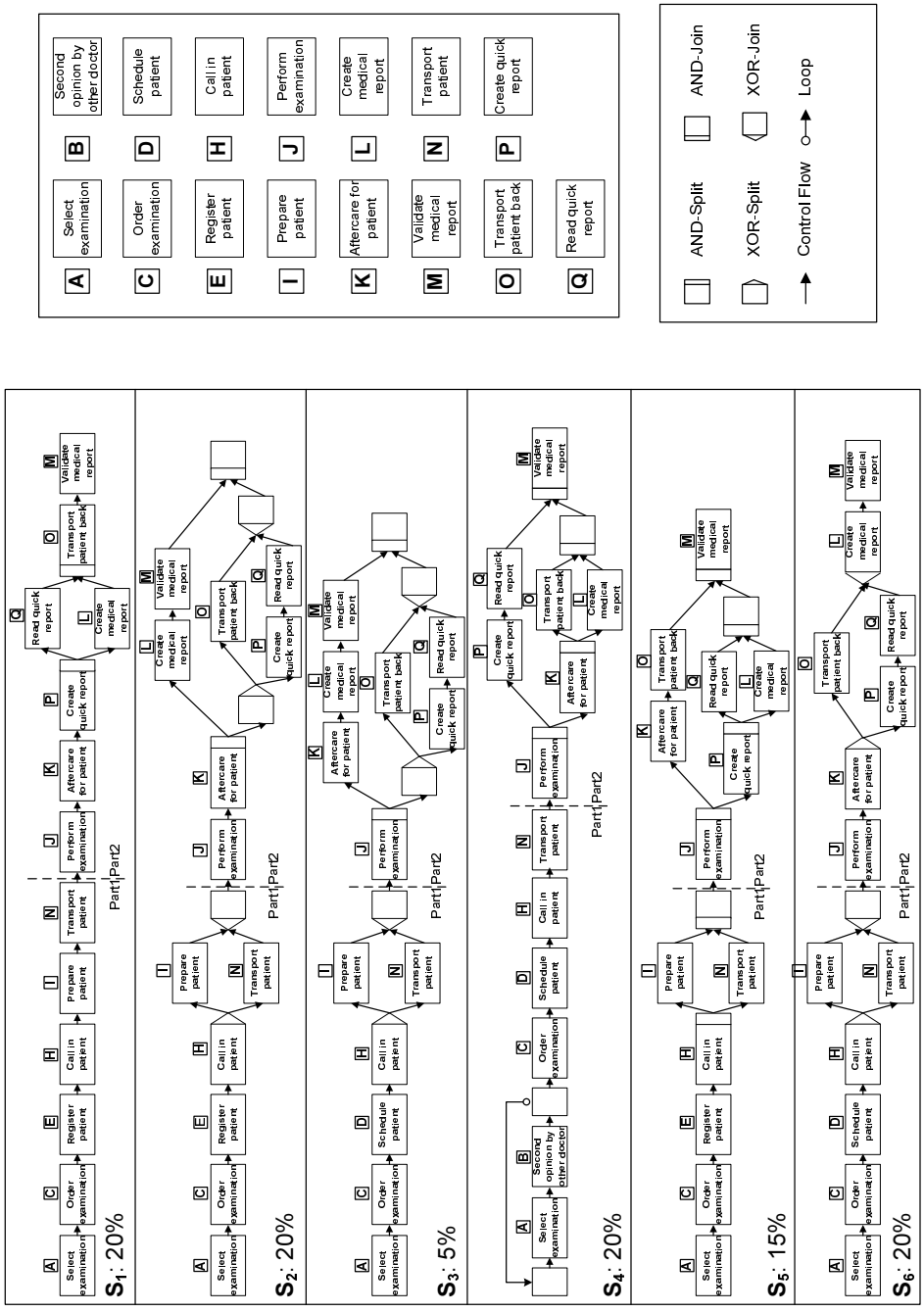


Fig. 4. Illustrative example

process variants which is easier configurable than the current one. Since we restrict ourselves to block-structured process models, we can build the new reference model by enlarging blocks, i.e., we first identify two activities that can form a block, then we merge this block with other activities and blocks respectively to form a larger block, and so forth. This procedure continues until all activities and blocks respectively are merged into one single block. This block and its internal structure then represent the new reference process model we are looking for.

Basically, our clustering approach for mining process variants works as follows:

(1) For all process variants calculate their order matrices. Aggregate them to one high-dimensional matrix representing all variants (cf. Section 6.1).
(2) Use this high-dimensional matrix and determine activities to be clustered in a block (cf. Section 6.2).
(3) Determine the order relation the clustered activities shall have within this block (cf. Section 6.3).
(4) After building a new block in Steps 2 and 3, reflect on the clustering of the activities by adjusting the high-dimensional matrix accordingly (cf. Section 6.4).
(5) Repeat Steps 2, 3 and 4 until all activities are clustered together; i.e., until the new process model has been constructed by the enlargement of blocks.

6.1. *Aggregated Order Matrix*

For each variant of the given collection of process variants, we first compute its order matrix (cf. Def. 4.2). Regarding our example from Fig. 4, we need to determine six order matrices (cf. Fig. 5). Afterwards, we analyze the order relation for each pair of activities considering all order matrices derived before. As the order relation between two activities might be not always the same in all order matrices, this analysis does not result in a fixed relationship, but provides a distribution for the five types of order relations (cf. Def. 4.2). Regarding our example, for instance, in 20% of all cases activity `O` is a successor of activity `Q` (as in S_1), in 15% of all cases `O` and `Q` are contained in different branches of an AND block (as in S_4 and S_5), and in 45% of all cases in different branches of an XOR block (as in S_2, S_3 and S_6) (cf. Fig. 5). Generally, we can define the order relation between two activities `a` and `b` as 5-dimensional vector $V_{ab} = (v^0_{ab}, v^1_{ab}, v^+_{ab}, v^-_{ab}, v^L_{ab})$. Each field then corresponds to the frequency of the respective relation type ('0', '1', '+', '-' or 'L') as specified in Def. 4.2.

Take again our running example and consider Fig. 5. Here, $v^0_{\mathtt{OQ}}$ corresponds to the frequency of all cases with activities `O` and `Q` having order relationship '0', i.e., all cases for which `O` succeeds `Q`; we obtain $V_{OQ} = (0.2, 0, 0.35, 0.45, 0)$. Formally, we define an *aggregated order matrix* as follows:

Definition 6.1. (Aggregated Order Matrix) *Let $S_i = (N_i, E_i, \ldots) \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants with activity sets N_i. Let further A_i be the order matrix of S_i, and let w_i represent the number of process instances*

being executed based on S_i. The ***Aggregated Order Matrix*** *of all process variants is defined as 2-dimensional matrix $V_{m\times m}$ with $m = |\bigcup N_i|$ and each matrix element $v_{a_j a_k} = (v^0_{a_j a_k}, v^1_{a_j a_k}, v^+_{a_j a_k}, v^-_{a_j a_k}, v^L_{a_j a_k})$ being a 5-dimensional vector. For $\Diamond \in \{0, 1, +, -, L\}$, element $v^\Diamond_{a_j a_k}$ expresses to what percentage, activities a_j and a_k have order relation $\Diamond$ within the collection of process variants $S_1, \ldots, S_n$. Formally: $\forall a_j, a_k \in \bigcup N_i, a_j \neq a_k$:*

$$v^\Diamond_{a_j a_k} = \frac{\sum_{A_{i_{a_j a_k}} = '\Diamond'} w_i}{\sum_{a_j, a_k \in N_i} w_i}. \tag{2}$$

The aggregated order matrix V of the process variants from Fig. 4 is shown in Fig. 5. Generally, the main diagonal of an aggregated order matrix is always empty since we do not specify the order relation of an activity with itself. For all other elements, a non-filled value in a certain dimension means it corresponds to zero.

In Section 4 we have shown that we can use an order matrix to determine blocks in a process model: i.e., two activities can be clustered into a block if they have same order relation with respect to other activities. As we will show, similar idea can be applied when analyzing an aggregated order matrix. Our goal is to derive an optimal reference process model for the given variants based on this representation form.

