Mining Based on Learning from Process Change Logs

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. Recently, a new generation of flexible PAIS was introduced, which additionally allows for dynamic process changes. This, in turn, leads to a large number of process variants, which are created from the same original 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 costs of process change and need for process adaptations will decrease. We compare process variant mining with conventional process mining techniques, and show that it is additionally needed to learn from process changes.

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 meet this fundamental requirement. It is deployed in different companies, domains, and countries, and therefore tends to be too generic and rigid. This in turn, causes huge customization efforts at the site of software buyers that exceed the price for software licenses by factor five to ten. Major progress has been achieved by shifting from function- to process- and service-centered software design. Along this trend a variety of process support paradigms (e.g., process orchestration, process choreography) and corresponding specification languages have emerged. In addition, different approaches for flexible and adaptive processes exist [1, 3]. Generally, process adaptations are not

* Work done in the MinAdept project which is Supported by the Netherlands Organization for Scientific Research (NWO) under contract number 612.066.512

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 process instances, it must be possible to dynamically adapt their structure (i.e., to insert, delete or move activities).

Obviously, the ability to adapt and configure processes at the different levels will result in a collection of process model variants 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 in Fig. 1b. Generally, a large number of process model variants (*process variants* for short) derived from the same original process model might exist [16].

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 leads to degeneration and divergence of the respective process models, 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.

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 [8]. Different mining techniques like alpha algorithm [8], heuristics mining [10], and genetic mining [11] have been proposed in this context. When considering the extensive research on process mining, only little work has dealt with *process variant mining* so far. Here the overall goal is to evolve a process model over time by learning from the changes applied to corresponding model instances in the past. By learning from these

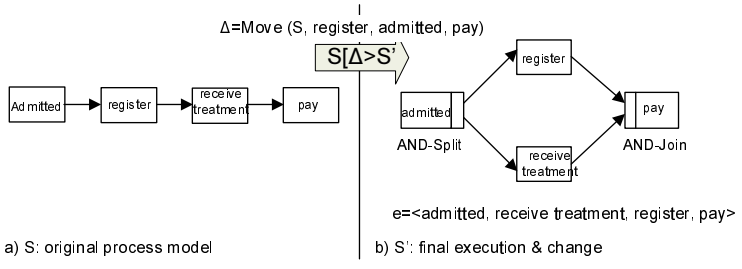

Fig. 1. Original Process Model S and Process Variant S'

model variants and by merging them into a generic process model, efforts for future process model configurations as well as adaptations can be reduced. This paper deals with the following research question:

Why is process variant 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 (see [15]), and will also utilize them for comparing variants mining with conventional process mining.

Sec. 2 gives background information needed for the understanding of this paper. Sec. 3 discusses why process changes should be expressed in terms of high-level change operations. Sec. 4 discusses major goals of process variant mining and shows why it is different from traditional process mining. In Sec. 5 we present a concrete example to elaborate these differences and in Sec. 6, we discuss related work. The paper concludes with a summary in Sec. 7.

2 Backgrounds

We first introduce basic notions needed in the following:

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) [3], 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 (cf. Fig. 1).

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 [3]. 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) 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 $i = \{1, \ldots n\}$. Further $|\sigma| = n$.*

Examples of high-level change operations include *insert activity*, *delete activity*, and *move activity* as implemented in the ADEPT change framework [3]. 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 [3]. 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 [1]. 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 [6]. 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 [1].

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 S into 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 high-level change operations needed for transforming S into S'. The corresponding sequence of change operations is denoted as *b*ias between S and S'.[4] Each change operation the same weight, i.e., we do not consider some changes being more important or more expensive than others. Generally, the distance between two process models measures the complexity for model transformation or configuration. As example take Fig. 1. Here, 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'. We describe a method to compute biases in [7].

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 $\mathcal{T}_S$. We consider two process models as being the same if they are *trace equivalent*, i.e., $S \equiv S'$ if and only if $\mathcal{T}_S \equiv \mathcal{T}_{S'}$.

