Issues in Process Variants Mining

Chen Li[1], Manfred Reichert[2], and Andreas Wombacher[3]

[1] Information System group, University of Twente, The Netherlands
lic@cs.utwente.nl
[2] Institute of Databases and Information System, Ulm University, Germany
manfred.reichert@uni-ulm.de
[3] Database group, University of Twente, The Netherlands
a.wombacher@utwente.nl

Abstract. In today's dynamic business world economic success of an enterprise increasingly depends on its ability to react to internal and external changes in a quick and flexible way. In response to this need, process-aware information systems (PAIS) emerged, which support the modeling, orchestration and monitoring of business processes and services respectively. Recently, a new generation of flexible PAIS was introduced, which additionally allows for dynamic process and service changes. This, in turn, will lead to a large number of process variants, which are created from the same original process model, but might slightly differ from each other. This paper deals with issues related to the mining of such process variant collections. Our overall goal is to learn from process changes and to merge the resulting model variants into a generic process model in the best possible way. By adopting this generic process model in the PAIS, future cost of process change and need for process adaptations will decrease. Finally, we compare our approach with existing process mining techniques, and show that process variants mining is additionally needed to learn from process changes.
Index Terms—**Process Configuration, Process Variants, High-level Change, process variants mining**

1 Introduction

Economic success of enterprises increasingly depends on their ability to react to changes in a quick, flexible, and cost-effective way. However, current off-the-shelf enterprise software does not fully meet this fundamental requirement [4]. It is deployed in different companies, domains, and countries, and therefore tends to be too generic and rigid. This causes huge customization efforts at the site of software buyers that exceed the price for software licenses by factor five to ten [2]. Software vendors, in turn, make endeavors to close this gap [5], and major progress has been achieved by shifting from function- to process- and service-centered software design.

Along this trend a variety of process and service support paradigms (e.g., service orchestration, service choreography, adaptive processes and services) and corresponding specification languages (e.g., WS-BPEL, WS-CDL, WSDL) have emerged. In addition, different approaches for flexible and adaptive processes exist [8, 12]. Generally, process and service adaptations are not only needed for configuration purposes at build time, but also become necessary during runtime to deal with exceptional situations and changing needs; i.e., for single instances of composite services and processes respectively, it must be possible to dynamically adapt their structure (i.e., to insert, delete or move activities during runtime). In response to this need adaptive process management technology has emerged, which allows for such dynamic process and service changes [6].

Obviously, the ability to adapt and configure processes at the different levels will result in a collection of process model variants (i.e., configurations [3]) created from the same process model, but slightly differing from each other. Fig. 1 depicts an example. The left hand side shows a high-level view on a patient treatment process as it is normally executed: a patient is *admitted* to a hospital, where he first *registers*, then *receives treatment*, and finally *pays*. In emergency situations, however, it might become necessary to deviate from this model, e.g., by first starting treatment of the patient and allowing him to register later during treatment. To capture this behavior in the model of the respective process instance, we need to move activity *receive treatment* from its current position to a position parallel to activity *register*. This leads to an (instance-specific) process model variant S' as shown on the right hand side of Fig. 1. Generally, a large number of process model variants (process variants for short) derived from the same original process model might exist.

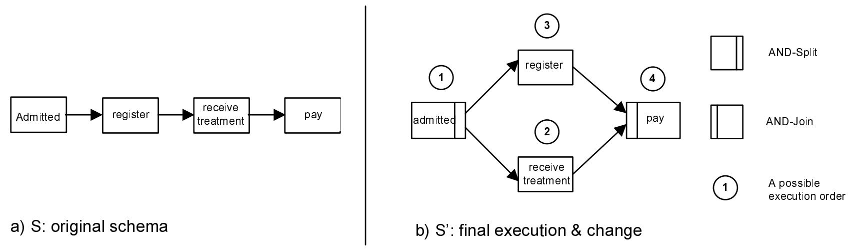


Fig. 1. Original Process Model S and Process Variant S'

In most approaches supporting the adaptation and configuration of process models each resulting process variant has to be maintained by its own, and even simple changes within a domain or organization (e.g. due to new laws or re-engineering efforts) might require manual re-editing of a large number of process variants. Over time this might lead to degeneration and divergence of the respective process models [4], which aggravates maintenance significantly. In this paper we deal with issues related to the mining of such process variant collections. Our goal is to learn from the process changes applied in the past and to merge the resulting process variants into a generic process model which covers the existing process variants best. By adopting this generic process model within the PAIS, cost of change and need for future process adaptations will decrease.

1.1 Full Life Cycle Support in PAIS

We give background information on process life cycle support to illustrate the setting of our approach. Fig. 2 depicts the life cycle of adaptive processes. It starts with the design of a process model based on which new process instances can be created and executed. Relevant execution events (e.g., about start and completion of process activities) are stored in the *execution log*. In Fig. 1b, for example, the numbers indicate the order in which the activities were executed. During runtime, authorized users may deviate from the predefined process model to deal with exceptional or non-anticipated situations. Such instance-specific deviations can be realized based on high-level change operations (e.g., to add, delete or move process activities). Each change may comprise several change operations (with parameterization) which are stored in the *change log* in the order they were applied [7]. Taking the information from execution and change logs, we can learn from the past and discover opportunities to optimize and evolve process models. Newly discovered process model versions will then replace the original one [8]. For long-running process instances it might additionally become necessary to migrate them to the new process model version [9].