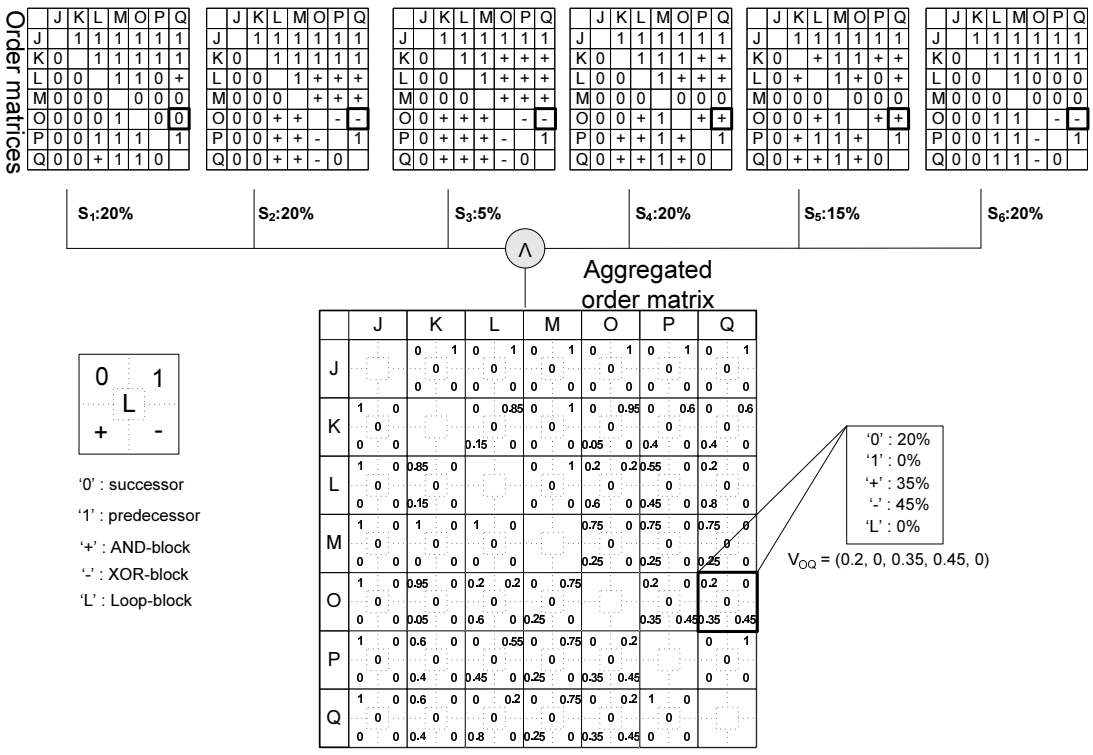


Fig. 5. Aggregated order matrix V

6.2. *Determining Activities to be Clustered*

This subsection describes how we derive the blocks for the reference model to be discovered from an aggregated order matrix, i.e., from a collection of process variants. There are two fundamental issues we have to consider in this context. First, we have to decide which activities (and blocks respectively) shall be "blocked". Second, we must choose an order relation for them. This subsection deals with the first issue, the second one is addressed in Section 6.3.

Regarding an order matrix, two activities can be clustered in a block if they have same order relations with respect to the other activities (cf. Section 4). We can apply similar idea when analyzing an aggregated order matrix. However, in an aggregated order matrix the relationship between two activities is expressed as 5-dimensional vector showing the distribution of the order relations over all process variants. When determining pairs of activities that can be clustered in a block, it would be too restrictive to require precise matching as in the case of an order matrix. To deal with this, we first introduce function $f(\alpha, \beta)$ which expresses the closeness between two vectors $\alpha = (x_1, x_2, ..., x_n)$ and $\beta = (y_1, y_2, ..., y_n)$:

$$f(\alpha, \beta) = \frac{\alpha \cdot \beta}{|\alpha| \times |\beta|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{3}$$

$f(\alpha, \beta) \in [0, 1]$ computes the cosine value of the angle θ between vectors α and β in Euclid space. If $f(\alpha, \beta) = 1$ holds, α and β exactly match in their directions; $f(\alpha, \beta) = 0$ means, they do not match at all. For example, when comparing closeness between $v_{\texttt{LP}} = (0.55, 0, 0.45, 0, 0)$ and $v_{\texttt{MP}} = (0.75, 0, 0.25, 0, 0)$, for example, we obtain $f(v_{\texttt{LP}}, v_{\texttt{MP}}) = 0.934$. This high value implies that the two vectors are close to each other though they are not the same.

Using $f(\alpha, \beta)$ we introduce **Separation** metrics. It indicates to what degree two activities of an aggregated order matrix are suited for being clustered in a block. More precisely, $Separation(a, b)$ expresses how similar order relations of activities a and b are when compared to the other activities. In our example from Fig. 4, $Separation(\texttt{J,K})$ is determined by the closeness (measured in terms of the cosine value) of $f(v_{\texttt{JL}}, v_{\texttt{KL}})$, $f(v_{\texttt{JM}}, v_{\texttt{KM}})$, $f(v_{\texttt{JO}}, v_{\texttt{KO}})$, $f(v_{\texttt{LP}}, v_{\texttt{KP}})$, and $f(v_{\texttt{JQ}}, v_{\texttt{KQ}})$. Generally, we define cluster separation as follows:

$$Separation(a, b) = \frac{\sum_{x \in N \setminus \{a,b\}} f^2(v_{ax}, v_{bx})}{|N| - 2} \tag{4}$$

N corresponds to the set of activities. Like most clustering algorithms [56], we square the cosine value to emphasize the differences between the two compared vectors. Finally, dividing this expression by $|N| - 2$ normalizes its value to a range between [0, 1]. Regarding our example from Fig. 4, we obtain $Separation(\texttt{J,K}) = 0.870$. This high separation value indicates that activities `J` and `K` have a relatively high similarity regarding their order relations to the remaining activities, i.e., they have relatively high chance of forming a block.

We determine the pair of activities best suited to form a block by measuring how much each activity pair is separated from the other activities. We accomplish this by computing the separation value for each activity pair. The higher this value is, the better the two activities are suited for being clustered. Fig. 6 depicts the separation values for our running example from Fig. 4. We denote this table as *separation table*. Obviously, in our example activities `P` and `Q` have the highest separation value of 0.929 (marked up in grey color in Fig. 6). We therefore choose `P` and `Q` as the activities forming our first block.

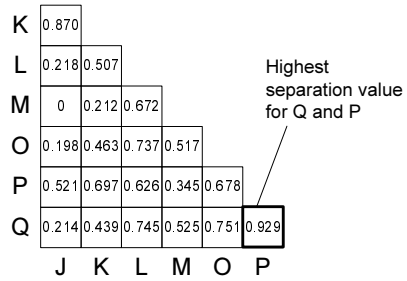


Fig. 6. Separation table of aggregated order matrix

6.3. *Determining Internal Order Relations*

After having decided that activities `P` and `Q` are clustered in the first block, we have to determine the order relation these two activities shall have. In addition, we measure how good our choice is. For this purpose, we introduce **Cohesion** as measure which indicates how significant particular order relations between two activities of the same cluster are.

In the aggregated order matrix of our running example the relationship between activities `P` and `Q` is depicted as 5-dimensional vector $v_{\mathtt{PQ}} = (0, 1, 0, 0, 0)$. It shows the distribution values of the five types of order relations. Obviously, when building a reference process model, only one of the five order relations can be chosen. Therefore, we want to choose that order relation which is most significant. Regarding our example, the significance of each order relation can be evaluated by the closeness vector $v_{\mathtt{BC}}$ and the five axes in the 5-dimensional space have. These axes can be represented by five benchmarking vectors: $v^0 = (1, 0, 0, 0, 0)$, $v^1 = (0, 1, 0, 0, 0)$, $v^+ = (0, 0, 1, 0, 0)$, $v^- = (0, 0, 0, 1, 0)$, and $v^L = (0, 0, 0, 0, 1)$. Based on this, we can compute the significance of each order relation using $f(\alpha, \beta)$ (cf. Section 6.2), with $\alpha = v_{\mathtt{PQ}}$ and β being one of the five benchmarking vectors. Regarding our example, the closest axis to $v_{\mathtt{PQ}}$ is v^1 (with $f(v_{\mathtt{PQ}}, v^1) = 1$). Therefore, we decide that `P` shall precede `Q` within the newly derived block (cf. Def. 4.2).

We use Cohesion to evaluate how good our choice is:

$$Cohesion(a, b) = \frac{\max_{\Diamond=\{0,1,+,-,L\}}\{f(v_{ab}, v^{\Diamond})\} - 0.4472}{1 - 0.4472} \tag{5}$$

This measure has value range [0,1] as well. $Cohesion(a, b)$ equals *one* if there is a dominant order relation, i.e., v_{ab} is on one of the five axes. $Cohesion(a, b)$ equals

zero if $v_{ab} = (0.2, 0.2, 0.2, 0.2, 0.2)$ holds, i.e., no order relation is more significant than the others. Regarding our example, $Cohesion(\texttt{P,Q})$ equals 1. This indicates that activity P precedes activity Q in all variants.

6.4. *Recomputing the Aggregated Order Matrix*

We have discovered the first block of our reference process model, which contains P and Q. We have further decided that P shall precede Q, and that the significance of this order relation is 1. We now have to decide about the relationship between the newly created block (comprising P and Q) and the other activities.

Regarding the process variants from Fig. 4, activities P and Q do not always constitute an elementary block (i.e., a block only containing Q and P). To be more precise, P and Q can form a block in S_2, S_3, S_4 and S_6, but not in S_1 and S_5. Nevertheless, P and Q are best suited to form a block based on our analysis of the aggregated order matrix. This, in turn, requires an adaptation of the original aggregated order matrix in order to represent the situation in which P and Q are clustered in a block.[h] We accomplish this adaptation by computing the means of the order relations between {P, Q} and the remaining activities. For example, as $v_{\texttt{PL}} = (0, 0.55, 0.45, 0, 0)$ and $v_{\texttt{QL}} = (0, 0.2, 0.8, 0, 0)$ hold, the order relation between the newly created block (P,Q) and activity L corresponds to $(v_{\texttt{PL}}+v_{\texttt{QL}})/2 = (0, 0.375, 0.625, 0, 0)$.[i] Such computation is applied to all remaining activities outside this block.

Generally, after clustering activities a and b, the new aggregated order matrix V' can be calculated as follows:

$$\forall x \in N \setminus \{a, b\} : \begin{cases} v'_{(a,b)x} = (v_{ax} + v_{bx})/2 \\ v'_{x(a,b)} = (v_{xa} + v_{xb})/2 \end{cases} \tag{6}$$

$$\forall x, y \in N \setminus \{a, b\} : v'_{xy} = v_{xy} \tag{7}$$

The aggregated order matrix V' we obtain after clustering P and Q is shown in Fig. 7. Since B and C are replaced by a block containing these two activities, the matrix resulting after the re-computation is one dimension smaller than V. Afterwards, we treat this block like a single activity, but keep its internal structure in order to build up the new reference process model at the end.