3 On Representing Process Changes

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 model of a process instance is given, it will be 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 model S, we obtain variant S' (i.e. $S[\sigma\rangle S'$) with change log $\sigma =< move(S, reveive\ treatment, admitted, pay) >$. Assume that this variant represents an instance-specific schema and that the trace of the particular instance is $\{admitted, receive\ treatment, register, pay\}$. If S and

[4] Generally, it is possible to have more than one minimal set of change operations to transform S into S', i.e., given two process models S and S' their bias is not necessarily unique. A detailed discussion of this issue can be found in [6, 7].

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; e.g., 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'.

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. 2. 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). 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 `C` to the position between `A` and `B`, resulting in process variant S_1, i.e., $S[\Delta_1\rangle S_1$ with $\Delta_1 = move(S, \texttt{C,A,B})$. Δ_2, in turn, moves `A` to the position between `B` and `C`, i.e., $S[\Delta_2\rangle S_2$ with $\Delta_2 = move(S, \texttt{A,B,C})$. Note that Fig. 2 additionally depicts the change primitives representing the snapshot differences between original model S and variants S_1 and S_2 respectively.

In comparison with low-level primitives, the use of high-level change operations offers the following advantages: First, 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' [3]. This also applies to our example from Fig. 2. 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.

Second, 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

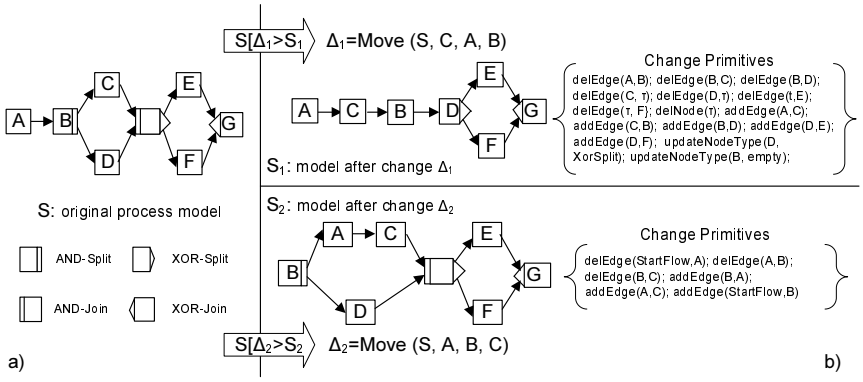

Fig. 2. High-Level Change Operation vs. Change primitives

pattern. As example take Δ_1 from Fig. 2. 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. 2, the movement of activity `C` is accompanied by the deletion of silent activity τ, since the parallel branching is no longer needed.

Third, another important aspect concerns the number of change operations needed to transform a process model S into an 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* primitives, while the second change Δ_2 can be realized with *6* primitives. This 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 variant mining on high-level change operations.

Finally when representing model changes by means of high-level operations, we can always derive corresponding change primitives, but not the other way around; i.e., with respect to 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 primitives can be easily obtained by determining which nodes and edges have been deleted or added [2]. By contrast, if we only store low-level primitives, computing the corresponding high-level change operation will be difficult; e.g., how to determine that the 15 primitives needed to transform S into S_1 only represent one single high-level change?

3.3 How Do High-level Changes Influence Process Behavior?

[5] 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 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 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 $\Delta_2 = move(S, pay, admitted, receivetreatment)$ 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.

4 Mining Process Variants: Goals and Comparison with Process Mining

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 variants can be efficiently configured out of the generic model. We measure 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 variant mining with traditional process mining. Obviously, input data for process and process variant 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 [8] or generic mining [11] can be applied to our problem as well. For example, we could derive all traces producible by a given collection of process variants [5] 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.

