Discovering Reference Process Models by Mining Process Variants

Chen Li
University of Twente
The Netherlands
lic@cs.utwente.nl

Manfred Reichert
University of Ulm
Germany
manfred.reichert@uni-ulm.de

Andreas Wombacher
University of Twente
The Netherlands
a.wombacher@utwente.nl

Abstract

Recently, a new generation of adaptive Process-Aware Information Systems (PAIS) has emerged, which allows for dynamic process and service changes (e.g., to insert, delete, and move activities and service executions in a running process). This, in turn, has led to a large number of process variants derived from the same model, but differing in structure due to the applied changes. Generally, such process variants are expensive to configure and difficult to maintain. This paper provides a sophisticated approach which fosters learning from past process changes and allows for mining process variants. As a result we obtain a generic process model for which the average distance between this model and the respective process variants becomes minimal. By adopting this generic model in the PAIS, need for future process configuration and adaptation decreases. We have validated the proposed mining method and implemented it in a powerful proof-of-concept prototype.

1 Introduction

In today's dynamic business world, success of an enterprise increasingly depends on its ability to react to changes in its environment in a quick, flexible and cost-effective way. Along this trend a variety of process and service support paradigms as well as corresponding specification languages (e.g., WS-BPEL, WS-CDL) have emerged. In addition, different approaches for flexible and adaptive processes exist [9, 11]. Generally, process and service adaptations are not only needed for configuration purposes at buildtime, 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 (e.g. to insert, delete or move activities during runtime).

In response to this need adaptive process management technology has emerged [17]. It allows to adapt and configure 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 from each other in their structure. Generally, a large number of process variants may exist in a Process-Aware Information System (PAIS) [8]. As example take an automotive supply chain where a supplier of car parts requires different process variants per car manufacturer and potentially per car model. This results in a high number of process variants, which is comparable to the number of customers of the supplier. To support the different choreographies the supplier requires the same number of orchestrations, i.e., process variants which have to be modeled and maintained. This number further increases when considering concrete cases as well. Here ad-hoc deviations from the standard processes occur frequently at process instance level, resulting in a multitude of process variants.

In most approaches which allow to adapt and configure process models, the resulting process variants have to be maintained separately. Then even simple changes (e.g. due to new laws) 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 [4].

Though considerable efforts have been made to ease process configuration and customization [9, 11, 4], we do not yet utilize the knowledge resulting from these process adaptations. Fig. 1 describes the goal of this paper. We aim at learning from past process changes by "merging" 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 BPEL) and (2) all activities in a process model have unique labels, this paper deals with the following fundamental research question: *Given a set of process variants, how to derive a reference process model out of them, such that the average dis-*

[0]Supported by the Netherlands Organization for Scientific Research (NWO) under contract number 612.066.512

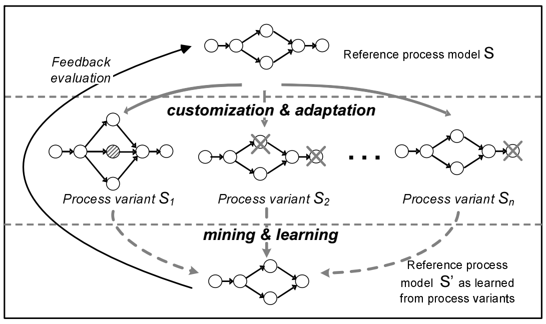

Feedback evaluation
Reference process model S
customization & adaptation
Process variant S_1
Process variant S_2
Process variant S_n
mining & learning
Reference process model S' as learned from process variants

Figure 1. Mining a new reference model with less expected changes

tance between the reference model and the process variants becomes minimal?

The distance between the reference process model and a process variant is measured by the number of high-level change operations (e.g., to insert, delete or move activities [9]) needed to transform the reference model into the variant. Furthermore, change distance directly represents the efforts needed for process adaptation and customization. Obviously, the challenge is to find the "best" reference model, i.e., the one with minimal average distance to the variants.

Sec. 2 gives background information needed for understanding this paper. In Sec. 3 we introduce a method to represent process models in a way such that they can be mined effectively. Sec. 4 presents our algorithm for mining process variants. We validate it in Sec. 5 and sketch a proof-of-concept prototype in Sec. 6. Sec. 7 discusses related work. We conclude with a summary in Sec. 8.

2 Backgrounds

We first introduce basic notions needed in the following:

Process Model: Let $\mathcal{P}$ denote the set of all sound process models. A particular *process model* $S = (N, E, \ldots) \in \mathcal{P}$ is defined as well-structured Activity Net [9]. N constitutes the set of process activities and E the set of control edges (i.e., precedence relations) linking them. To limit the scope, we assume Activity Nets to be block-structured (like in BPEL). An example is depicted in Fig. 2.