6.5. *Mining Result for Part 2*

After obtaining the newly aggregated order matrix, we repeat the three steps as described in Sections 6.2, 6.3 and 6.4; i.e., we first identify the two activities (and

[h] Our approach is different to traditional clustering algorithms [56], in which only distances are re-computed, but not the original dataset.

[i] This approach is an unweighted one; i.e., we simple take the average of the two vectors without considering their importance; e.g., how many activities are included in the block. In this way, we can ensure that when merging two blocks of different sizes, the order relations of the resulting block are not too much dominated by the bigger one. Such unweighted approach is widely used from other clustering approaches [56].

	J	K	L	M	O	(P,Q)
J		0 1 0 0 0 0	0 1 0 0 0 0	0 1 0 0 0 0	0 1 0 0 0 0	0 1 0 0 0 0
K	1 0 0 0 0 0		0 0.85 0 0 0.15 0	0 1 0 0 0 0	0 0.95 0 0 0.05 0	0 0.6 0 0 0.4 0
L	1 0 0 0 0 0	0.85 0 0 0 0.15 0		0 1 0 0 0 0	0.2 0.2 0 0 0.6 0	0.375 0 0 0 0.625 0
M	1 0 0 0 0 0	1 0 0 0 0 0	1 0 0 0 0 0		0.75 0 0 0 0.25 0	0.75 0 0 0 0.25 0
O	1 0 0 0 0 0	0.95 0 0 0 0.05 0	0.2 0.2 0 0 0.6 0	0 0.75 0 0 0.25 0		0.2 0 0 0 0.35 0.45
(P, Q)	1 0 0 0 0 0	0.6 0 0 0 0.4 0	0 0.375 0 0 0.625 0	0 0.75 0 0 0.25 0	0 0.2 0 0 0.35 0.45	

Fig. 7. Aggregated order matrix V' resulting after the clustering of `B` and `C`

blocks respectively) to be clustered next, then we determine their order relation within the block, and finally we re-compute the aggregated order matrix considering the newly determined block. In every iteration, we merge two activities and blocks respectively into one bigger block. This iterative clustering continues until all activities from the original aggregated order matrix are clustered. Finally, we obtain our new reference process model. Obviously, the number of required iterations equals the number of activities minus one. Regarding our running example, Fig. 8 depicts the final result S'_{Part2} we obtain after running through all iterations of our clustering algorithm.

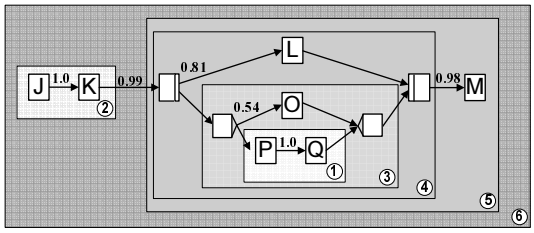


Fig. 8. The discovered process model S'_{Part2} based on part 2 of process variants

Fig. 8 does not only show process model S'_{Part2}, which we have discovered through the mining of the variants from Fig. 4, but also the intermediate results we obtain after every iteration (indicated through the number at the right-bottom corner of each block). In the first iteration, for example, `P` and `Q` are clustered to form a block. In the second iteration, this block is merged with activity `O` to form an XOR- block. Finally, after the sixth iteration, all activities are clustered together into one single block, which corresponds to the discovered reference model.

Fig. 8 additionally shows cohesion values, which reflect the significance of the order relations we haven chosen in different iterations. For example, in the second iteration the cohesion we obtain when clustering activity `O` and the block containing `P` and `Q` equals 0.54. Since cohesion reflects the significance of the chosen order relation, it also expresses local fitness of the control flow in the reference model. For

example, when comparing the cohesion values, it turns out to be of high significance that activity P precedes activity Q, but less significant that they can form an XOR-block with activity O. When reconsidering our process variants from Fig. 4, for example, we can draw similar conclusions as the ones described here.

7. Mining Process Variants with Different Activity Sets

So far, our basic method for mining process variants has assumed that all variant models comprise the same set of activities. However, process variants may differ in their activity sets in general. In this section we discuss how to mine process variants with different activity sets. We illustrate relevant issues as well as necessary extensions of our basic method by analyzing Part1 of the process variants shown in Fig. 4.

7.1. *Analyzing the Occurrences of Activities*

One fundamental challenge is to decide which activities shall be considered in the resulting reference model and which not. Another challenge is to fix the order relations between the considered activities, which is not trivial since not all activities occur in all variant models. Here we first define *Coexistence Matrix* as follows:

Definition 7.1. (Coexistence Matrix) *Let $S_i = (N_i, E_i, \ldots) \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants with activity sets N_i. Let further w_i represent the percentage of instances that were executed based on S_i. The **Coexistence Matrix** of process variants $S_1, \ldots, S_n$ is then defined as 2-dimensional matrix $E_{m \times m}$ with $m = |\bigcup N_i|$. Each matrix element $E_{a_j a_k}$ corresponds to the relative frequency with which activities a_j and a_k appear together within the given collection of variants. Formally: $\forall a_j, a_k \in \bigcup N_i$:*

$$E_{a_j a_k} = \sum_{S_i : a_j, a_k \in N_i} w_i \tag{8}$$

Coexistence matrix measures how often activity pairs occur in a collection of process variants. The main diagonal ($a_i = a_k$) measures how often a particular activity a_i occurs in the variants, and the remaining elements in the matrix measure how frequent two particular activities co-occur in the variants. For our example from Fig. 4, the Coexistence Matrix is shown in Table 1. Note that silent activity τ is included to represent the Loop structure in S_4 (cf. Def. 4.2).

Table 1 summarizes the relative frequencies of single activities (main diagonal) as well as co-occurring activity pairs (other matrix elements). For example, activities A,C,H and N occur in 100% of the process variants while activity E only appears in 55% of the variants (S_1, S_2 and S_5). In MinAdept, the user may set a threshold value in order to determine which activities shall be contained in the resulting reference process model and which not. This way we can exclude activities with low frequency if desired. For example, if we only want to consider activities with a

relative frequency greater than 40%, activity B as well as silent activity τ will be excluded from the reference process model (excluding τ means the loop structure will not be considered). Generally, process engineers have to set respective threshold values depending on whether they want to add more or fewer activities to the reference process model. Obviously, a good threshold value is the key to success. We discuss how to find a suitable threshold in Section 9.3.

7.2. *Coping with Unclear Order Relations*

Except the main diagonal, the other elements in the coexistence matrix show the relative frequencies with which activity pairs co-occur in the considered collection of process variants (and process instances respectively). For example, activities A and C co-occur in all instances, while A and E only co-occur in 55% of the instances (cf. Table 1). We can use these numbers when computing the aggregated order matrix as described in Section 6.1. However, since not all activities are present in all variants, the absolute frequencies of the different order relations of a particular activity have to be put in relation to the total number of process instances existing for the respective variants. For example, though A is a predecessor of B in only 20% of the process instances (only in S_4), this corresponds to 100% of all instances containing this activity pair; i.e., in all process instances containing both A and B, activity A always precedes activity B. Thus, the execution order between A and B should be represented by vector $V_{AB} = (0, 1, 0, 0, 0)$ and be reflected in the definition of the aggregated order matrix (cf. Def. 6.1) accordingly.

Following this approach does not counteract the basic mining method presented in Section 6. Note that we target at a (reference) process model which covers the *structural relations* between the considered activities best. In principle, it is also possible that an activity is considered in the reference model though it has only low frequency and therefore co-occurs with other activities in only few cases. Generally, if the relative frequency of an activity is higher than the specified threshold value, it will be considered in the reference model to be discovered. For this case, we are basically interested in the order relations of this activity with respect to the other activities considered in the reference model.

There may be pairs of activities which do not co-occur in any of the process variants and process instances respectively (i.e., their co-occurrence is 0), but which

	A	B	C	D	E	H	I	N	τ
A	1.0	0.2	1.0	0.45	0.55	1.0	0.8	1.0	0.2
B	0.2	0.2	0.2	0.2	0.0	0.2	0.0	0.2	0.2
C	1.0	0.2	1.0	0.45	0.55	1.0	0.8	1.0	0.2
D	0.45	0.2	0.45	0.45	0.0	0.45	0.25	0.45	0.2
E	0.55	0.0	0.55	0.0	0.55	0.55	0.55	0.55	0.0
H	1.0	0.2	1.0	0.45	0.55	1.0	0.8	1.0	0.2
I	0.8	0.0	0.8	0.25	0.55	0.8	0.8	0.8	0.0
N	1.0	0.2	1.0	0.45	0.55	1.0	0.8	1.0	0.2
τ	0.2	0.2	0.2	0.2	0.0	0.2	0.0	0.2	0.2

Table 1. Occurrence Matrix

shall be both included in the new reference process model. Since the two activities never co-occur, we are unable to derive relative frequencies for the five possible order relations; i.e., their order relations as reflected in the aggregated order matrix would be (0,0,0,0,0). Regarding our example, activities `D` and `E` do not co-occur in any process variant. Since $f(\alpha, \beta)$ (cf. Equation 3) will be invalid if one of the two vectors α or β equals $(0, 0, 0, 0, 0)$, we need to find ways to handle such unclear relationship:

(1) **Compute separation.** When computing separations (cf. Equation 4), we can ignore these unclear relationships. For example, when computing $Separation(a, b)$ between activity a and b, we can see that order relation v_{ax} is an unclear one, i.e., it equals $(0, 0, 0, 0, 0)$. We therefore do not include it in the computation.
To be more precise, let V be an aggregated order matrix and let $a, b \in N$ be two activities considered in V. We can define a set $\mathcal{M} = \{x \big| x \in N \setminus \{a, b\}, v_{ax} = (0, 0, 0, 0, 0) \vee v_{bx} = (0, 0, 0, 0, 0)\}$ which contains all x with v_{ax} or v_{bx} being (0,0,0,0,0). Then instead of using Equation 4, we use Equation 9 to compute $Separation(a, b)$ to ignore the influence of unclear relations.

$$Separation(a, b) = \frac{\sum_{x \in N \setminus \{a,b\}, x \notin \mathcal{M}} f^2(v_{ax}, v_{bx})}{|N| - 2 - |\mathcal{M}|} \tag{9}$$

(2) **Compute cohesion.** When computing $Cohesion(a, b)$ (cf. Equation 5), we will also run into problems if their order relation is (0,0,0,0,0). In this case we can define $cohesion(a, b) := 0$, i.e., none of the five order relations is considered being more significant than the others.