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 [5, 8]. 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 (cf. Def. 2). 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. 3 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 [8], 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 instances to it, all instances running on S_1 or S_2 will have a non-empty bias. 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 $\sigma_1 = move(S, \texttt{B}, \texttt{A}, \texttt{C})$ and $S[\sigma_2\rangle S_2$ with $\sigma_2 = move(S, \texttt{B}, \texttt{C}, \texttt{D})$ (cf. Def. 2). Using the number of instances as weight for each variant, the average

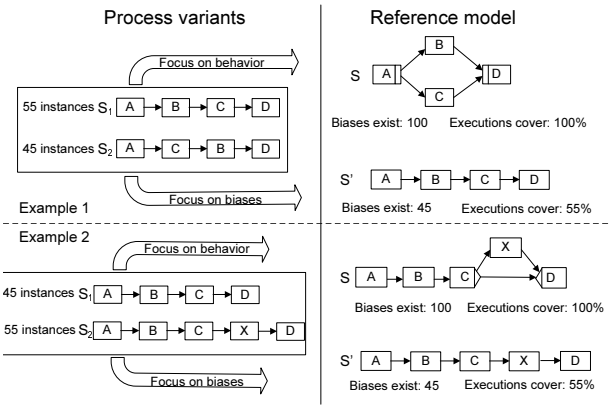

Fig. 3. Mining focusing either on Behavior or on Minimization of Biases

weighted distance between S and S_1, 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.

By contrast, if we focus on bias, 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' cannot 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 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 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 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. 3 we have considered rather simple process models to illustrate basic ideas. Of course, our approach works for process models with more complex structure as well (see [15] for details).

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.

5 Example and Evaluation

Consider Fig. 4 and assume that process model S defines a standard business process. Six different variants S_1, S_2, S_3, S_4, S_5 and S_6 have been configured out of 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 instances are running on S_1 and 8 % on S_5). We therefore compute the distance and biases between the reference model and the six variants. For example, if we want to configure the reference model S into S_1, we need to perform one change operation $move(S, \texttt{E, A, D})$, i.e., distance between S and S_1 is one. The distance and bias between S and the process variants are shown in Fig. 4.

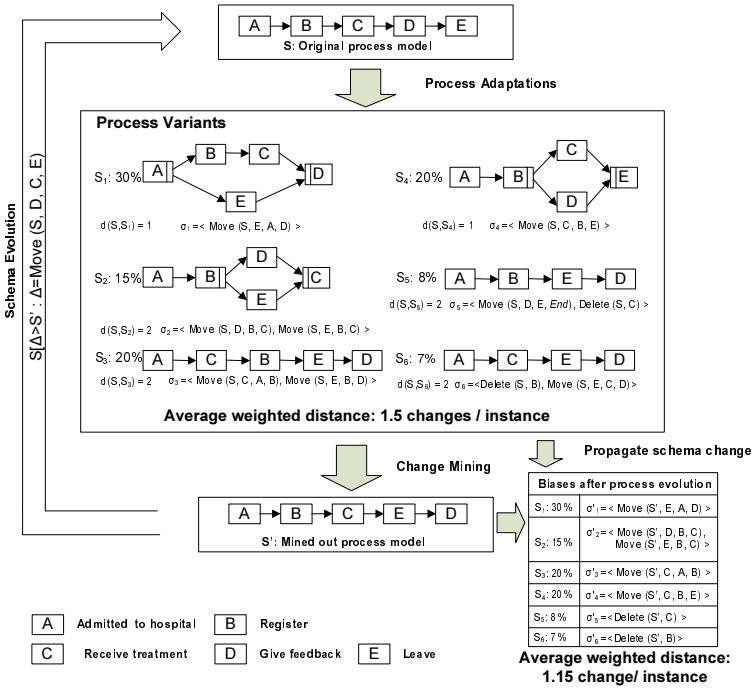

Fig. 4. One example

Based on the relative frequency of each process variant (i.e., its weight), the weighted average number of changes of the six variants is 1.5. This means we have to perform on average 1.5 changes on the original model S in order to configure these variants out of it. When mining the six process variants by our approach (see [15]), we obtain process model S' as result (cf. Fig. 4). The discovered model S' is better in the sense that it can reduce the average distance the variants have with respect to a reference process model (if using S' instead of S' as reference model). In our case, weighted average distance between the reference model and the variants decreases from 1.5 to 1.15 (cf. Fig. 4 for the biases). It means we can reduce the configuration effort by replacing S with S'.