Process change: We assume that a process change is accomplished by applying a sequence of change operations to a given process model S over time [9]. Such change operations modify the initial process model by altering the set of activities and their order relations. Thus, each application of a change operation results in a new process model. We define *process change* and *process variants* as follows:

Definition 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 process 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. We also denote S' as **variant** of 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 $i \in \{1, \ldots n\}$.*

Examples of high-level change operations include *insert activity*, *delete activity*, and *move activity* as implemented in the ADEPT change framework [9]. While *insert* and *delete* modify the set of activities in the process model, *move* changes activity positions and thus the structure of the process model. For example, operation $move(S, \texttt{A}, \texttt{B}, \texttt{C})$ 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. Issues concerning the correct use of these operations, their generalizations, and formal pre-/post-conditions are described in [9]. 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 [17]. 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 [14]. We refer to ADEPT since it covers by far most high-level change patterns and change support features when compared to other approaches [17].

Definition 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., $d_{(S,S')} := min\{|\sigma| \mid \sigma \in \mathcal{C}^* \wedge S[\sigma\rangle S'\}$. 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 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 *bias* between S and S'.[1] Usually, the distance between two process models measures the complexity for model transformation or configuration. As example take Fig. 3. Here, distance between model S and variant S_1 is *one*, since we only need to perform one change operation $move(S, \texttt{E}, \texttt{A}, \texttt{D})$ to transform S into S_1 [7]. In general, determining the bias and distance between two process models has complexity at $\mathcal{NP}$ level [7].

Here, we use high-level change operations rather than change primitives (i.e., elementary changes like adding/removing nodes and edges) to measure the distance

[1]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 process models S and S' their bias needs not to be unique. A detailed discussion of this issue can be found in [14, 7].

between process models. This allows to guarantee soundness of process models and also provides a more meaningful measure for distance [7].

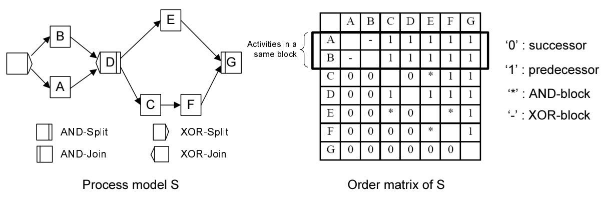
Trace: A *trace* t on process model $S = (N, E, \dots)$ denotes a valid and complete execution sequence $t \equiv < a_1, a_2, \dots, a_k >$ of activities $a_i \in N$ on S according to the control flow set out by S. All traces process model S can produce are summarized in trace set $\mathcal{T}_S$. $t(a \prec b)$ is denoted as precedence relation between activities a and b in trace $t \equiv < a_1, a_2, \dots, a_k >$ iff $\exists i < j : a_i = a \wedge a_j = b$. Here, we only consider traces composing 'real' activities, but no events related to silent ones (i.e., activity nodes which contain no operation and exist only for control flow purpose [7]). Finally, we will consider two process models being the same if they are *trace equivalent*, i.e., $S \equiv S'$ iff $\mathcal{T}_S \equiv \mathcal{T}_{S'}$.

3 Represent Process Models as Order Matrices

Theoretical backgrounds of high-level change operations have been extensively discussed in ADEPT [9]. One key feature of our ADEPT change framework is to maintain the structure of the unchanged parts of a process model [9]. For example, if we delete an activity this will neither influence the successors nor predecessors of this activity, and therefore also not their order relations. To incorporate this feature in our approach, rather than only looking at direct predecessor-successor relationships between two activities (i.e., control edges), we consider the transitive control dependencies between all activity pairs; i.e., for process model $S = (N, E, \dots) \in \mathcal{P}$, for every pair of activities $a_i, a_j \in N$, $a_i \neq a_j$ their order relations compared to each other are examined. Logically, we determine order relations by considering all traces the respective process model can produce (cf. Sec. 2). Results are aggregated in a matrix $A_{|N| \times |N|}$, which considers four types of control relations (cf. Def. 3):

Definition 3 (Order matrix) *Let $S = (N, E, \dots) \in \mathcal{P}$ be a process model with $N = \{a_1, a_2, \dots, a_n\}$. Let further $\mathcal{T}_S$ denote the set of all traces producible on S. Then: Matrix $A_{|N| \times |N|}$ is called* ***order matrix*** *of S with A_{ij} representing the order relation between activities a_i,$a_j \in N$, $i \neq j$ iff:*