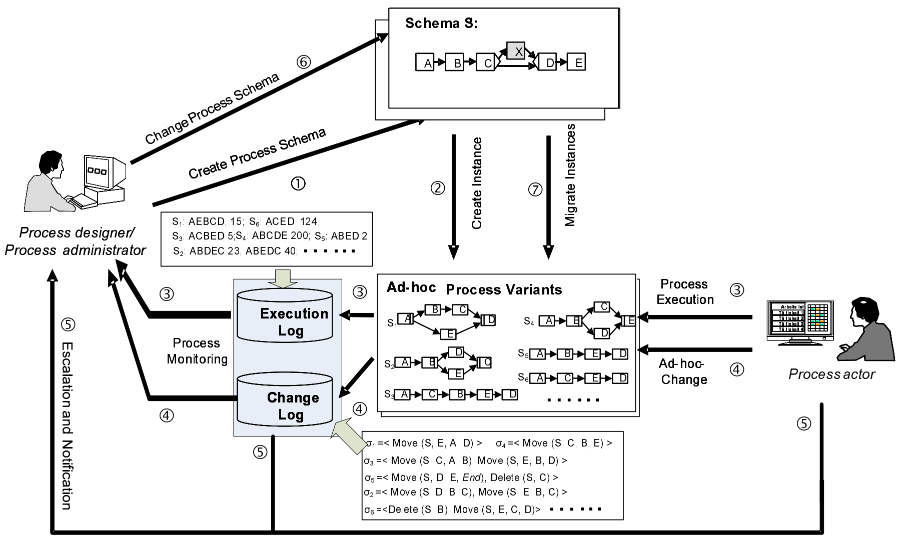

Schema S:
A
B
C
X
D
E
⑥
Change Process Schema
Create Process Schema
①
② Create Instance
⑦ Migrate Instances
Process designer/ Process administrator
S1: AEBCD, 15; S6: ACED 124;
S3: ACBED 5;S4: ABCDE 200; S5: ABED 2
S2: ABDEC 23, ABEDC 40; ......
Execution Log
Change Log
Process Monitoring
③
④
⑤
Escalation and Notification
Ad-hoc Process Variants
S1
S2
S3
S4
S5
S6
Process Execution
Ad-hoc-Change
Process actor
σ1 =< Move (S, E, A, D) > σ4 =< Move (S, C, B, E) >
σ3 =< Move (S, C, A, B), Move (S, E, B, D) >
σ5 =< Move (S, D, E, End), Delete (S, C) >
σ2 =< Move (S, D, B, C), Move (S, E, B, C) >
σ6 =<Delete (S, B), Move (S, E, C, D)> ....

Fig. 2. The Life Cycle of a PAIS

Many efforts have been undertaken to realize PAIS with full lifecycle support. In particular, adaptive process management technology like ADEPT [8], WASA2

[10], or TRAMs[11] has emerged (for an overview see [12]), which provide users with the needed flexibility to adapt the PAIS to real-world situations at both the process instance and the process type level. In particular, these systems allow for dynamic process changes during runtime without affecting robustness and reliability of the PAIS. Furthermore, process mining techniques [17] have been introduced for discovering optimized processes based on execution logs. The only incomplete link with respect to full process life-cycle support in Fig. 2 concerns Step 5; i.e. so far there have been no advanced techniques which enable process learning based on change logs.

1.2 Research Question

Process mining has been extensively studied in literature [17]. Its key idea is to discover a process model by analyzing the execution behavior of (completed) process instances as captured in execution logs. Different mining techniques like the alpha algorithm [24], heuristics mining [26], and genetic mining [27] have been proposed in this context. When considering the extensive research on process mining, only little work has dealt with *process variants mining* so far. Here the overall goal is to to evolve a process model over time by learning from the changes applied to corresponding model instances in the past. As aforementioned, process model adaptations are captured in change logs and can be applied at different levels; e.g., in the context of process model configurations or in connection with deviations at the process instance level. In any case, we obtain a collection of process model variants when applying changes to different instances of a process model. By learning from these model variants and by merging them into a generic process model, effort for future process model configurations as well as adaptations can be reduced.

This paper deals with the following research questions:

Why is process variants mining needed and what are the differences between traditional process mining and the mining of process variants?

Our aim is to motivate the need for mining process variants and to discuss some of the major challenges arising in this context. Details of the mining algorithm itself are out of the scope of this paper. However, we have developed and implemented respective techniques, and will also utilize them for comparing variants mining with conventional process mining.

The remainder of this paper is organized as follows: Section 2 gives background information needed for the understanding of this paper. In Section 3 we discuss why process changes should be expressed in terms of high-level change operations. Section 4 discusses major goals of process variants mining and shows why it is different from traditional process mining. In Section 5 we present a concrete example to elaborate these differences. The paper concludes with a summary and an outlook in Section 7.

2 Backgrounds

We first introduce basic notions needed in the following: *process model*, *process change*, and *trace*.

Process model Let $\mathcal{P}$ denote the set of all process models. A single *process model* $S = (N, E, \ldots) \in \mathcal{P}$ is represented as Well-Structured Marking Net (WSM Net) [8], where N corresponds to the set of process activities and E constitutes the set of causal relations between them (i.e., control edges linking activities). To limit the scope, we omit other process aspects (e.g., data flow) here. Further, we assume process-models to be block structured. A detailed description of WSM Nets and their properties can be found in [21].

Process change We assume that a process variant results from an original process model S by applying a sequence of changes to it over time [8]. Such changes modify the initial model S by altering its set of activities or by changing their order relations through the application of a sequence of change operations. Thus, each change to a process model results in another (intermediate) process model.

Definition 1 (Process change). *Let $\mathcal{P}$ be 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 process changes performed on initial process model S. Then we can define:*

- *$S[\Delta\rangle S'$ iff Δ is applicable to S and S' is the process schema 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 $S_i[\Delta_i\rangle S_{i+1}$ with $i = \{1, \ldots n\}$. $|\sigma| = i$*

Examples of high-level change operations include *insert activity*, *delete activity*, and *move activity* as implemented in the ADEPT change framework [8]. While *insert* and *delete* modify the set of activities in the process model, *move* changes the position of an activity and thus the structure of the process model. For example, operation *move*(S, `A, B, C`) means to move activity `A` from its current position within process model S to the position after activity `B` and before activity `C`, while operation *delete*(S, A) expresses to delete activity `A` from process model S. Issues concerning the correct use of these operations as well as formal pre- and post-conditions are described in [8]. Though the depicted change operations are discussed in relation to ADEPT, they are generic in the sense that they can be easily applied in connection with other process meta models as well [6]. For example, a process change as described in the ADEPT framework can be mapped to the concept of life-cycle inheritance as known from Petri Nets [18]. We refer to ADEPT in this paper since it covers by far most high-level change patterns and change support features when compared to other approaches [6].