Referring to the theoretical comparison between process mining and process variant mining from Section 4, we now compare these two paradigms taking our example from Fig. 4. For this purpose, for each candidate model S_{can}, we assume that it is considered as new reference process model and therefore calculate average weighted distance between S_{can} and the six process variants. For example, if we choose S_1 as the reference model, (i.e., $S_{can} = S_1$), the distances between S_{can} and $S_1 - S_6$ will be 0, 2, 2, 2, 2 and 2. And based on the weight of each variant, we obtain average weighted distance between S_{can} and variants $S_1 - S_6$ is 1.4. We therefore compare the candidate process models by comparing their average weighted distance to the six 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 model S and the six variants S_i, $i = 1 \ldots 6$ (cf. Fig.

4). Comparing these models with the one we obtained through our approach for process variant mining, shows that it is 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, model S' from Fig. 4 belongs to this group. In addition, we consider process models that can be discovered based on traditional process mining techniques [8]. 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. 4 (see Fig. 5), and then used them as input for different process mining techniques: Alpha algorithm [8], Alpha++ algorithm [9], Heuristics mining [10], and Genetic mining [11]. (These are some of the most well-known algorithms for discovering process models from execution logs). The discovered process models are shown in Fig. 5. 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 distance to each process variant is higher than *three*).

We compute the average weighted distances between the candidate models and the six process variants. Fig. 6 shows the distance between each candidate model and process variant, e.g., if we consider variant S_1 as candidate reference model, we can see that the distance between this model and variants S_2, S_3, S_4, S_5, and S_6 equals 2, and the average weighted distance between this candidate and the six variants is 1.4 (cf. column S_1 in Fig. 6). Results from Fig. 6 show that S' (see Fig. 4)), the process model resulting from the method we suggest [15], has the shortest average weighted distance to the different variants, i.e., setting S' as new reference process model would 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 models S_{alp} and S_{hrs}, as discovered by the process mining algorithms (based on traces), show larger distances to the variants. This also complies to our analysis from Sec. 4.

Comparison results do not imply that process variant 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 variant mining focus on discovering a generic reference model which is easy configurable for process variants. If we use process mining evaluation criteria to measure the result of process variant mining, the discovered process model S' (cf. Fig. 4) will be also not good in terms of behavior, since behavior of S' is limited. However, we do not consider it as a critical

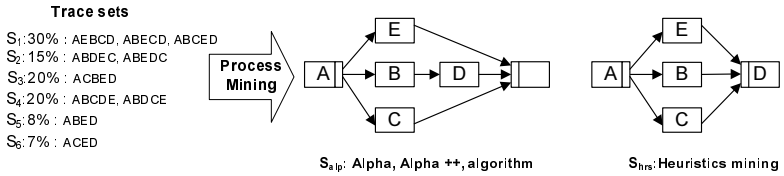

Fig. 5. Process Models Resulting from Process Mining

S_{hrs}	S_{alp}	S'	S	I_6	I_5	I_4	I_3	I_2	I_1	Reference model
1	2	1	1	2	2	2	2	2	0	Distance to I_1 (30%)
2	2	2	2	2	2	2	2	0	2	Distance to I_2 (15%)
2	2	1	2	1	1	2	0	2	2	Distance to I_3 (20%)
2	2	1	1	2	2	0	2	2	2	Distance to I_4 (20%)
2	2	1	2	2	0	2	1	2	2	Distance to I_5 (8%)
3	3	1	2	0	2	2	1	2	2	Distance to I_6 (7%)
2	2	1	2	2	2	2	1	2	1	Average distance

Fig. 6. The number of biases when adapting different models

limitation, since in most cases it is the process variant rather than the reference model which will act as the process model for instance executions.