- *A_{ij} = '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 traces containing activities a_i and a_j, a_i always appears BEFORE *a_j, we denote A_{ij} as* ***'1'****, i.e., a_i always precedes of a_j in the flow of control.*
- *A_{ij} = '0' iff ($\forall t \in \mathcal{T}_S$ with $a_i, a_j \in t \Rightarrow t(a_j \prec a_i)$)*
  If for all traces containing activities a_i and a_j, a_i always appears AFTER *a_j, we denote A_{ij} as a* ***'0'****, i.e., a_i always succeeds of a_j in the flow of control.*
- *A_{ij} = '*' iff ($\exists t_1 \in \mathcal{T}_S$, with $a_i, a_j \in t_1 \wedge t_1(a_i \prec a_j)$) $\wedge$ ($\exists t_2 \in \mathcal{T}_S$, with $a_i, a_j \in t_2 \wedge t_2(a_j \prec a_i)$)*
  If there exists at least one trace in which a_i appears before a_j and another trace in which a_i appears after a_j, we denote A_{ij} as '', i.e., a_i and a_j are contained in different parallel branches.*
- *A_{ij} = '-' iff ($\neg\exists t \in \mathcal{T}_S : a_i \in t \wedge a_j \in t$)*
  If there is no trace containing both activity a_i and a_j, we denote A_{ij} as **'-'***, i.e., a_i and a_j are contained in different branches of a conditional branching.*



	A	B	C	D	E	F	G
A		-	1	1	1	1	1
B	-		1	1	1	1	1
C	0	0		0	*	1	1
D	0	0	1		1	1	1
E	0	0	*	0		*	1
F	0	0	0	0	*		1
G	0	0	0	0	0	0	

Figure 2. Process model and its order matrix

Fig. 2 shows an example. Besides control edges, which express direct predecessor-successor relationships, model S also contains four kinds of control connectors: AND-Split and AND-Join (corresponding to *flow* in BPEL), XOR-Split and XOR-join (corresponding to *switch* or *pick* in BPEL). The order matrix can represent all these relationship. For example, activities A and B never appear in the same trace since they are contained in different branches of an XOR block. Therefore, we assign '-' to matrix element A_{AB}. Similarly, we obtain the relation for each pair of activities. The main diagonal of the matrix is empty since we do not compare an activity with itself.

Under certain conditions, an order matrix uniquely represents the process model it was created from. This is stated by Theorem 5. Before giving this theorem, we need to define the notion of *substring of trace*:

Definition 4 (Substring of trace) *Let $S \in \mathcal{P}$ be a process model and let $t, t' \in \mathcal{T}_S$ be two traces on S. We denote t as sub-string of t' iff [$\forall a_i$, $a_j \in t$, $t(a_i \prec a_j) \Rightarrow a_i$, $a_j \in t' \wedge t'(a_i \prec a_j)$] and [$\exists a_k \in N$: $a_k \notin t \wedge a_k \in t'$].*

Theorem 5 *Let $S, S' \in \mathcal{P}$ be two process models with same activity set $N = \{a_1, a_2, \dots, a_n\}$. Let further $\mathcal{T}_S$, $\mathcal{T}_{S'}$ be the related trace sets and $A_{n\times n}$, $A'_{n\times n}$ be the order matrices of S and S'. Then $S \neq S' \Leftrightarrow A \neq A'$, if [$\neg\exists t_1, t'_1 \in \mathcal{T}_S$: t_1 is a substring of t'_1] and [$\neg\exists t_2, t'_2 \in \mathcal{T}_{S'}$: t_2 is a substring of t'_2].*

We give a proof of Theorem 5 in [7]. According to Theorem 5, there will be a one-to-one mapping between a process model S and its order matrix A, if the substring constraint is met. (Note that the substring constraint can be easily checked and handled [7]); i.e., if the conditions of Theorem 5 are met, the order matrix will uniquely represent the process model. Analyzing its order matrix (cf. Def. 3) will then be sufficient in order to analyze the process model.

Using the order matrix, 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. 2. 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. 2, 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; further these activities are contained in different branches of an XOR-block (as indicated by A_{AB} = '-'). Our algorithm for mining process variants (cf. Sec. 4) utilizes the sketched representation form and block concept.

4 Discovering Reference Process Models

We present a sophisticated algorithm for mining a collection of process variants. Our goal is to derive a new reference model 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. 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.

Our approach for mining process variants is as follows:

1. For all process variants calculate their order matrices and aggregate them to one high-dimensional matrix representing the whole variant collection (Sec. 4.1).
2. Based on this high-dimensional matrix, determine activities to be clustered in a block (Sec. 4.2).
3. Determine the order relation the clustered activities shall have within this block. (Sec. 4.3).
4. After building a new block in Steps 2 and 3, reflect the clustering of activities by adjusting the high-dimensional matrix accordingly (Sec. 4.4).
5. Repeat Steps 2, 3 and 4 until all activities are clustered together, i.e., until the new process model is constructed by enlargement of blocks.

These five steps are explained in the following. An illustrative example is given in Fig. 3. Reference model S has been configured into five process variants $S_i \in \mathcal{P}$ ($i = 1, 2, \ldots 5$), which are weighted based on the number of process instances created from them. In our example, 30% of all process instances were executed according to variant S_1, while 15% of the instances did run on S_2. 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 has weight $1/n$, where n corresponds to the number of variants in the system.