Definition 2 (Bias and Distance). *Let S, $S' \in \mathcal{P}$ be two process models. Then: The distance $d_{(S,S')}$ between S and S' corresponds to the minimal number of high-level change operations needed to transform process model S into process model S'; i.e., $d_{(S,S')} := min\{|\sigma| \mid \sigma \in \mathcal{C}^* \wedge S[\sigma\rangle S'\}$. 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 two process models S and S' is the minimal number of change operations needed for transforming S into S'. The corresponding sequence of change operations is denoted as *b*ias between S and S'.[4] Obviously, the shorter the distance between two process models, the more similar their control flow structure is. Generally, the distance between two process models measures the complexity for process model transformation or configuration. As example take Fig. 1. Here, the distance between S and S' is *one*, since we only need to perform one change operation $move(S, receivetreatment, admitted, pay)$ to transform S into S'. In general, determining bias and distance between two process models is rather difficult. The complexity is at $\mathcal{NP}$ level [20]. We omit further details and refer to [20].

Trace A *trace* t on process model S denotes a valid execution sequence $t \equiv < a_1, a_2, \ldots, a_k >$ of activities $a_i \in N$ on S according to the control flow defined by S. All traces process model S can produce are summarized in set $\mathcal{T}_S$. Finally, $t(a \prec b)$ denotes a precedence relationship between activities a and b in trace $t \equiv < a_1, a_2, \ldots, a_k >$ if and only if $\exists i < j : a_i = a \wedge a_j = b$. Here, traces only capture events related to 'real' activities, but not to silent ones (i.e., activity nodes which contain no operation and exist only for control flow purpose). Furthermore, we consider two process models as being the same if they are *trace equivalent*, i.e. $S \equiv S'$ if and only if $\mathcal{T}_S \equiv \mathcal{T}_{S'}$. Like most existing process mining algorithms [17], we do not consider the stronger notion of bi-similarity [22] here.

3 On Representing Process Changes

Before we deal with issues related to the the mining of process variants, we discuss basic aspects concerning the representation of process changes. First, we explain why process variants mining cannot be solely based on execution logs, but necessitates change log information as well. Second, we sketch why it is beneficial to express changes in terms of high-level operations (e.g., to move an activity) rather than low-level change primitives (e.g., to add/delete nodes and edges). Third, we discuss how the application of high-level change operation affects execution behavior of a process model.

[4] Generally, it is possible to have more than one minimal set of change operations to realize the transformation from S into S', i.e., given two process models S and S' their bias is not necessarily unique. A detailed discussion of this issue is out of the scope of this paper since we are more interested in the change distance. We refer readers to [18, 20] for details.

3.1 Why Do We Need a Change Log?

Change and execution logs capture different runtime information on process instances and are not interchangeable. Even if the original process model of a process instance is given, it is not possible to convert its change log to its execution log or vice verse. We refer to our example from Fig. 1. When applying the aforementioned change to original process model S, we obtain process variant S' (i.e. $S[\sigma\rangle S'$) with change log $\sigma =< move(S, reveive\ treatment, admitted, pay) >$. Assume that this process variant represents an instance-specific schema and that the trace of the particular instance is $\{admitted, receive\ treatment, register, pay\}$ (as indicated by the numbers in Fig. 1b). If original process model S and the instance trace had been the only available information, it would be not possible to determine the respective change. Note that the process model, which can produce the given trace, is not unique. For example, a process model with the four activities contained in four parallel branches could produce this trace as well. By contrast, it is generally not possible to derive the trace of a process instance from its change log, because execution behavior of S' is also not unique. For example, trace $< admitted, register, receive\ treatment, pay >$ is also producible on S'. Consequently, change and execution log capture different runtime information.

3.2 High-level Change Operations vs. Change Primitives

We now discuss why it is beneficial to measure the distance between process models based on high level change operations rather than on low-level change primitives. Consider the left-hand side of Fig. 3. It shows an original process model S which comprises a parallel branching (`C` and `D` may be performed concurrently), a conditional branching (either `E` or `F` is executed), and a silent activity τ (depicted as an empty node) connecting these two branching blocks. Assume that in two different scenarios high-level change operations are applied to S resulting in the two models S_1 and S_2 respectively: Δ_1 moves activity `C` from its current position to the position between activities `A` and `B`, resulting in process variant S_1, i.e., $S[\Delta_1\rangle S_1$ with $\Delta_1 = move(S, \texttt{C}, \texttt{A}, \texttt{B})$. Δ_2, in turn, moves activity `A` to the position between activities `B` and `C`, i.e., $S[\Delta_2\rangle S_2$ with $\Delta_2 = move(S, \texttt{A}, \texttt{B}, \texttt{C})$. Note that Fig. 3 additionally depicts the change primitives representing the snapshot differences between original process model S and process variants S_1 and S_2 respectively.

In comparison with low-level primitives, the the use of high-level change operations offers the following advantages:

1. High-level change operations with formal pre- and post-conditions, as supported by ADEPT and other process change frameworks, usually guarantee soundness (i.e., absence of deadlocks and livelocks); i.e., their application to a sound process model S results in another sound model S' [8]. This also applies to our example from Fig. 3. By contrast, when applying single primitives (e.g. deleting an edge), soundness cannot be guaranteed in general. For example, if we delete any of the edges in S, the resulting model will not be necessarily sound.