(3) **Recompute Aggregated Order Matrix.** Since an aggregated order matrix captures the distribution of the five order relations within the collection of variants, for each vector $v_{ab}, a, b \in N$, we obtain value 1 as the sum of its elements. However, this does not apply to the unclear order relations. Therefore, when re-computing the (reduced) aggregated order matrix after the creation of a new block (cf. Section 6.4), we do not take such unclear order relations into account.
To be more precise, let V be an aggregated order matrix and let $a, b, x \in N$ be three activities contained in V. Assume further that in one iteration of our algorithm, activities a and b are clustered into a block. Then we set new matrix element $v'_{(a,b)x} = v_{ax}$ if $v_{bx} = (0, 0, 0, 0, 0)$ holds or $v'_{(a,b)x} = v_{bx}$ if $v_{ax} = (0, 0, 0, 0, 0)$ holds, respectively.

7.3. *Mining Result for Part 1 of Our Running Example*

Regarding Part1 of our running example (i.e., Part1 of the process variants in Fig. 4), Fig. 9 shows the discovered process model S'_{Part1} as well as the intermediate results we obtain after every iteration of our clustering approach (indicated through

the numbers at the right bottom corner of each block). Note that the discovered model comprises all process activities in the variants (Part1). This includes silent activity τ, which represents the loop structure in S_4.

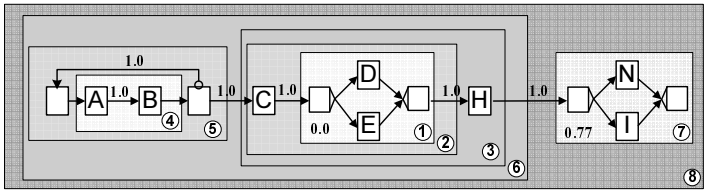


Fig. 9. The discovered process model S'_{Part1} based on part 1 of process variants

Special attention should be paid to the cohesion values between activities `D` and `E`. $Cohesion(D, E)$ equals 0, which means that the order relation between these two activities is unclear. This can be also observed from the coexistence matrix E for which $E_{DE} = 0$ holds. In principle, if the order relation is not clearly indicated within the collection of variants, any one of the four order relations {0,1,+,-} [j] can be applied. We set as default option '-' (XOR) in such case. We choose this option as default taking the behavior of the process variants into account: If two activities never show up together in a variant (i.e., their co-occurrence is 0), we can assume that they never co-occur for any process instance.

7.4. *Setting Different Threshold for Mining Reference Models*

Regarding our example from Fig. 4 we can now consider both parts of the process variants and apply our algorithm to discover a process model. Fig. 10 depicts the discovered models when setting different threshold values:

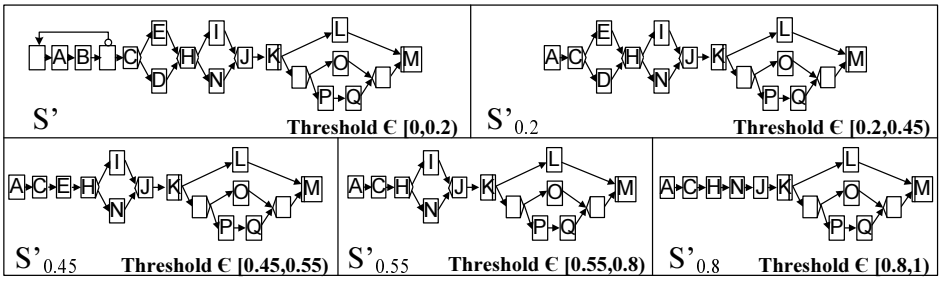


Fig. 10. Discovered process models when setting threshold at different levels

Regarding Fig. 10 process model S' contains all activities that have appeared in at least one of the process variants. Note that we obtain same model when merging S'_{Part1} (cf. Fig. 9) and S'_{Part2} (cf. Fig. 8). In fact, when discovering process model

[j]Two activities can have order relation 'L', if and only if one of the two activities is a silent one (cf. Def. 4.2).

S', we obtain same (sub-)block and cohesion value in each iteration as the ones we obtained when mining S'_{Part1} and S'_{Part2} separately, i.e., only the order of iterations differs.

Having a closer look at the process models we obtain when setting different threshold values, we can see that they are similar to each other as well. In particular, the structure of these models is the same except that their activity sets differ. This additionally indicates good performance of our algorithm: the structure of process models discovered by our algorithm is independent from the initial activity set considered in the model. In Section 9.3, we will discuss how to find a good threshold value in our context.

7.5. *Case Study Results*

We applied the presented clustering algorithm to the hospital case described in Section 5. All 84 process variants as well as their relative weights were considered. The threshold was set to 50%. Fig. 11 shows the discovered reference process model S' as well as the original reference process model S we found in the organizational handbook of the clinical centre.

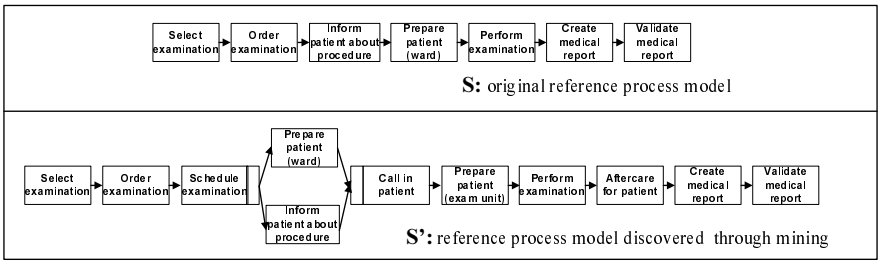


Fig. 11. Discovered reference process model of the healthcare case study

Average weighted distance between the original reference process model S and the process variants is 5.307, while average weighted distance between the discovered reference process model S' and the process variants is only 2.795. This indicate that by applying our mining algorithm, we can significantly reduce the configuration effort by 47.3%.[k] Note that the original reference process model S constitutes a very simple model since it only contains 7 activities that are organized in sequence. When discussing this reference model with process owners we further learned that its original purpose was to define the basic organizational steps of an arbitrary medical order and medical process, respectively, but that the different variant models have evolved over time and thus more or less differ from this reference model. We showed the newly discovered reference model S' to the process owners at the clinical

[k]When replacing old reference process model by a new one, we can propagate the changes on the reference process model to the process instances as well. This means that even the process instances, which were created and executed according to the old reference process model, can benefit from the replacement of the reference model. We refer to [48] for further details.

centre who confirmed that it is closer to the variants than the one from the original handbook.

8. Validation

The complexity of the mining algorithm described in Section 6 corresponds to $\mathcal{O}(n^2m + n^3)$, where n equals the number of activities each variant comprises and m equals the number of variants. $\mathcal{O}(n^2m)$ corresponds to the complexity needed to build the aggregated order matrix, while $\mathcal{O}(n^3)$ corresponds to the complexity needed to mine the reference model. Consequently, our algorithm has polynomial complexity.

8.1. *Average Weighted Distance*

As discussed in Section 1, the goal of our algorithm is to discover a process model which has minimal average distance to the variants. Therefore, we first need to define **average weighted distance** between a reference process model S and its variants.

Definition 8.1. (Average Weighted Distance) *Let $S = (N, E, \ldots) \in \mathcal{P}$ be a reference process model. Let further $\mathcal{M}$ be a set of process variants $S_i = (N_i, E_i, \ldots) \in \mathcal{P}$, $i = 1, \ldots, n$, with w_i representing the relative frequency of process instances that were executed on basis of S_i and $d(S, S_i)$ being the distance between S and S_i (cf. Def. 2.2).* ***Average Weighted Distance*** *$D_{(S,\mathcal{M})}$ between S and $\mathcal{M}$ can be computed as follows:*

$$D_{(S,\mathcal{M})} = \sum_{i=1}^{n} d_{(S,S_i)} \cdot w_i \tag{10}$$

The complexity to compute average weighted distance is $\mathcal{NP}$-*hard* since the complexity to compute the distance between two variants is $\mathcal{NP}$-*hard* (cf. Def. 2.2). For example, assume that we take the discovered process model S' (cf. Fig. 10) as new reference process model. Then distance between S' and each of the six process variants S_i (cf. Fig. 4) is as follows: $d(S', S_1) = 6$, $d(S', S_2) = 4$, $d(S', S_3) = 5$, $d(S', S_4) = 4$, $d(S', S_5) = 7$, and $d(S', S_6) = 4$. Taking variant weights into account as well (cf. Fig. 4), we obtain as average weighted distance

$$(6 \times 0.2 + 4 \times 0.2 + 5 \times 0.05 + 4 \times 0.2 + 7 \times 0.15 + 4 \times 0.2) = 4.9.$$

This means we need to perform on average 4.9 high-level change operations to configure a process variant (and related instance respectively) out of the reference process model. Generally, average weighted distance between a reference model and its process variants represents how "*close*" they are.