We can easily compute the distances (cf. Def. 2) between reference model and process variants. For example, when comparing S with S_1 we obtain distance *one* (cf. Fig. 3). Note that we only need to perform one change operation (i.e., $move(S, \texttt{E}, \texttt{A}, \texttt{D})$; cf. Def. 1) to transform S into S_1. Or when comparing S with S_2, needed change operations are $move(S, \texttt{D}, \texttt{B}, \texttt{C})$ and $move(S, \texttt{E}, \texttt{B}, \texttt{C})$, and distance between S and S_2 is *two*. Based on the weight of each variant, we can compute average weighted distance between reference model S and its variants; e.g., the distances between S and S_i are 1(i=1), 2(i=2), 2(i=3), 1(i=4), and 2(i=5); and the weights are 30%, 15%, 20%, 20%, and 15% (cf. Fig. 3). Thus average weighted distance equals $1\times0.3+2\times0.15+2\times0.2+1\times0.2+2\times0.15 = 1.5$. This means we need to perform on average 1.5 change operations to configure the reference model to a process variant or related instance respectively. Generally, the average weighted distance between a reference model and the process variants represents how "*close*" they are. The goal of our mining algorithm is to discover a reference model for a collection of (weighted) process variants with minimal average weighted distance to these process variants. In the following, we assume that each process variant has the same activity set (for a relaxation of this constraint see [6]).

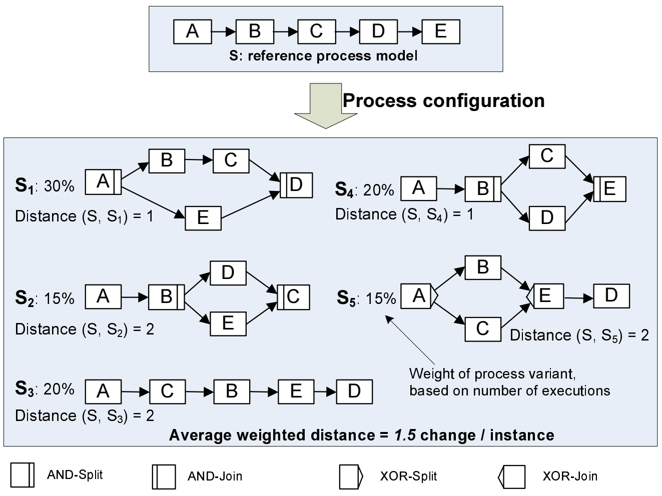


Figure 3. Illustrative example

4.1 Aggregated Order Matrix

For the given collection of process variants, we first compute the order matrix (cf. Def. 3) for each process variant. In our case, we need to determine five order matrices (cf. Fig. 4). 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 the same in all order matrices, this analysis does not result in a fixed relationship, but provides a distribution for the four types of order relations (cf. Def. 3). Regarding our example, activity C is in 65% of all cases a successor of activity B (as in S_1, S_2, S_4), in 20% of all cases a predecessor of B (as in S_3), and in 15% of the cases B and C are contained in different branches of an XOR block (as in S_5) (cf. Fig. 4). Therefore, we can define the order relation between two activities a and b as a 4-dimensional vector $V_{ab} = (v^0_{ab}, v^1_{ab}, v^*_{ab}, v^-_{ab})$: each field then corresponds to the frequency of the respective relation type ('0', '1', '*' or '-') as specified in Def. 3. Take our example from Fig. 4; here $v^1_{\mathtt{CB}}$ corresponds to the frequency of all cases with activities B and C having order relationship '1', i.e., where C is predecessor of B. In our example, we obtain $V_{CB} = (0.65, 0.2, 0, 0.15)$.

We define an *aggregated order matrix* as follows:

Definition 6 (Aggregated Order Matrix) *Let $S_i \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants with same activity set N. 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 $m = |N|$ and each matrix element $v_{jk} = (v^0_{jk}, v^1_{jk}, v^*_{jk}, v^-_{jk})$ being a 4-dimensional vector. For $\tau \in \{0, 1, *, -\}$, element v^τ_{jk} expresses to what percentage, activities a_j and a_k have order relation τ within the collection of process variants S_i. Formally: $\forall a_j, a_k \in N, a_j \neq a_k : v^\tau_{jk} = (\sum^n_{i=1, A_{i_{jk}}='\tau'} w_i)/(\sum^n_{i=1} w_i)$.*

The aggregated order matrix V for the process variants from Fig. 3 is shown in Fig. 4.