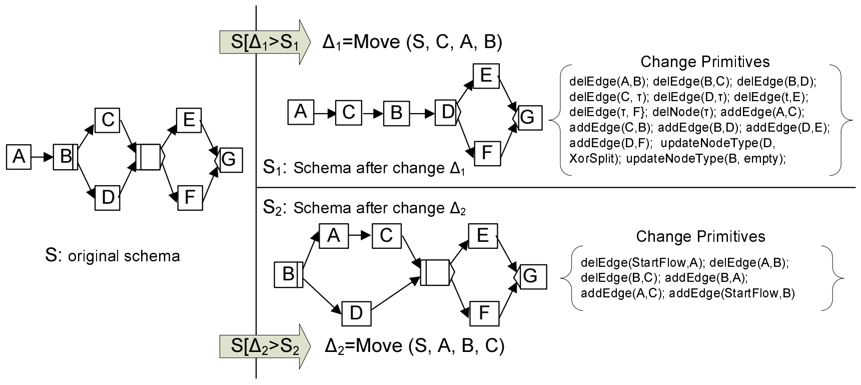

S[Δ1>S1
Δ1=Move (S, C, A, B)
Change Primitives
delEdge(A,B); delEdge(B,C); delEdge(B,D); delEdge(C, τ); delEdge(D,τ); delEdge(t,E); delEdge(τ, F}; delNode(τ); addEdge(A,C); addEdge(C,B); addEdge(B,D); addEdge(D,E); addEdge(D,F); updateNodeType(D, XorSplit); updateNodeType(B, empty);
S1: Schema after change Δ1
S: original schema
S2: Schema after change Δ2
Change Primitives
delEdge(StartFlow,A); delEdge(A,B); delEdge(B,C); addEdge(B,A); addEdge(A,C); addEdge(StartFlow,B)
S[Δ2>S2
Δ2=Move (S, A, B, C)
A
B
C
D
E
F
G

Fig. 3. High-Level Change Operation vs. Change primitives

2. High-level change operations enable more effective user support when compared to low-level change primitives. Generally, a high-level change operation is based on a set of primitives which collectively realize a particular change pattern. As example take Δ_1 from Fig. 3. This operation is internally based on *15* change primitives to delete and add edges, to delete the silent activity, and to update node types. By defining changes with high-level operations, cost of change can be significantly reduced. As another benefit, high-level operations usually perform model optimizations when realizing a process change. Regarding change Δ_1 from Fig. 3, for example, the movement of activity `C` is accompanied by the deletion of silent activity τ, since the parallel branching is no longer needed afterwards.
3. An important aspect, not discussed so far, concerns the number of change operations needed to transform a process model S into another model S'. Regarding our previous example, for instance, we only need *one* move operation to transform S to either S_1 or S_2. When using change primitives instead, migrating S to S_1 requires *15* change primitives, while the second change Δ_2 can be realized with *6* primitives. This also demonstrates that change primitives do not provide an adequate means to express the difference between two process models; i.e., the number of primitives needed for a process model transformation should not be used for expressing change efforts. As a consequence, we base our approach for process variants mining on high-level change operations.
4. When representing model changes by means of high-level operations, we can always derive the corresponding change primitives, but not the other way around; i.e., with respect to process model S it is sufficient to log the applied high-level change operations in order to derive the corresponding primitives and the resulting model variant S'. The change primitives can be derived by snapshot analysis; i.e., when performing snapshots of S and S', respective change primitives can be easily obtained by determining which nodes and edges have been deleted or added [7]. By contrast, if we only store low-level

primitives, computing the corresponding high-level change operation will be difficult. For example, how to determine that the 15 primitives needed to transform S into S_1 only represent one single high-level change? Generally, determining the high level change based on snapshots is an $\mathcal{NP}$ hard problem [20].

For these reasons we propose to use high-level change operations instead of low-level change primitives when measuring the distance between two process models.

3.3 How Do High-level Changes Influence Process Behavior?

[14] provides an approach to measure the similarity between two process models based on their trace sets: More precisely, the behavioral distance between process models S_1 and S_2 is calculated as the sum of the edit distances of all possible pairs of traces (t_1,t_2) with $t_1 \in \mathcal{T}_{S_1}$ and $t_2 \in \mathcal{T}_{S_2}$. Obviously, this method evaluates to what degree the behaviors of the two process models differ from each other rather than on what the effort for transforming one process model into another is. On the one hand, application of one high-level change operation might significantly modify the execution behavior of the respective process model. On the other hand, several high-level change operations might be required to realize a smooth change in execution behavior of a given process model. When considering the two process models from Fig. 1, for example, we obtain as behavioral distance *one*. However, when performing another change $S'[\Delta_2\rangle S''$ with $\Delta_2 = move(S, pay, admitted,\ receivetreatment)$ the behavioral distance between S and the S'' will be *four*. Particularly, when a change moves or adds activities to parallel branches, the number of possible traces might grow exponentially.

Based on these considerations, we have decided to focus on the relationship between behavior of process models and biases. In the next section, we compare the design strategies for process mining and process variants mining; i.e., we elaborate the goal for process variants mining and show why it differs from traditional process mining.

4 Mining Process Variants: Goals and Comparison with Process Mining

So far, we have motivated the need for representing process changes in terms of high-level operations. This section discusses the major goal of mining process variants, namely to derive a generic process model out of a given collection of process variants. This shall be done in a way such that the different process variants can be efficiently configured out of the generic model. We measure the efforts for respective process configurations by the number of high-level change operations needed to transform the generic model into the respective model variant. The challenge is to find a generic model such that the average number of change operations needed (i.e., the average distance) becomes minimal.

To make this more clear, we compare process variants mining with traditional process mining. Obviously, input data for process and process variants mining differ. While traditional process mining operates on execution logs, mining of process variants is based on change logs (or the process variants we can obtain from them). Of course, such high-level consideration is insufficient to prove that existing mining techniques do not provide optimal results with respect to the above goal. In principle, methods like alpha algorithm or genetic mining can be applied to our problem as well. For example, we could derive all traces producible by a given collection of process variants [14] and then apply existing mining algorithms to them. Or, if there are enough instances for each process model variant, we can mine their traces to discover a corresponding process model. To make the difference between process and process variants mining more evident, in the following, we consider behavioral similarity between two process models as well as structural similarity based on their bias.

The behavior of a process model S can be represented by the set of traces $\mathcal{T}_S$ it can produce. Therefore, two process models can be compared based on the difference between their trace sets [16, 14, 17]. By contrast, biases can be used to express the (structural) distance between two process models, i.e., the minimal number of high-level change operations needed to transform one model into the other [20]. While the mining of process variants addresses structural similarity, traditional process mining focuses on behavior. Obviously, this leads to different choices in algorithm design and also suggest different mining results. Fig. 4 shows two examples.