Since computing average weighted distance has $\mathcal{NP}$-*hard* complexity [26], our algorithm does not aim at finding the global optimum, i.e., the model which has minimal average weighted distance to the variants. However, this is a rather uncritical issue. Note that most clustering techniques and other data mining algorithms aim at finding a local optimum rather than a global one since it is almost impossible

to find the global optimum in reasonable time.[56] Our suggested clustering algorithm constitutes a heuristic approach which tries to solve a complex combinatorial optimization problem in polynomial time. As benefit we can solve a large scale problem in reasonable time. However as our algorithm only searches for a local optimum, neither we can theoretically prove that the discovered model is the one with minimal average weighted distance to the variants, nor we can claim how close the discovered model is to the global optimum. Section 9.3 presents comprehensive simulation results to show performance of our clustering algorithm in different scenarios.

8.2. *Candidate Reference Process Models*

In order to validate our mining method, we compare the process model discovered with other candidate models. For this purpose, for each candidate model S_{cand} we assume that it is considered as new reference model. We then calculate average weighted distance between S_{cand} and the six variants $S_1 - S_6$ (cf. Fig. 4). For example, average weighted distance between S' (cf. in Fig 10) and the six variants corresponds to 4.9 (cf. Section 8.1), which reflects how close the discovered reference model is to these process variants. Using the same method, we can compute the average weighted distance the other candidate reference models show in respect to the given variant collection.

There are two groups of process models that are candidates for becoming the new reference model. The first group contains all models we already know. Clearly, the six process variants S_1, S_2, S_3, S_4, S_5 and S_6 (cf. Fig. 4) belong to this group. Comparing these models with the one we obtain through process variant mining, already shows that it is not sufficient to simply set the reference model to the most frequently used process variant.

The second group of candidate reference process models are the ones we can discover through mining. Clearly, process model S' (cf. Fig. 10) we obtained with our algorithm belongs to this group. So far there has been no algorithm directly supporting the mining of process variants. Therefore, we apply traditional techniques from the field of *process mining* [61], and compare them with our approach as well. The goal of process mining techniques is to discover process models from execution logs. An execution log typically documents the start/end of each activity execution in a PAIS, and therefore reflects the behavior of its implemented processes. In our case, we assume that the behavior of all process variants is fully covered by an execution log, i.e., we enumerate all traces producible by each process variant (see [72] for a technique we apply in this context[1]). The trace sets generated by different variants are merged together into one trace set. When determining the number of instances for each trace [61], we also take the weight of each variant into consideration. For example, since weight w_1 of variant S_1 corresponds to 20%, we ensure

[1]In principle, a process model containing loop structure can generate infinite number of traces. To ease the comparison without loosing generality, we assume that a loop in a process model is either not triggered or is triggered maximally once.

that the number of instances generated by S_1 accounts for 20% of the total number of instances as well.

The enumerated trace set is imported to ProM framework, which is one of the most popular tools for process mining and process analysis [64]. We consider Alpha algorithm [63], Heuristics Mining [71], Multi-phase Mining [65], DWS Mining [14] and Genetic Mining [10], which are some of the most well-known algorithms for discovering process models from execution logs. For each of these algorithms, we consider two cases. In the first one we consider all activities in the trace set, while the second case only includes activities which occur in more than 20% of the instances. The discovered process models are depicted in Fig. 12. We do no consider the model discovered by genetic mining here any longer, since it is too complex in structure, i.e., it results in a complex model with 13 silent activities.

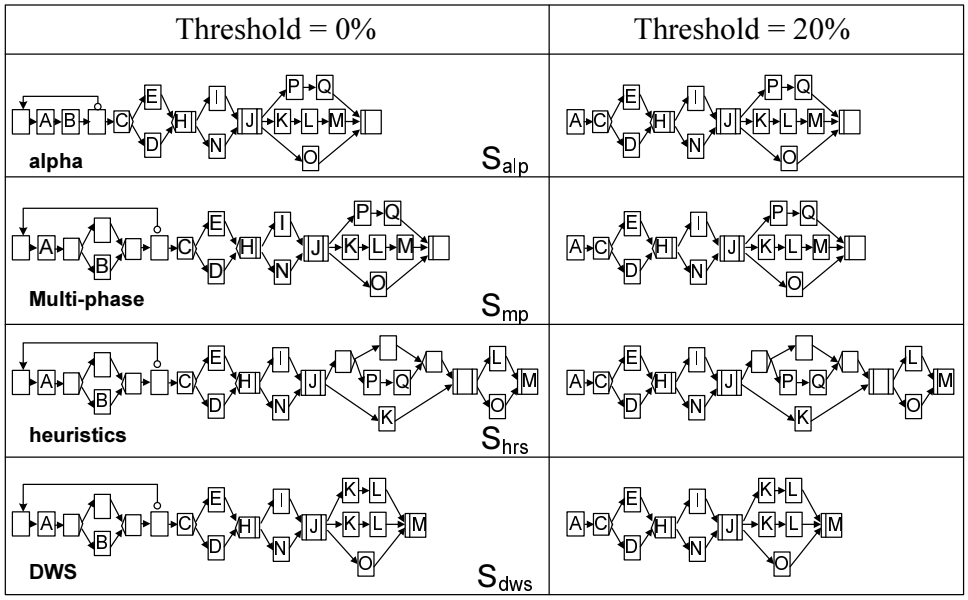


Fig. 12. The candidate process models

8.3. *Comparison Results*

We now compute average weighted distances between each candidate model S_{cand} from the two groups and the given collection of process variants $S_1 - S_6$. Comparison results are depicted in Fig. 13. When compared with traditional process mining techniques, it is clear that our algorithm can find reference models which have shorter average weighted distances: independent from the threshold we set on occurrences, the discovered model using our algorithm is better than the ones generated by other process mining algorithms. If we set the threshold of occurrence to 20%, the discovered model is also better than all the process variants existing in the

variants. Clearly, setting different threshold values would lead to different resulting models and consequently result in different average weighted distances to the variants. We discuss in detail on how to find a good threshold value later in Section 9.3. Altogether, results from Fig. 13 show that S' (cf. Fig. 10) – the process model resulting from the approach we suggest – has shortest average weighted distance to the given process variants; i.e., setting S' as new reference process model requires lowest efforts for configuring the variants. More precisely, we only need to perform on average 3.7 changes to configure a process variant out of S'. Similar results are obtained when mining other collections of process variants.

Existing models						Discovered models					
S_1	S_2	S_3	S_4	S_5	S_6	Occurrence	S'	S_{alp}	S_{hrs}	S_{mp}	S_{dws}
4.15	4.1	4.3	5.55	4.3	4.05	> 20%	3.7	4.75	5.5	4.75	4.65
						> 0%	4.9	5.95	6.7	5.95	5.85

Fig. 13. Average weighted distance between candidate process models and process variants

Obviously, it is not possible to enumerate all process models, since the number can be infinite. However, as depicted in Fig. 13, the discovered process model is at least better than the existing ones and the ones we can discover by traditional process mining algorithms based on traces. Keeping our search at local optimum also makes our approach applicable to real-world scenarios, since we can limit complexity to polynomial level.

Of course, our comparison results do not imply that process variant mining is better than process mining. Each of them has different inputs and goals. When compared to process mining, which tries to discover the underlying process model by learning from PAIS behavior, process variant mining focuses on discovering a reference process model which can be easily configured to the different process variants. If we apply the process mining evaluation criteria to measure the result of process variant mining, obviously, the discovered process model S' (cf. Fig. 10) will be also not good in terms of behavior.[54] Reason is that the behavior of S', which can be expressed by the trace set producible on S', is limited when compared to the trace sets the variants can produce.

9. Implementation and Simulation

In this section, we formally describe our algorithm, sketch our proof-of-concept implementation, and provide some simulation results.

9.1. *The MinAdept Algorithm*

In pseudo code, the MinAdept clustering algorithm to perform process variants mining (cf. Sections 6 and 7) can be expressed as follows:

The mining starts with deciding on the set of activities to be included in the (new) reference process model (Lines 1-4). If the relative occurrence of an activity is larger than the specified threshold, we include it in the reference model. Following

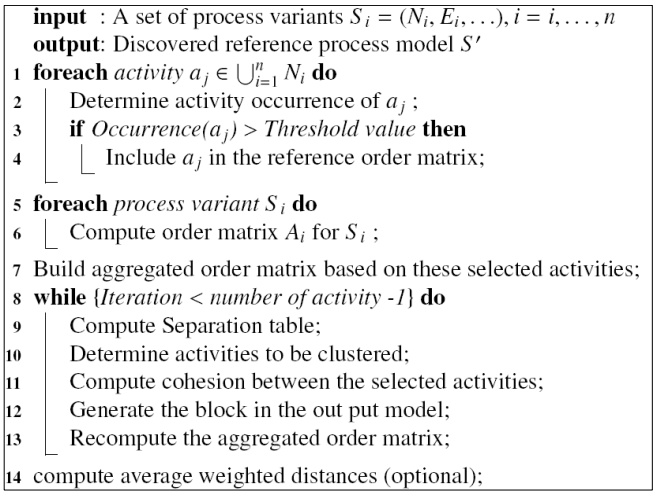


Fig. 14. Algorithm for discovering a reference process model by mining process variants

this we can construct the aggregated order matrix based on the order matrices of each process variant (cf. Section 6.1). Afterwards we apply our mining approach as described in Section 6, i.e., we cluster activities and blocks iteratively until all activities are contained in one block (lines 8-13 in Fig. 14). This block then represents the discovered reference model. Note that if there are unclear order relations, we apply the techniques described in Section 7.2. Finally, we can evaluate this model by computing average weighted distance of the reference model. Since computing this metrics has $\mathcal{NP}$-*hard* complexity, this step is optional.