In an aggregated order matrix, main diagonal 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 3 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,

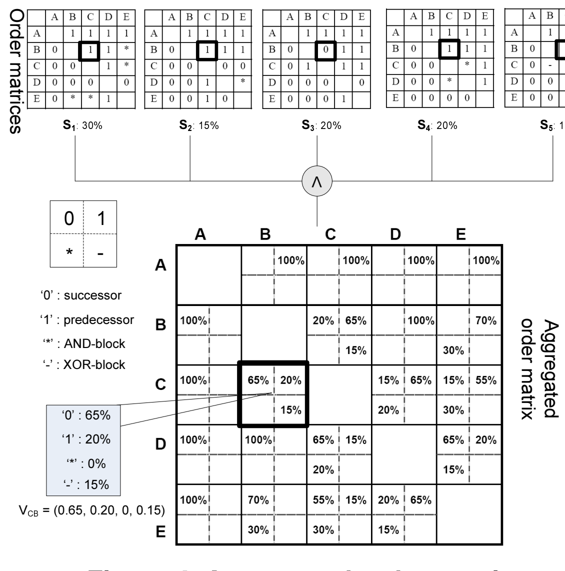


Figure 4. Aggregated order matrix V

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 from this representation form.

4.2 Determine Activities to Be Clustered

This subsection describes how to derive blocks of our reference model based on an aggregated order matrix. There are two issues we have to consider. First, we have to decide which activities 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 Sec. 4.3.

In an order matrix, two activities can be clustered in a block if they have same order relations with respect to the other activities (cf. Sec. 3). 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 4-dimensional vector showing the distributions of the order relations over all process variants. When determining the 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. For this reason, we first introduce function $f(\alpha, \beta)$ which expresses the closeness between two vectors $\alpha = (x_1, x_2, ..., x_n)$ and $\beta = (y_1, y_2, ..., y_n)$:

$$f(\alpha, \beta) = \frac{\alpha \cdot \beta}{|\alpha| \times |\beta|} = \frac{\sum^n_{i=1} x_i y_i}{\sqrt{\sum^n_{i=1} x_i^2} \times \sqrt{\sum^n_{i=1} y_i^2}}$$

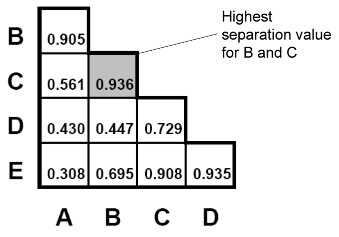
$f(\alpha, \beta) \in [0, 1]$ computes the cosine value of the angle θ between the two vectors α and β in Euclid space. If $f(\alpha, \beta) = 1$ holds, α and β will exactly match in their directions; $f(\alpha, \beta) = 0$ means, they do not match at all. When comparing closeness between $v_{\mathtt{BE}}$ and $v_{\mathtt{CE}}$, for instance, we obtain $f(v_{\mathtt{BE}}, v_{\mathtt{CE}}) = 0.968$. This high value implies that these two vectors are close to each other, though they are not the same.

Based on $f(\alpha, \beta)$ we introduce **Separation**. It indicates how well two activities of an aggregated order matrix are suited for being clustered in a block. More precisely, $Separation(\mathtt{A}, \mathtt{B})$ expresses how similar order relations of activities A and B are when compared to the rest of activities. In our example from Fig. 3, $Separation(\mathtt{A}, \mathtt{B})$ is determined by the closeness (measured by the cosine value) of $f(v_{\mathtt{AC}}, v_{\mathtt{BC}})$, $f(v_{\mathtt{AD}}, v_{\mathtt{BD}})$ and $f(v_{\mathtt{AE}}, v_{\mathtt{BE}})$. Generally, we define cluster separation as follows:

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

N corresponds to the set of activities. Like most clustering algorithms [13], we square the cosine value to emphasize the differences between the two compared vectors. Finally, by dividing this expression by $|N| - 2$, we normalize its value to a range between [0, 1]. Regarding our example from Fig. 3, we obtain $Separation(\texttt{A},\texttt{B}) = 0.905$.

We determine the pair of activities mostly suited to form a block by measuring how much each pair of activities is separated from the others. We accomplish this by computing the separation value for each activity pair. The higher this value is, the more suited the two activities are for being clustered. For our example from Fig. 3, the separation values are shown in Fig. 5. We denote this table as *separation table*. It becomes clear that activities `B` and `C` have the highest separation value 0.936 (marked up in grey). We therefore choose `B` and `C` for creating our first block.