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 [17], the discovered process model will have a structure like 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 instances to it, all instances running eon S_1 or S_2 will have a non-empty bias. The average weighted distance between S and S_1 (S and S_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. More precisely, we would need to move either B or C in S to either obtain S_1 or S_2; i.e., $S[\sigma_1\rangle S_1$ with $\sigma_1 = move(S,\texttt{B},\texttt{A},\texttt{C})$ and $S[\sigma_2\rangle S_2$ with $\sigma_2 = move(S,\texttt{B},\texttt{C},\texttt{D})$. We describe a method to compute biases in [20].

By contrast, if the goal is to reduce the average bias between reference model and process (instance) variants, we should choose S' as reference model. Though S' does not cover all traces S_1 and S_2 can produce (i.e., $\mathcal{T}_{S_2} \nsubseteq \mathcal{T}_{S'}$) the average distance between generic model and process instances becomes minimal with this approach. More precisely, the average distance between S' and instances running either on S_1 or S_2 corresponds to *0.45*; i.e., while no adaptations become necessary for the 55 instances running on S_1, we need to move activity B for the 45 instances based on S_2, i.e. $S'[\sigma'\rangle S_2$ with $\sigma' = move(S',\texttt{B},\texttt{C},\texttt{D})$. Though S'

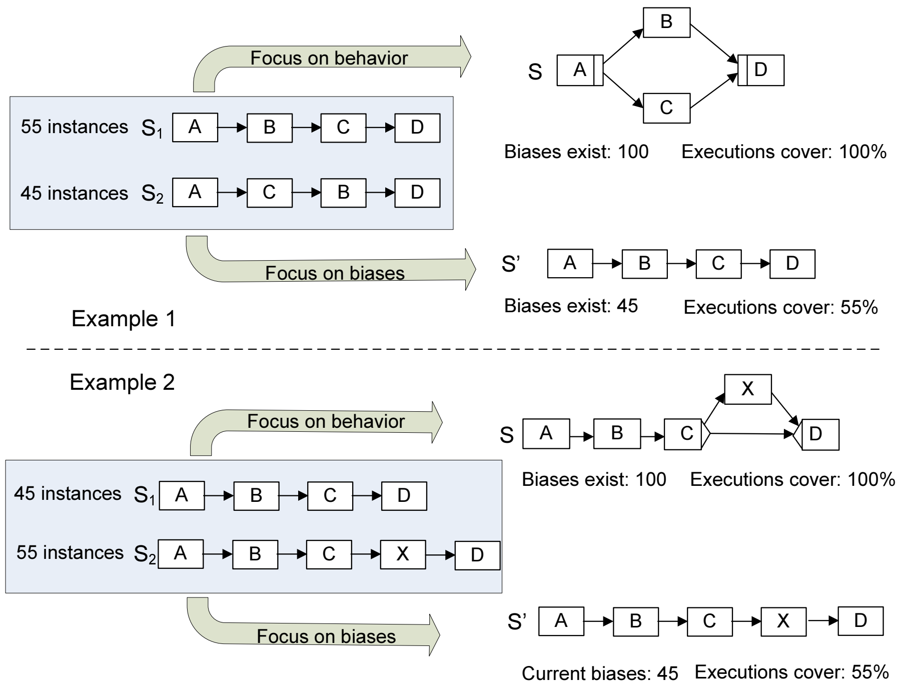


Fig. 4. Mining focusing either on Behavior or on Minimization of Biases

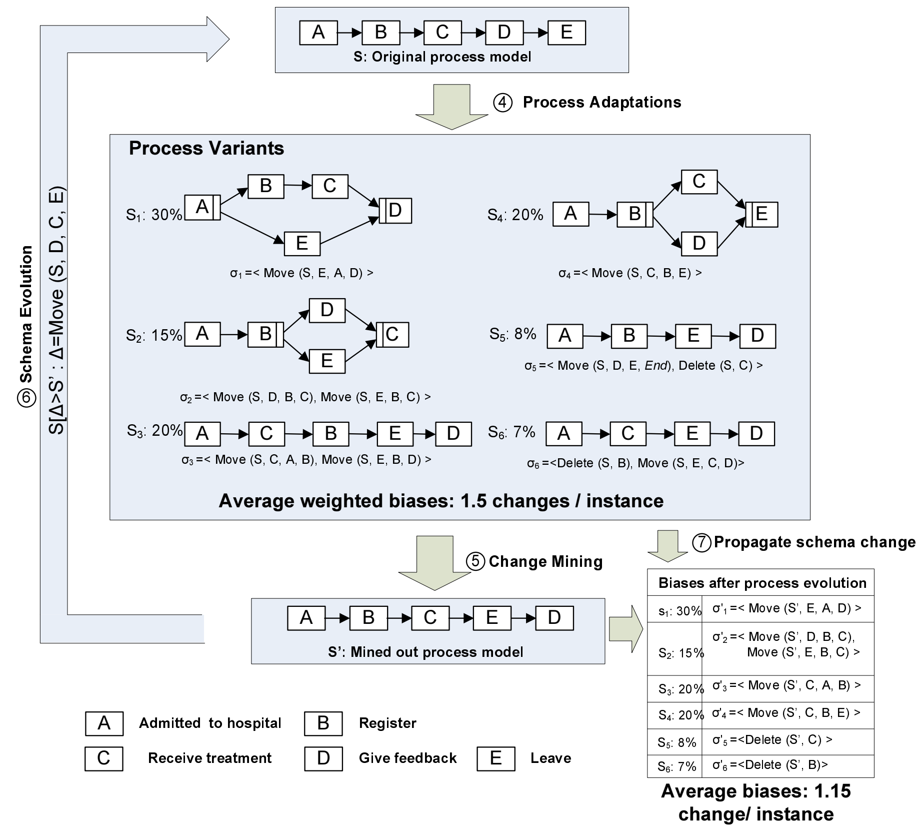
cannot cover all traces the process variants S_1 and S_2 can produce, adapting S' rather than S as the new generic process model requires less efforts for process configuration, since the average weighted distance between S' and the instances running on both S_1 and S_2 is 55% lower than when using S.