6 Related Work

A variety of techniques for process mining has been suggested in literature [10, 11, 8]. As illustrated in this paper, traditional process mining is different from process variant mining due to its different goal and inputs. Some improvements of process mining algorithms have been made to enhance their performance (e.g., DWS Mining [17]), but they are still different from variant mining due to their different goals and inputs. A few techniques have been proposed to learn from process variants by mining change primitives. [12] 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 [13] and maximal sub-graph mining as known from graph theory [14]; 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, [4] 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 execution sequences of the changes (i.e., the change meta process). However, it simply considers each change as individual operation so that the result is more like a visualization of changes. None of the discussed approaches aims at creating a generic process model, which allows for easy and optimized configuration of process variants.

7 Summary and Outlook

We have motivated the need for process variant mining, discussed its major goals and issues, and elaborated its differences when compared to traditional process mining. We believe that process variant mining will contribute to business intelligence and allow to learn from adapted processes respectively. Basically, as input our approach takes a collection of process variants (i.e., process models), 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.

We have compared process variant mining with process mining by means of an example. This comparison indicates that traditional 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 variant mining, which we present in other papers. Our results are promising, but there are still many research questions left open; e.g., it seems even better to integrate process mining with process variant mining such that we can consider both execution behavior (i.e., execution logs) and past process changes (i.e., change logs) to learn from process executions. In future work, we will also assign weight for different change operations, i.e., we will be able to express that some changes are more important or expensive to configure than others.

References

1. B. Weber, S. Rinderle, M. Reichert: *Change Patterns and Change Support Features in Process-Aware Information Systems.* CAiSE'07, LNCS 4495, 2007, pp. 574-588
2. S. Rinderle, M. Jurisch, and M. Reichert: *On Deriving Net Change Information From Change Logs - The DELTALAYER-Algorithm.* BTW'07, 2007 pp. 364-381.
3. M. Reichert and P. Dadam. *ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control.* Journal of Intelligent Info. Sys., 10(2):93-129, 1998.
4. C.W. Günther, S. Rinderle, M. Reichert, W.M.P. van der Aalst. *Change Mining in Adaptive Process Management Systems.*CoopIS'06, LNCS 4275, pp 309-326. 2006.
5. A.Wombacher, M.Rozie: Evaluation of Workflow Similarity Measures in Service Discovery. Service Oriented Electronic Commerce 2006: 51-71
6. W.M.P. van der Aalst and T. Basten:*Inheritance of Workflows: An Approach to Tackling Problems Related to Change.* Theoretical CS, 270(1-2):125-203, 2002
7. C. Li, M. Reichert, and A. Wombacher: *On measuring process model similarity based on high-level change operations.* In ER 08, Barcelona, 2008.
8. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. *Workflow Mining: Discovering Process Models from Event Logs. TKDE*, 16(9):1128-1142, 2004.
9. L. Wen, J. Wang, and J.G. Sun. *Detecting Implicit Dependencies Between Tasks from Event Logs.* In APWeb 2006, LNCS 3841, pp 591-603, 2006.
10. A.J.M.M. Weijters and W.M.P. van der Aalst. *Rediscovering Workflow Models from Event-Based Data using Little Thumb.* Int. CA. Eng., 10(2):151-162, 2003.
11. A.K. Alves de Medeiros. *Genetic Process Mining.* PhD thesis, Eindhoven University of Technology, Eindhoven, 2006.
12. J.Bae, L. Liu, J. Caverlee, W. B. Rouse: *Process Mining, Discovery, and Integration using Distance Measures* ICWS06, pp. 479-488, 2006
13. P. Tan, M. Steinbach, V. Kumar: *Introduction to Data Mining.* Add-Wesley, 2006.
14. K.H.Rosen: *Discrete Mathematics and Its Applications.* McGraw-Hill, 2003.
15. C. Li, M. Reichert, and A. Wombacher: *Discovering reference process models by mining process variants.* In ICWS'08, Beijing, 2008.
16. A. Hallerbach, and T. Bauer, and M. Reichert: *Managing Process Variants in the Process Lifecycle.*ICEIS'08, pp 154-161, 2008.
17. A.K.A de Medeiros, A. Guzzo G. Greco W.M.P. van der Aalst, A.J.M. M. Weijters, B.F. van Dongen, D. Saccà: *Process Mining Based on Clustering: A Quest for Precision.* BPM'07 workshop. pp 17-29, 2008.