	A	B	C	D
B	0.905			
C	0.561	0.936		
D	0.430	0.447	0.729	
E	0.308	0.695	0.908	0.935

Highest separation value for B and C

Figure 5. Separation table of the aggregated order matrix in Fig. 4

4.3 Determine Internal Order Relations

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

In our aggregated order matrix, the relationship between activities `B` and `C` is depicted as a 4-dimensional vector $v_{\texttt{BC}} = (0.2, 0.65, 0, 0.15)$. It shows the distribution values of the four types of order relations in a 4-dimensional space. Obviously, when building a reference process model, only one of the four order relations can be chosen. Therefore, we want to choose that type of order relation which is most significant when compared to the others. Regarding our example, the significance of each order relation can be evaluated by the closeness $v_{\texttt{BC}}$ and the four axes in the 4-dimensional space have. These four axes can be represented by four benchmarking vectors: $v^0 = (1, 0, 0, 0)$, $v^1 = (0, 1, 0, 0)$, $v^* = (0, 0, 1, 0)$, and $v^- = (0, 0, 0, 1)$. Therefore, we can compute the significance of each order relation using formula $f(\alpha, \beta)$ (cf. Sec. 4.2), where $\alpha = v_{\texttt{BC}}$ and β is one of the four benchmarking vectors. Regarding our example, the closest axis to $v_{\texttt{BC}}$ is v^1 ($f(v_{\texttt{BC}}, v^1) = 0.933$). Therefore, we decide that `B` shall precede `C` within the derived block (cf. Def. 2).

We can use cohesion to evaluate how good our choice is:

$$Cohesion(a, b) = 2\times max\{f(v_{ab}, v^0), f(v_{ab}, v^1), f(v_{ab}, v^*), f(v_{ab}, v^-)\}-1$$

The measure has value range [0,1]. $Cohesion(a, b)$ will equal *one* if there is a dominant order relation, i.e., v_{ab} is on one of the four axes. $Cohesion(a, b)$ will equal *zero* if v_{ab} is $(0.25, 0.25, 0.25, 0.25)$, i.e., no order relation is more significant than others. Regarding our example, $Cohesion(\texttt{B},\texttt{C})$ equals 0.867. This indicates high significance for setting `B` as predecessor of `C`.

4.4 Recompute the Aggregated Order Matrix

We have discovered the first block of our reference process model, which contains `B` and `C`. We have further decided that `B` shall precede `C`, and that the significance of this order relation is 0.867. We now have to decide on the relationship between this newly created block and the other activities.

Regarding the process variants from Fig. 3, `B` and `C` do not always constitute an elementary block (i.e., a block only containing `B` and `C`). To be more precise, `B` and `C` represent an elementary block in S_1, S_3 and S_5, but not in S_2 and S_4. Nevertheless, `B` and `C` are most suited to form a block based on our analysis of the aggregated order matrix. This requires adaptation of the original aggregated order matrix in order to represent the situation in which `B` and `C` are clustered in a block.[2] We accomplish this adaptation by computing the means of the order relations between $\{\texttt{B}, \texttt{C}\}$ and the remaining activities. For example, as $v_{\texttt{BD}} = (0, 1, 0, 0)$ and $v_{\texttt{CD}} = (0.15, 0.65, 0.2, 0)$ hold, the order relation between the newly created block $(\texttt{B},\texttt{C})$ and activity `D` will be $(v_{\texttt{BD}} + v_{\texttt{CD}})/2 = (0.075, 0.825, 0.1, 0)$. 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:

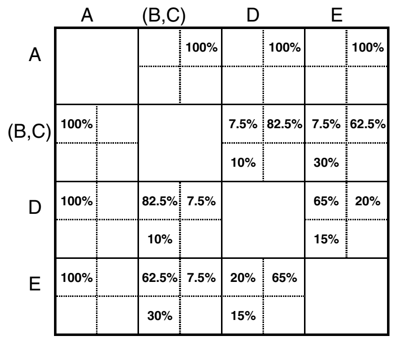
1. $\forall x \in N \setminus \{a, b\} : \begin{cases} v'_{(a,b)x} = (v_{ax} + v_{bx})/2 \\ v'_{x(a,b)} = (v_{xa} + v_{xb})/2 \end{cases}$
2. $\forall x, y \in N \setminus \{a, b\} : v'_{xy} = v_{xy}$

The aggregated order matrix V' we obtain after clustering `B` and `C` is shown in Fig. 6. Since `B` and `C` are replaced by a block containing them, the matrix resulting after the recomputation is one dimension smaller than V. Afterwards, we treat this block like a single activity.

4.5 Mining Result and Evaluation

After obtaining a newly aggregated order matrix, we repeat the three steps as described in Sec. 4.2, 4.3 and 4.4; i.e., first identify the activities (and blocks respectively) to be clustered, then determine their order relation within the block, and finally re-compute the aggregated order matrix considering the newly determined block. In every iteration,

[2]Our approach is different to traditional clustering algorithms [13], in which only distances are re-computed, but not the original dataset. We give a discussion why we need to change the original dataset in [6].



	A		(B,C)		D		E	
A				100%		100%		100%
(B,C)	100%				7.5%	82.5%	7.5%	62.5%
					10%		30%	
D	100%		82.5%	7.5%			65%	20%
			10%				15%	
E	100%		62.5%	7.5%	20%	65%		
			30%		15%			

Figure 6. New aggregated order matrix V' after clustering B and C

we merge two activities (and blocks respectively) into a bigger block. This iterative clustering continues until all activities (and blocks respectively) are clustered; then we have constructed the new reference model. Obviously, the number of required iterations equals the number of activities minus one. The final result after all iterations is shown in Fig. 7.