9.2. *Proof-of-Concept Prototype*

The described approach was implemented and tested using Java. Figure 15 depicts a screenshot of our prototype. We used the ADEPT2 Process Template Editor [43] as tool for creating and visualizing process variants. For each process model, the editor can generate an XML representation with all relevant information (like nodes, edges, blocks) being marked up. We store created variants in a variants repository (cf. Fig. 15) which can be accessed by our mining procedure.

The mining algorithm was developed as stand-alone Java program, independent from the process editor. It can read the process variants and generate the discovered reference model according to the XML schema of the process editor. The obtained model is stored in the folder *"miningResult"* and can be visualized by again using the ADEPT2 editor.

ADEPT2 is a next generation adaptive process management tool, which enables flexible execution of process instances. Particularly, the ADEPT2 framework enables ad-hoc changes of single process instances during runtime as well as changes at the process type level and their propagation to running process instances if desired and possible.[42] In the meanwhile, an industrial-strength version of the ADEPT2 process management technology called AristaFlow BPM Suite is available for both academic

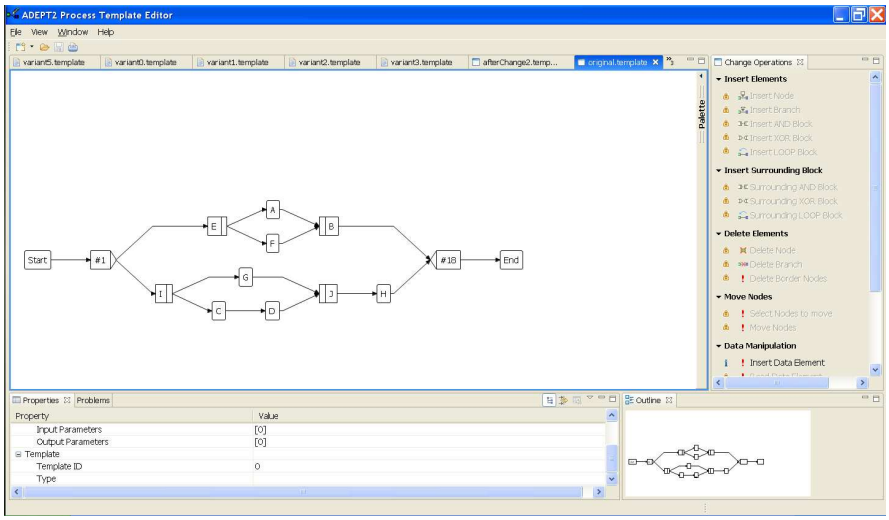


Fig. 15. Screenshot of the prototype

and industrial use.[8] The presented mining approach constitutes an important step towards full process life cycle support for dynamic processes.

9.3. *Simulation*

In Section 7, we have discussed how to mine process variants with different activity sets. One important step is to determine which activities should be considered in the reference model. In our approach, we apply a user-defined threshold to select activities: the reference model may only contain activities whose occurrence within the variants exceeds this threshold (see MinAdept algorithm in Fig. 14).

Clearly, determining a good threshold is critical. If the threshold is set too low (i.e., too many non-relevant activities are considered in the reference model), this will increase efforts for configuring process variants based on the discovered model. Then we need to delete non-relevant activities when configuring the variants. On the contrary, if the threshold is set too high (i.e., only few activities are considered in the reference model), configuration efforts might increase since we need to frequently insert activities to configure specific variants. Therefore, the influence of the chosen threshold value on our algorithm is of high interest.

Consider our illustrating example from Fig. 4. Fig. 10 shows the reference process models we can discover when setting different threshold values. When computing average weighted distance between these reference models and the six process variants, we obtain 4.9 for S', 3.7 for $S'_{0.2}$, 3.6 for $S'_{0.45}$, 3.7 for $S'_{0.55}$, and 3.95 for $S'_{0.8}$. This example indicates to set the threshold value at around 0,5 in order to obtain a reference process model with minimal average weighted distance.

Clearly, concluding this based on one example is not sufficient. We perform a simulation to analyze the influence of the threshold value. In our simulation, more than 5000 process models are generated and analyzed. We describe how the

simulation is setup in the next subsection, and discuss simulation results afterwards.

9.3.1. *Simulation Setup*

In order to obtain profound simulation results, we consider different parameters when generating the datasets for our simulation. Amongst others, these parameters include the *size* and the *similarity* of process models. We described our data generation method in detail in a Technical Report.[28]. In summary, we generate 54 groups of datasets according to different scenarios. Each dataset group contains:

(1) *A reference process model*, i.e., a randomly generated model from which we configure the process variants (see our report [28] for an algorithm).
(2) *100 process variants*. We generate each variant by configuring the reference model according to a particular scenario. For each group of datasets, we generate 100 process variants.

In total, we generated 5454 process models in our simulation. Note that the scenario just describes certain properties of the collection of variants configured from the reference model, but does not control the way a particular variant is generated; i.e., the 100 variants belonging to the same group are not the same, but share certain properties (e.g., having the same number of activities).

Since the variants are generated by configuring a given reference model, we can statistically control the occurrence of activities within the given collection of variants (see our Technical Report [28] for the method). In each group, we control the occurrence of the activities by inserting new activities with certain probabilities during the configuration of the process variants. The probabilities for inserting activities range from 0% to 100%. Consequently we obtain activities with different frequencies appearing in the variants. This way, setting different threshold values would result in different activity sets, and consequently different process model (as discovered by our clustering algorithm).

Following this, for each dataset group we apply our algorithm to discover a reference model by setting threshold values to 0%, 10%, 20%, ..., and 100%. We further evaluate these results by computing average weighted distance of the discovered model. We apply the described approach to all 54 dataset groups. In order to compare results from different groups, the absolute average weighted distance values are of less interest since each group has different features and covers different parts of the search space. Therefore, in each group we set the model discovered with a threshold value of 0% as basic model. We then evaluate the remaining models that were discovered when using other threshold values by comparing their average weighted distances to the one of the basic model. In the next subsections, we present the results we obtained from the 54 groups (5400 variants) of datasets.

9.3.2. *Simulation Results: Influence of Threshold Values*

Fig. 16 shows relatively changes of average weighted distances when applying different threshold values. We identified two types of clusters in the 54 groups of

datasets with each cluster containing 27 groups of datasets. The results from Fig. 16 are plotted as mean of the correlative values in each group.[m]

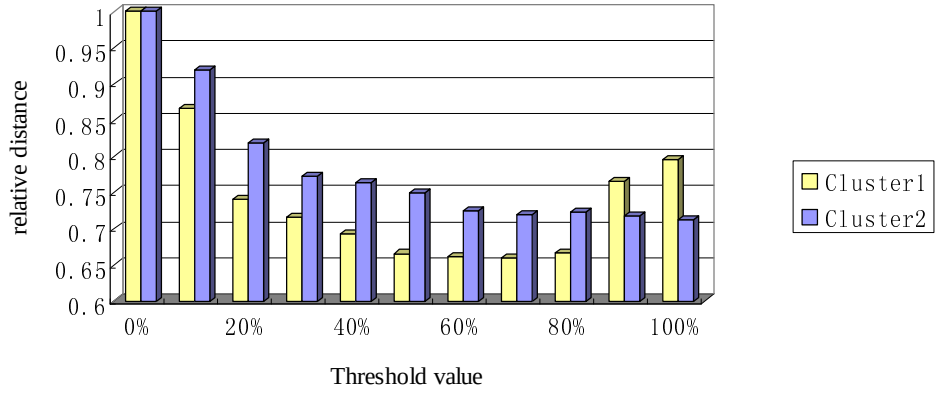


Fig. 16. Average weighted distances of discovered reference models when setting different threshold values

The two clusters all start at 1 when setting the threshold to 0%. This is easy to comprehend since this is the model based on which we compare the results in each group. In both clusters the relative distances decrease with increasing threshold until around 50%. This indicates that if we filter out the activities with low occurrence, the discovered reference model will have a shorter distance to the variants, and consequently will require less configuration efforts. When the threshold value reaches 50%, the two clusters start to show different behaviors. While average weighted distance in Cluster1 begins to increase with increasing threshold, the one in Cluster2 remains relatively stable. This difference triggered us to perform further analysis on the datasets.

For the datasets in Cluster1, the positions where the activities are inserted in the reference model to configure the variants are relatively stable. Therefore, when considering these frequent changes in the discovered reference model, we can potentially reduce its average weighted distance to the variants. As shown in Fig. 16, when filtering out these activities (i.e., not considering these frequent changes [n]), average weighted distance of the discovered reference increases. For the datasets in Cluster2, the positions where the activities are inserted in the reference model during process configurations are not stable. Therefore, it does not matter that much whether or not to include these activities in the discovered reference model. Since the positions of these activities are not stable, even if we include them in the new reference model, there will be a frequent need to move them to their respective positions in the different variants. This explains why changing the threshold value does not influence the average weighted distance too much for the datasets in Cluster2.