Regarding Example 2 from the same figure, activity `X` is only present in model S_2, but not in S_1. When applying traditional process mining focusing on behavior, we obtain process model S (with X being contained in a conditional branch). If focus is on minimizing average change distance, S' will have to be chosen as reference model. Note that in Fig. 4 we have chosen rather simple process models to illustrate basic ideas. Of course, our approach works for process models with more complex structure as well. Further examples are given in Section 5.

Our discussions on the difference between behavioral and structural similarity also demonstrates that current process mining algorithms do not consider structural similarity based on bias and change distance (we will systematically compare our mining approach with existing algorithms in Section 5). 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 set these traces as the input source for process mining algorithms based on traces. Unfortunately, this does also not satisfy our need on reducing biases since it focuses on covering execution behavior as captured in execution log (recall Example 1 and 2). Clearly, enumerating all the traces would be also a tedious and expensive job. For example, if an AND-split and AND-join block contains five branches and each branch contains five activities, the number of traces for such a structure is $(5 \times 5)!/(5!)^5 = 623360743125120$.

5 Example and Evaluation

We now give an example (cf. Fig. 5) to illustrate the basic goal as well as relevant issues and challenges for mining process variants. We further compare process variants mining with traditional process mining techniques in order to quantitatively validate the claim we made in the former sections.



S: Original process model
④ Process Adaptations
Process Variants
S1: 30%
σ1 =< Move (S, E, A, D) >
S4: 20%
σ4 =< Move (S, C, B, E) >
S2: 15%
σ2 =< Move (S, D, B, C), Move (S, E, B, C) >
S5: 8%
σ5 =< Move (S, D, E, End), Delete (S, C) >
S3: 20%
σ3 =< Move (S, C, A, B), Move (S, E, B, D) >
S6: 7%
σ6 =<Delete (S, B), Move (S, E, C, D)>
Average weighted biases: 1.5 changes / instance
⑤ Change Mining
S': Mined out process model
⑦ Propagate schema change
⑥ Schema Evolution
S[Δ>S' : Δ=Move (S, D, C, E)
A Admitted to hospital
B Register
C Receive treatment
D Give feedback
E Leave
Average biases: 1.15 change/ instance

Biases after process evolution	
s_1: 30%	σ'$_1$ =< Move (S', E, A, D) >
S_2: 15%	σ'$_2$ =< Move (S', D, B, C), Move (S', E, B, C) >
S_3: 20%	σ'$_3$ =< Move (S', C, A, B) >
S_4: 20%	σ'$_4$ =< Move (S', C, B, E) >
S_5: 8%	σ'$_5$ =<Delete (S', C) >
S_6: 7%	σ'$_6$ =<Delete (S', B)>

Fig. 5. One example

5.1 Evaluation

Consider Fig. 5 and assume that process model S defines a standard business process. Six different process variants S_1, S_2, S_3, S_4, S_5 and S_6 have been configured out of S, by applying a respective sequence of change operations to S. Furthermore, on each of these process variants several instances are running. A simple statistics is given to show the respective ratio (e.g., 30 % of all process instances are running on model variant S_1 and 8 % on model variant S_5). Based on the relative frequency of each process variant (i.e., its weight), the weighted average number of changes of the six process variants is 1.5. This means we have to perform on average 1.5 changes on the original process model S in order to configure these process variants out of it.

The advantages resulting from the use of high-level change operations have been discussed in Section 3. Here we only give an example. Consider process variants S_1 and S_4 respectively. Though both variants can be configured out of S by applying *one* high-level change operation, the number of corresponding change primitives is *five* (transforming S to S_1) and *seven* (S to S_4) respectively. As behavioral distances, in turn, we obtain *six* (S and S_1) and *two* (S and S_4).

When mining the six process variants by our approach, we obtain process model S' as result (cf. Fig. 5). The discovered model S' is better in the sense that it can reduce the average distance the process variants have with respect to a reference process model (if using S' instead of S' as reference model). In our case, the weighted average number of change operations (i.e., the average distance between reference model and process variants) then decreases from 1.5 to 1.15 (cf. Fig. 5).

The next step in the life cycle is to transform the original process model S to the newly discovered one S'. Obviously, the efforts for process transformation should be minimized. In [20] we have described a method, which allows to create a minimal number of high-level change operations to realize such a transformation. In our example, the change operation transforming S to S' corresponds to $\Delta = move(S, \texttt{D}, \texttt{C}, \texttt{E})$ (i.e., $S[\Delta\rangle S'$). When switching to a new (reference) process model corresponding process variants have to be re-linked to this new model. This also requires the calculation of the biases between new reference model and process variants. We omit further details and refer to [21, 20]. The new biases, representing the differences between S' and the six process variants are listed in the lower table in Fig. 5.

5.2 Comparing Process Variants Mining with Process Mining

Referring to the theoretical comparison between process mining and process variants mining from Section 4, we now compare these two paradigms taking our example from Fig. 5. For this purpose, for each candidate model S_{can}, we assume that it is considered as new reference process model and therefore calculate the biases and distances between S_{can} and the different process variants. For example, consider process variant S_1 from Fig. 5 and the process model S' we obtained from the mining of the process variants (cf. Fig. 5). Obviously, we only need to move activity `E` to the position between `A` and `D` (i.e., $move(S', \texttt{E, A, D})$) in order to transform S' into S_1. Consequently, if we use S' as new reference model and relink the process instances based on variant S_1 to S', each of these instances will have a bias comprising this move operation (i.e., change distance is one). In the same way, we can re-compute biases and distances between S' and the other variants. Based on the weight of each process variant (e.g., 30 % for S_1), the average weighted distance between the new reference model S' and the process variants expresses how close S' is its variants.