Fig. 7 shows the process model S' we have discovered through mining the variants from Fig. 3. It also shows how such result is constructed after every iteration (shown as a number at the right-bottom corner of each block). In iteration 1, B and C are clustered to form a block; in the next iteration, such block is enlarged by clustering activity E with it. Finally, after the fourth iteration, all activities have been clustered together into a single block, i.e., we have obtained our new reference model. Fig. 7 also shows cohesion values, which reflect the significance of the order relations we haven chosen in the different iteration.

5 Validation

The complexity of the mining algorithm described in Sec. 4 corresponds to $\mathcal{O}(n^3)$. Like most clustering algorithms in data mining [13], this algorithm is trying to solve a complex combinatorial optimization problem in polynomial time. This leads to the benefit that it can solve a problem with large scale, but has the disadvantage that it only searches for a local, but not global optimum. Therefore, it is not possible to prove that the algorithm really does what we want, i.e., to reduce biases in the system by improving the reference process model.

However, we can compare the process model discovered with our mining method with some other models. This comparison is based on how many high-level change operations are required to configure the respective process variants out of the reference model (cf. Def. 2). For this purpose, for each candidate model S_{can}, we assume that it is considered as the new reference model and then calculate the average weighted distance between S_{can} and the five variants $S_1 - S_5$; e.g., in Fig 3 the average weighted distance between S and the five variants is 1.5, which reflects how close the current reference model is to the process variants. Using same method, we can compute the average weighted distance between candidate reference model S_{can} and variants $S_1 - S_5$.

There are two groups of process models that are candidates for becoming the new reference process model. The first group contains all models we already know. Clearly, reference model S and the five variants S_1, S_2, S_3, S_4 and S_5 (cf. Fig. 3) belong to this group. Comparing these existing models with the one we obtained through process variant mining, for example, already 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, process model S' (cf. Fig. 7), as obtained with our algorithm for process variant mining, 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* [15], and compare them with our approach. The goal of process mining techniques is to discover process models from execution log. An execution log typically documents the start and/or end of each activity in a PAIS, and therefore reflects PAIS's behavior. 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 from Fig. 3 (cf. Fig. 8). We then use them as input for different process mining techniques. We consider Alpha algorithm [16], Alpha++ algorithm [19], Heuristics Mining [18], and Genetic Mining [2], which are some of the most well-known algorithms for discovering process models from execution logs. The discovered process models are shown in Fig. 8. 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 model with six silent activities (and the distances to each process variant is higher than *three*).

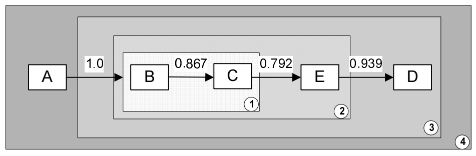


Figure 7. The resulting process model S'

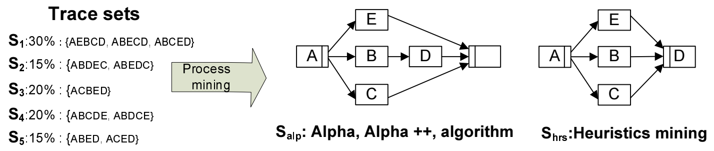


Figure 8. The candidate process models

We now compute the distances between each candidate model S_{can} from the two groups and the five process variants $S_1 - S_5$. Further, we compute the average weighted distance. Comparison results are depicted in Fig. 9. For example, if we consider process variant S_1 as reference model, the distance between this model and variants S_2, S_3, S_4, and S_5 will equal 2, and average weighted distance between S_1 and the five process variants will be 1.4 (cf. column S_1 in Fig. 9). This means that when choosing S_1 as new reference model we would need to perform on average 1.4 changes to configure the process variants out of S_1. Similarly, we can compute distances for the other candidate models. Altogether, results from Fig. 9 show that S' (cf. Fig. 7) –the process model resulting from the approach we suggested –has the shortest average weighted distance to the given process variants. This means, setting S' as new reference process model would require lowest efforts for configuring the variants. More precisely, we only need to perform on average 1.15 changes to configure a process variant out of S'. Note that process models S_{alp} and S_{hrs}, we discovered by applying conventional process mining algorithms (based on traces), show the largest average distance to the five variants. We obtained similar results when mining other collections of process variants.

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 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, the discovered process model S' (cf. Fig. 7) is also not good in terms of behavior, since the behavior of S', which can be expressed by the trace set producible by S', is limited when compared to the trace sets the variants can produce. We give a detailed comparison in [6].