[m]Data of each group is available online at http://wwwhome.cs.utwente.nl/ lic/Resources.html.
[n]activity insertions in our case

Besides analyzing the means of the average weighted distances for the models resulting from each dataset group (cf. Fig. 16), we analyzed their standard deviations. In both clusters, standard deviations are low and stable when using different threshold values. Except the cases for which the threshold value is 0%, the standard deviations of the distances for all threshold values are around 0.115 (with plus/minus of 0.02). Such stable and low standard deviations indicate that in all groups the results follow almost the same trend as depicted in Fig. 16. Therefore, our conclusion has very low probability of being drawn by randomness.

For a given collection of variants, knowing which cluster it belongs to can greatly help for deciding about a good threshold value. However, it is very difficult to know whether or not the positions of an activity within a collection of variants are stable. We are able to obtain such information since we can control how the variants are generated in our simulation. Therefore, in order to obtain a model with shorter average weighted distance, we suggest setting the threshold to *around 50%*. In this interval, both clusters show relatively better results when compared to other values.

9.3.3. *Simulation Results: Running Time*

Besides analyzing the influence of threshold values on the end results, we also analyze how fast our clustering algorithm runs. In our simulation, we evaluate scenarios in which the process models contain per average 10, 20, and 50 activities in each group [o] The average running time for the 54 groups of datasets (each group contains 100 variants) is summarized in Fig. 17.

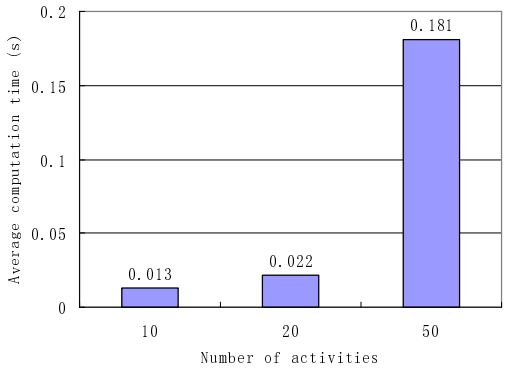


Fig. 17. The average running time for process model with different sizes

We use Dell Latitude D630 laptop (2.4GHz CUP and 3.5 GB RAM) to run the simulation under Windows environment. It is clear from Fig. 17 that the average running time required for even large process models (containing 50 activities) is 0.18 seconds, which is a significant low number.

[o]According to a recent study [34], process models containing more than 50 activities have a high risk of errors. Following this guideline, we set the largest size of a process model to 50 activities in our simulation.

10. Related Work

Though the flexible support of process variants is highly relevant in practice, only few approaches for process variant management exist. In particular, there is no adequate solution for learning from the model adaptations that have been applied when configuring the process variants out of a given process model.

Structural process changes during runtime and approaches for flexible process configuration have been intensively discussed in literature for several years.[46,47,68,70,44] A comprehensive analysis of theoretical and practical issues related to (dynamic) process changes, for example, has been provided in the context of the ADEPT2 change framework.[41,44] Furthermore, there exist theoretical frameworks for dynamic structural changes of Petri nets [58]. Based on these theoretical considerations, the AristaFlow BPM Suite [43,8] and tools for configurable workflow models [12] have emerged.

There exist approaches which provide support for the management and retrieval of separately modeled process variants. For example, the approach described in [31,32] provides support for storing, managing, and querying large collections of process variants within a process repository. Graph-based search techniques are used in order to retrieve variants that are similar to a user-defined process fragment. Obviously, this approach requires profound knowledge about the structure of stored processes, an assumption which does not always hold in practice. Apart from this, no techniques for analyzing the different process variants and for learning from their specific customizations are provided.

In the field of process mining, a variety of techniques has been suggested [61,71,10,63,65,14]. As illustrated in Section 8, traditional process mining is different from process variant mining due to its different goals and inputs. The approach introduced in [21] presents a method to mine configurable process models based on event logs, but is still focusing on discovering process models from event logs rather than reducing efforts for process configuration.

There are few techniques which allow to learn from process variants by mining change primitives (e.g., to add or delete control edges). For example, the approach presented in [1] measures process model similarity based on change primitives and suggests mining techniques using this measure. Similar techniques for mining change primitives exist in the field of association rule mining [56], frequent sub-graph mining [20,23], or graph pattern discovery [74] known from graph theory [53]; here common edges between different nodes are discovered to construct a common sub-graph from a set of graphs. Such techniques are commonly applied in the field of bioinformatics for "subdue" discovery, where "subdue" represents a certain sub-structure of genes or proteins, which has a certain chemical or biological behavior.[19] However, this approach does not consider important features of process meta model; e.g., it is unable to deal with silent activities and cannot differentiate between AND- and XOR-branchings or Loops.

The ProCycle system enables change reuse at the process instance level to ef-

fectively deal with recurrent problem situations.[49,69] ProCycle applies case-based reasoning techniques to allow for the semantic annotation as well as the retrieval of process changes. Based on this respective process adaptations can be re-applied in similar problem context to configure other process instances later on. If the reuse of a particular change exceeds a certain threshold, it becomes a candidate for adapting the process schema at type level; i.e., for evolving this schema accordingly and thus for considering the change for future process instances as well. Though the basic goal of ProCycle is similar to our approach, the techniques applied are much more simpler and do not consider variation in changes. An approach similar to ProCycle, which also enables change reuse based on cased-based reasoning techniques, is provided by CAKE2.[38]

To mine high level change operations, [15] presents an approach based on process mining techniques, i.e., the input consists of a change log capturing applied structural changes and process mining algorithms are applied to discover the execution sequences of the changes (i.e., the change process). However, this approach simply considers each change as individual operation such that the result is more like a visualization of changes. None of the discussed approaches aims at creating a reference process model, which allows for the easy and cost-effective configuration of process variants in future. [30] presents a technique to rank activities based on their potential involvement in process configurations. [27] further provides an approach to discover a reference model by performing a sequence of changes on the original reference model. However, this approach has high complexity and does also not provide any information on how different parts of the reference model fit to the variants.

In Configurable Workflow Models [12] all process variants are merged into one big reference process model based on inheritance rules known from Petri Nets [58]. Though techniques like questionnaire-based configuration can help users when configuring a variant [51], the reference model resulting from the merging of the variants turns to be complex and contains many decision points [13]. This approach becomes even more difficult when being confronted with a large collection of process variants not being equally important. In this case, an extremely large (complex) process model would result which might contain too many decision points and cannot differentiate between important variants and trivial ones.

In summary, none of the discussed approaches is sufficient in supporting the evolution of reference process model towards an easy and cost-effective model by learning from process variants in a controlled way.

11. Summary and Outlook

In this paper, we have provided a cluster-based approach for mining block-structured process variants. Our overall goal is to discover a reference process model out of a collection of process variants which can be easily configured to these variants. The proposed algorithm has polynomial complexity ($\mathcal{O}(n^3)$), which allows us to scale up when solving real-world problems. Based on a quantitative analysis, we have shown that the reference model discovered with our approach is better (i.e.,

requires a lower number of change operations for configuring the variants) than all process models known in the system. It is also better than all models we can discover when applying conventional process mining algorithms [61] to our problem. Our validation results also imply that current process mining techniques cannot fulfill the goal of discovering a process model which is easily configurable. To our best knowledge, this paper has provided the first polynomial algorithm to discover easy to configure reference process models through the mining of process variants.

Our approach looks promising, but there are still several questions left open. First we have to include more control structures (like synchronization constraints for parallel activities) as proposed in the ADEPT framework [41] or in WS-BPEL. Second, as learned from our case studies, data flow plays a very important role in process configuration as well. We therefore will consider data flow issues when designing other algorithms for process variant ming. It will be also advantageous if we can extend our approach to directly consider non-block-structured process models. At last, we are planning to integrate our algorithm with process mining techniques [61] such that the discovered reference process model can take both the structure and behavior perspective into account.

References

1. J. Bae, L. Liu, J. Caverlee, and W.B. Rouse. Process mining, discovery, and integration using distance measures. In *ICWS '06*, pages 479–488, Washington, DC, USA, 2006.
2. R.A. Baeza-Yates. Text-retrieval: Theory and practice. In *IFIP'92*, pages 465–476. North-Holland Publishing Co., 1992.
3. R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. Context aware trace clustering: Towards improving process mining results. In *SDM'09*, pages 401–412. SIAM, 2009.
4. BPEL. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.
5. OMG BPMI.org. *Business Process Modeling Notation Specification*. Object Management Group, 2006. available at: http://www.bpmn.org.
6. P.F. Brown, V.J. Della Pietra, P.V. de Souza, J.C. Lai, and R.L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.
7. C. Combi and M. Gambini. Flaws in the flow: The weakness of unstructured business process modeling languages dealing with data. In *OTM Conferences (1)*, pages 42–59. LNCS 5870, Springer, 2009.
8. P. Dadam and M. Reichert. The ADEPT project: A decade of research and development for robust and flexible process support - challenges and achievements. *Computer Science - Research and Development*, 23(2):81–97, 2009.
9. T.H. Davenport. *Mission Critical - Realizing the Promise of Enterprise Systems*. Harvard Business School, 2000.
10. A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, NL, 2006.
11. R.M. Dijkman, M. Dumas, L. Garcia-Banuelos, and R. Kaarik. Aligning business process models. In *EDOC'09*, pages 45–53, 2009.
12. F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable workflow models. *Int. J. Cooperative Inf. Syst.*, 17(2):177–221, 2008.
13. F. Gottschalk, T.A.C. Wagemakers, M.H. Jansen-Vullers, W.M.P. van der Aalst, and M. La Rosa. Configurable process models: Experiences from a municipality case study.