There are two groups of process models that serve as candidates for an optimized reference process model. The first group contains all process models we already know, like original process model S and the six process variants S_1, S_2, S_3, S_4, S_5 and S_6 (cf. Fig. 5). Comparing these existing models with the one we obtained through our approach for process variants mining, for example, already shows that it will be not sufficient to simply set the reference model to the most frequently used process variant (S_1 in our example). The second group includes the process models we can discover through mining. Clearly, the process model S' from Fig. 5, which we obtained with our algorithm for process variants mining, belongs to this group. In addition, we consider process models that can be discovered based on traditional process mining techniques [17]. Since a process model can be represented by the set of traces it can produce, we have calculated all traces producible by all process variants in Fig. 5 (see Fig. 6 for all the traces), and then used them as input for different process mining techniques: Alpha algorithm [24], Alpha++ algorithm [25], Heuristics mining [26], and Genetic mining [27]. (These are some of the most well-known algorithms for discovering process models from execution logs). The discovered process models are shown in Fig. 6. Both Alpha and Alpha++ algorithm result in model S_{alp}, whereas Heuristics mining provides model S_{hrs}. We do no consider the model discovered by genetic mining, since it is too different; i.e., genetic mining resulted in a complex structure model with six silent activities (and the distances to each process variant is higher than *three*).

We compute the distances between the candidate models from the two groups and the six process variants. The average weighted distance expresses how close the respective candidate model is to the six process variants. Comparison results are shown in Fig. 7. Fig. 7 shows the distance between each candidate model and each process variant. For example, if we consider process variant S_1 as reference model, we can see that the distance between this model and the variants S_2, S_3, S_4, S_5, and S_6 equals 2. When considering relative frequencies of each variant as weight, we obtain as average weighted distance between S_1 and the six process variants the value 1.4 (cf. column S_1 in Fig. 7). This means that when choosing S_1 as new reference model we need to perform on average 1.4 changes in order to configure the process variants out of S_1. Similarly, we can compute distances for the other candidate models. Altogether, the results from Fig. 7 show that S' (see Fig. 5)), as the process model resulting from the the method we suggested, has the shortest average weighted distance to the different process variants. This means, setting S' as the new reference process model would

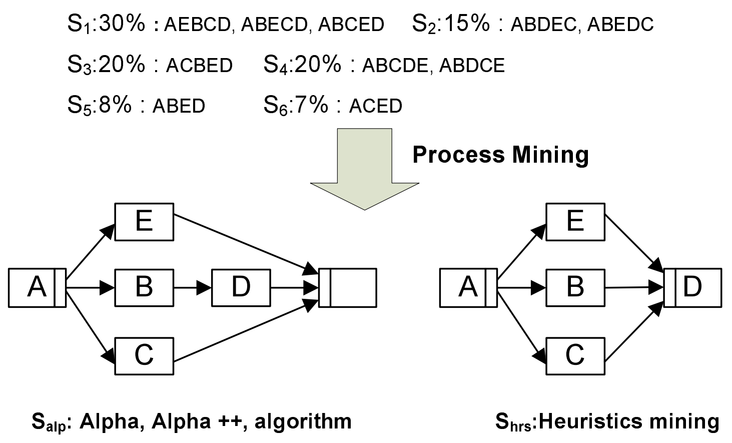


Fig. 6. Process Models Resulting from Process Mining

require lowest efforts for configuring the variants; i.e. we only need to perform on average 1.15 changes to configure a process variant out of S'. Note that the process models S_{alp} and S_{hrs}, as discovered by the process mining algorithms (based on traces), show the largest average distance to the six process variants. This also complies to our analysis from Section 4.

S_{hrs}	S_{alp}	S'	S	S_6	S_5	S_4	S_3	S_2	S_1	**Reference model**
1	2	1	1	2	2	2	2	2	0	**Distance to S_1 (30%)**
2	2	2	2	2	2	2	2	0	2	**Distance to S_2 (15%)**
2	2	1	2	1	1	2	0	2	2	**Distance to S_3 (20%)**
2	2	1	1	2	2	0	2	2	2	**Distance to S_4 (20%)**
2	2	1	2	2	0	2	1	2	2	**Distance to S_5 (8%)**
3	3	1	2	0	2	2	1	2	2	**Distance to S_6 (7%)**
1.77	2.07	1.15	1.5	1.66	1.64	1.6	1.45	1.7	1.4	**Average distance**

Fig. 7. The number of biases when adapting different models

Comparison results do not imply that process variants mining is better than process mining. Each of them has different inputs and goals. Compared to process mining, which tries to discover the underlying process model by learning from the behavior of a system, process variants mining focus on discovering a genetic process reference model which is easy configurable for process variants. If we use the process mining evaluation criteria to measure the result of process variants mining, the discovered process model S' (cf. Fig. 5) is also not good in terms of behavior, since behavior of S' is very limited according to the process evaluation framework [28].

6 Related Work

A variety of techniques for process mining has been suggested in literature [17, 26, 27, 24]. As illustrated in this paper, traditional process mining is different from process variants mining due to its different goal and inputs. A few techniques have been proposed to learn from process variants by mining change primitives. [29] measures process model similarity based on change primitives and suggests mining techniques using this measure. However, this approach does not consider important features of our process meta model; e.g., it is unable to deal with silent activities or loop backs, and does also not differentiate AND- and OR-splits. Similar techniques for mining change primitives exist in the fields of rule mining [30] and maximal sub-graph mining as known from graph theory [31]; here common edges between different nodes are discovered to construct a common sub-graph from a set of graphs. To mine high level change operations, [13] presents an approach based on process mining techniques, i.e., the input consists of a change log, and process mining algorithms are applied to discover the execution sequences of the changes (i.e., the change meta process). However it simply considers each change as an individual operation so that the result is more like a visualization of changes rather than mining them. None of the discussed approaches aims at creating a generic process model, which allows for easy and optimized configuration for process variants.

7 Summary and Outlook

We have motivated the need for process variants mining, discussed its major goals as well as relevant issues, and elaborated its differences when compared to traditional process mining. We believe that process variants mining will contribute to business intelligence and allow to learn from adapted processes and services respectively. Basically, as input our approach takes a collection of process variants, and then produces a generic process model as output which covers these variants best; i.e., the generic model is chosen in a way such that the average bias between generic model and process variant becomes minimal. Note that this will reduce adaptation and configuration costs as well. As our evaluation has shown both process variants mining and process mining are needed to enable full process lifecycle support in PAIS.

We have compared process variants mining with process mining by means of an example. This comparison shows that traditional process mining does not satisfy the need for deriving a process model which is easy configurable. This justifies the efforts for designing specific algorithms for process variants mining, which we will present in other papers. Our results are promising, but there are still many research questions left open. For example, it seems even better to integrate process mining with process variants mining such that we can consider both execution behavior (as captured in execution logs) and past process changes (as stored in change logs) in order to learn from process executions.

References

1. T. Curran, G. Keller, A. Ladd: *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model.* Enterprise Resource Planning Series. Prentice Hall PTR, 1997
2. T.H. Davenport: *Mission Critical - Realizing the Promise of Enterprise Systems.* Harvard Business School
3. G. Halmans, K. Kohl: *Communicating the Variability of a Software Procut Family to Customers.* Software and Systems Modeling, 2(1): 15-36, 2003
4. D.L. Parnas (1994): *Software Aging.* Proc. 16th Int'l Conf. on Software Engineering, Sorrento, Italy, May 1994, pp. 279-287
5. R. Sabherwal, Y.E. Chan: *Alignment between business and IS strategies: A study of prospectors, analyzers, and defenders.* Inf. Systems Research, 12:11-33, 2001
6. B. Weber, S. Rinderle, M. Reichert: *Change Patterns and Change Support Features in Process-Aware Information Systems.* CAiSE'07, LNCS 4495, Norway, 2007, pp. 574-588
7. S.B.Rinderle, M. Jurisch, and M. Reichert: *On Deriving Net Change Information From Change Logs - The DELTALAYER-Algorithm.* BTW'07, Germany. 2007 pp. 364-381.
8. M. Reichert and P.Dadam. *ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control.* Journal of Intelligent Information Systems, 10(2):93–129, 1998.
9. M. Reichert, S. Rinderle, and P. Dadam: *On the common support of workflow type and instance changes under correctness constraints.* CoopIS'03, LNCS 2888, pages 407–425, Catania, Italy, 2003.
10. M.Weske: *Formal foundation and conceptual design of dynamic adaptations in a workflow management system.* HICSS-34, pp 7051, 2001.
11. M.Kradolfer, A.Geppert. Dynamic workflow schema evolution based on workflow type versioning and workflow migration. CoopIS'99, Edinburgh 1999, pp. 104-114
12. 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,
13. C.W. Günther, S. Rinderle, M. Reichert, and W.M.P. van der Aalst. *Change Mining in Adaptive Process Management Systems.*CoopIS'06, LNCS 4275, pp 309-326. Berlin, 2006.
14. A.Wombacher, M.Rozie: Evaluation of Workflow Similarity Measures in Service Discovery. Service Oriented Electronic Commerce 2006: 51-71
15. L. I. Levenshtein. *Binary codes capable of correcting deletions, insertions, and reversals,.*Soviet PhysicsCDoklady, 10(8):707C710, 1966
16. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, A. J. M. M. Weijters: *Workflow mining: a survey of issues and approaches.* Data & Knowledge Engineering. V47-2. pp 237-267. 2003
17. http://ga1717.tm.tue.nl/wiki/
18. W.M.P. van der Aalst and T. Basten:*Inheritance of Workflows: An Approach to Tackling Problems Related to Change.* Theoretical Computer Science, 270(1-2):125-203, 2002

19. W. M. P. van der Aalst, T. Basten. *Identifying Commonalities and Differences in Object Life Cycles Using Behavioral Inheritance.* ICATPN '01. LNCS 2075. pp 32-52. 2001
20. C.Li, M.Reichert, A.Wombacher. *Process Similarity Based on High Level Change Operations.* CTIT Technical Report, University of Twente, The Netherlands.
21. S.Rinderle: *Schema Evolution in Process Management Systems.* Ph.D thesis, Univ. of Ulm, 2004
22. J.Hidders, M.Dumas, W. M.van der Aalst, A. H. ter Hofstede, and J.Verelst. *When are two workflows the same?.* 2005 Australasian Symposium on theory of Computing - Vol 41. ACM vol. 105. 3-11. 2005
23. B.Weber, M.Reichert, W.Wild: *Case-Base Maintenance for CCBR-Based Process Evolution.* ECCBR06, LNCS 4106. pp 106-120. 2006
24. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. *Workflow Mining: Discovering Process Models from Event Logs.* IEEE Transactions on Knowledge and Data Engineering, 16(9):1128C1142, 2004.
25. L. Wen, J. Wang, and J.G. Sun. *Detecting Implicit Dependencies Between Tasks from Event Logs.* In APWeb 2006, LNCS 3841, pp 591C603, 2006.
26. A.J.M.M. Weijters and W.M.P. van der Aalst. *Rediscovering Workflow Models from Event-Based Data using Little Thumb.* Integrated Computer-Aided Engineering, 10(2):151C162, 2003.
27. A.K. Alves de Medeiros. *Genetic Process Mining.* PhD thesis, Eindhoven University of Technology, Eindhoven, 2006.
28. A. Rozinat, A.K. Alves de Medeiros, C.W. Günther, A.J.M.M. Weijters, and W.M.P. van der Aalst: *Towards an Evaluation Framework for Process Mining Algorithms.* BPM-07-06, BPMcenter.org, 2007.
29. J.Bae, L. Liu, J. Caverlee, W. B. Rouse: *Process Mining, Discovery, and Integration using Distance Measures* ICWS06, pp. 479-488, 2006
30. P. Tan, M. Steinbach, V. Kumar: *Introduction to Data Mining.* Addison Wesley, 2006.
31. K.H.Rosen: *Discrete Mathematics and Its Applications.* fifth edition. McGraw-Hill, 2003.
32. Workflow Management Coalition wWorkflow Standard Process Definition Interface – XML Process Definition Language, Document Number WFMC-TC-1025. V 1.13.