In *CAiSE'09*, pages 486–500. LNCS 5565, Springer, 2009.
14. G. Greco, A. Guzzo, and L. Pontieri. Mining hierarchies of models: From abstract views to concrete specifications. In *BPM'05*, pages 32–47. LNCS 3649, Springer, 2005.
15. C.W. Günther, S. Rinderle-Ma, M. Reichert, W.M.P. van der Aalst, and J. Recker. Using process mining to learn from process changes in evolutionary systems. *Int'l Journal of Business Process Integration and Management*, 3(1):61–78, 2008.
16. A. Hallerbach, T. Bauer, and M. Reichert. Managing process variants in the process lifecycle. In *ICEIS '08*, pages 154–161. Springer, 2008.
17. A. Hallerbach, T. Bauer, and M. Reichert. Capturing variability in business process models: the provop approach. *Software Process: Improvement and Practice*, page to appear, 2009.
18. J. Hidders, M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede, and J. Verelst. When are two workflows the same? In *CATS '05:*, pages 3–11, 2005.
19. L.B. Holder, D.J. Cook, and S. Djoko. Substructure discovery in the subdue system. In *AAAI Workshop'94*, pages 169–180, 1994.
20. J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. In *KDD '04*, pages 581–586, New York, NY, USA, 2004. ACM.
21. M.H. Jansen-Vullers, W.M.P. van der Aalst, and M. Rosemann. Mining configurable enterprise information systems. *Data Knowl. Eng.*, 56(3):195–244, 2006.
22. B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. On structured workflow modelling. In *CAiSE'00*, pages 431–445. LNCS 1789, Springer, 2000.
23. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM'01*, pages 313–320. IEEE Computer Society, 2001.
24. R. Lenz and M. Reichert. It support for healthcare processes - premises, challenges, perspectives. *Data Knowledge Engineering*, 61(1):39–58, 2007.
25. C. Li, M. Reichert, and A. Wombacher. Discovering reference process models by mining process variants. In *ICWS'08*, pages 45–53. IEEE Computer Society, 2008.
26. C. Li, M. Reichert, and A. Wombacher. On measuring process model similarity based on high-level change operations. In *ER '08*, pages 248–262. LNCS 5231, Springer, 2008.
27. C. Li, M. Reichert, and A. Wombacher. Discovering reference models by mining process variants using a heuristic approach. In *BPM'09, LNCS 5701*, pages 344–362. Springer, 2009.
28. C. Li, M. Reichert, and A. Wombacher. A heuristic approach for discovering reference models by mining process model variants. Technical Report TR-CTIT-09-08, University of Twente, The Netherlands, March 2009.
29. C. Li, M. Reichert, and A. Wombacher. Representing block-structured process models as order matrices: Basic concepts, formal properties, algorithms. Technical Report TR-CTIT-09-47, University of Twente, The Netherlands, 2009.
30. C. Li, M. Reichert, and A. Wombacher. What are the problem makers: Ranking activities according to their relevance for process changes. In *ICWS'09*, pages 51–58. IEEE Computer Society, 2009.
31. R. Lu and S.W. Sadiq. Managing process variants as an information resource. In *BPM'06*, pages 426–431, 2006.
32. R. Lu and S.W. Sadiq. On the discovery of preferred work practice through business process variants. In *ER'07*, pages 165–180. Springer, 2007.
33. J. Mendling. *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction and Guidelines for Correctness*, volume 6 of *LNBIP*. Springer, 2008.
34. J. Mendling, G. Neumann, and W.M.P. van der Aalst. Understanding the occurrence of errors in process models based on metrics. In *CoopIS'07, LNCS 4803*, pages 113–

130, 2007.
35. J. Mendling, H.A. Reijers, and J. Cardoso. What makes process models understandable? In *BPM'07*, pages 48–63. LNCS 4714, Springer, 2007.
36. J. Mendling, H.A. Reijers, and W.M.P. van der Aalst. Seven process modeling guidelines (7pmg). *Information & Software Technology*, 52(2):127–136, 2010.
37. J. Mendling, B.F. van Dongen, and W.M.P. van der Aalst. Getting rid of or-joins and multiple start events in business process models. *Enterprise Information Systems*, 2(4):403–419, 2008.
38. M. Minor, A. Tartakovski, D. Schmalen, and R. Bergmann. Agile workflow technology and case-based change reuse for long-term processes. *International Journal of Intelligent Information Technologies*, 4(1):80–98, 2008.
39. B. Mutschler, M. Reichert, and J. Bumiller. Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors and implications. *IEEE Transactions on Systems, Man, and Cybernetics*, 38(3):280–291, 2008.
40. D.L. Parnas. Software aging. In *ICSE '94*, pages 279–287. IEEE Computer Society Press, 1994.
41. M. Reichert and P. Dadam. ADEPTflex -supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
42. M. Reichert, S. Rinderle, and P. Dadam. On the common support of workflow type and instance changes under correctness constraints. In *CoopIS'03*, pages 407–425, 2003.
43. M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with ADEPT2. In *ICDE '05*, pages 1113–1114. IEEE Computer Society, 2005.
44. M. Reichert, S. Rinderle-Ma, and P. Dadam. Flexibility in process-aware information systems. *Transactions Petri Nets and Other Models of Concurrency*, 2:115–135, 2009.
45. H.A. Reijers and J. Mendling. Modularity in process models: Review and effects. In *BPM'08*, pages 20–35. LNCS 5240, Springer, 2008.
46. S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems – a survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.
47. S. Rinderle, M. Reichert, and P. Dadam. Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases*, 16(1):91–116, 2004.
48. S. Rinderle, M. Reichert, and P. Dadam. On dealing with structural conflicts between process type and instance changes. In *BPM'04*, pages 274–289. LNCS 3080, 2004.
49. S. Rinderle, B. Weber, M. Reichert, and W. Wild. Integrating process learning and process evolution a semantics based approach. In *Proc. 3rd Int'l Conf. on Business Process Management (BPM'05)*, LNCS 3649, pages 252–267, 2006.
50. S. Rinderle-Ma, M. Reichert, and B. Weber. On the formal semantics of change patterns in process-aware information systems. In *ER'08*, LNCS 5231, pages 279–293, 2008.
51. M. La Rosa, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Questionnaire-based variability modeling for system configuration. *Software and System Modeling*, 8(2):251–274, 2009.
52. M. Rosemann and W.M.P. van der Aalst. A configurable reference modelling language. *Information Systems*, 32(1):1–23, 2007.
53. K.H. Rosen. *Discrete Mathematics and its Application*. McGraw-Hill, 2003.
54. A. Rozinat, A.K.A. de Medeiros, C.W. Günther, A.J.M.M. Weijters, and W.M.P. van der Aalst. The need for a process mining evaluation framework in research and practice. In *BPI'07*, pages 84–89, Brisbane, Australia, 2007. LNCS 4928, Springer.
55. R. Sabherwal and Y.E. Chan. Alignment between business and is strategies: A study of prospectors, analyzers, and defenders. *Info. Sys. Research*, 12(1):11–33, 2001.
56. P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley,

2005.
57. L. Thom, M. Reichert, and C. Iochpe. Activity patterns in process-aware information systems: Basic concepts and empirical evidence. *International Journal of Business Process Integration and Management (IJBPIM)*, 4(2):93–110, 2009.
58. W.M.P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203, 2002.
59. W.M.P. van der Aalst and T.Basten. Identifying commonalities and differences in object life cycles using behavioral inheritance. In *ICATPN '01*, pages 32–52, London, UK, 2001. Springer.
60. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
61. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: a survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, 2003.
62. W.M.P. van der aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems*. The MIT Press, 2002.
63. W.M.P van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1128–1142, 2004.
64. B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In *ICATPN'05*, pages 444–454. LNCS 3536, Springer, 2005.
65. B.F. van Dongen and W.M.P. van der Aalst. Multi-phase process mining: Building instance graphs. In *ER'04*, pages 362–376. LNCS 3288, Springer, 2004.
66. J. Vanhatalo, H. Völzer, and J. Koehler. The refined process structure tree. *Data Knowledge Engineering*, 68(9):793–818, 2009.
67. B. Weber and M. Reichert. Refactoring process models in large process repositories. In *CAiSE'08*, pages 124–139. LNCS 5074, Springer, 2008.
68. B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, 66(3):438–466, 2008.
69. B. Weber, M. Reichert, W. Wild, and S. Rinderle-Ma. Providing integrated life cycle support in process-aware information systems. *Int'l Journal of Cooperative Information Systems*, 18(1):115–165, 2009.
70. B. Weber, S. Sadiq, and M. Reichert. Beyond rigidity - dynamic process lifecycle support: A survey on dynamic changes in process-aware information systems. *Computer Science - R&D*, 23(2):47–65, 2009.
71. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integr. Comput.-Aided Eng.*, 10(2):151–162, 2003.
72. A. Wombacher, P. Fankhauser, and E. Neuhold. Transforming BPEL into annotated deterministic finite state automata for service discovery. In *ICWS'04*, page 316. IEEE Computer Society, 2004.
73. A. Wombacher and M. Rozie. Evaluation of workflow similarity measures in service discovery. *Service Oriented Electronic Commerce*, pages 51–71, 2006.
74. X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM'02*, pages 721–724. IEEE Computer Society, 2002.
75. M. zur Muehlen and J. Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *CAiSE'08*, pages 465–479. LNCS 5074, Springer, 2008.