Obviously, it is not possible to enumerate all process models, since the number of process models can be infinite. However, as depicted In Fig. 9, the discovered process

S_{hrs}	S_{alp}	S'	S	S_5	S_4	S_3	S_2	S_1	
1	2	1	1	2	2	2	2	0	**Distance to S_1 (30%)**
2	2	2	2	2	2	2	0	2	**Distance to S_2 (15%)**
2	2	1	2	1	2	0	2	2	**Distance to S_3 (20%)**
2	2	1	1	2	0	2	2	2	**Distance to S_4 (20%)**
2	2	1	2	0	2	1	2	2	**Distance to S_5 (15%)**
1.7	2	1.15	1.5	1.5	1.6	1.45	1.7	1.4	**Average distance**

Figure 9. The average weighted distance between candidate models and the variants

model is at least better than the process models currently known and the process models which can be produced by process mining algorithms based on traces. Keeping our search at local optimum will also make our approach applicable to real cases, since we can limit complexity at polynomial level.

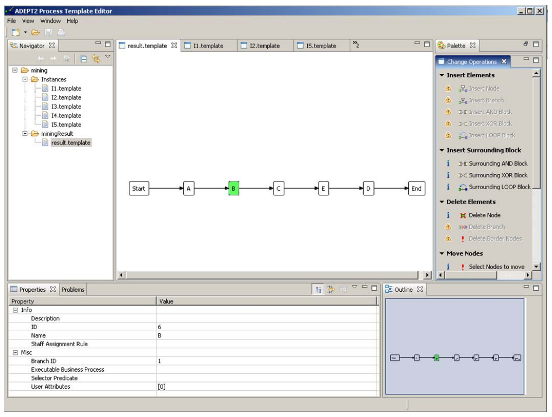
6 Proof-of-Concept Prototype

The described approach has been implemented and tested using Java. Figure 10 shows a screenshot of our prototype. We use ADEPT2 Process Template Editor [10] as the tool to create process variants. For each process model, the editor can generate an XML document with all information about the process model (like nodes, edges, blocks) being marked up. We store created variants in the variants repository folder *"instances"* (cf. Fig. 10) for mining.

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

7 Related Work

A variety of techniques for process mining has been suggested in literature [15, 18, 2, 16]. As illustrated in Sec. 5, traditional process mining is different from process variant mining due to its different goals and inputs. [5] 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. In addition, a few techniques have been proposed to learn from process variants by mining change primitives. [1] 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 and it does



Property	Value
Info	
Description	
ID	6
Name	B
Staff Assignment Rule	
Misc	
Branch ID	1
Executable Business Process	
Selector Predicate	
User Attributes	[0]

Figure 10. Screenshot of the prototype

also not differentiate AND- and XOR- branchings. Similar techniques for mining change primitives exist in the field of association rule mining [13] and maximal sub-graph mining [13] as known from graph theory [12]; 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, [3] 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, this approach simply considers each change as individual operation so that the result is more like a visualization of changes rather than mining them. [8] presents a method for retrieving process variants using a query model, but does not provide any mining support. None of the discussed approaches aims at creating a reference process model, which allows for easy and optimized configuration for process variants in future.

8 Summary and Outlook

In this paper, we provide a sophisticated approach to mine block-structured process variants in order to discover a reference process model 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 low 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 based on conventional process mining algorithms [15]. The validation result also implies 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 algorithm to support the mining of process variants with the goal of obtaining a reference model which is easily configurable.

Our approach looks promising, but there are still several questions left open. First we have to include more control structures (like loops or synchronization constraints for parallel activities) as proposed in ADEPT [9] or BPEL. In addition, we want to design search or heuristic algorithms to limit differences between original reference process model and the one we discover through variant mining. In this way, we can also limit the costs to update the old reference model to the new one.

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] A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, NL, 2006.

[3] C.W. Günther, S. Rinderle, M. Reichert, and W.M.P. van der Aalst. Change mining in adaptive process management systems. In *CoopIS'06*, pages 309–326, 2006.

[4] A. Hallerbach, T. Bauer, and M. Reichert. Managing process variants in the process lifecycle. In *ICEIS '08*, to appear, 2008.

[5] 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.

[6] C. Li, M. Reichert, and A. Wombacher. Discovering process reference models from process variants using clustering techniques. Technical Report TR-CTIT-08-30, University of Twente, Enschede, March 2008.

[7] C. Li, M. Reichert, and A. Wombacher. On measuring process model similarity based on high-level change operations. In *ER '08*, 2008.

[8] R. Lu and S.W. Sadiq. Managing process variants as an information resource. In *BPM'06*, pages 426–431, 2006.

[9] M. Reichert and P. Dadam. ADEPT flex -supporting dynamic changes of workflows without losing control. *Journal of Intelligent Info. Sys.*, 10(2):93–129, 1998.

[10] M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with ADEPT2. In *ICDE '05*, pages 1113–1114. IEEE Computer Society, 2005.

[11] M. Rosemann and W.M.P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, 2007.

[12] K.H. Rosen. *Discrete Mathematics and Its Application*. McGraw-Hill, 2003.

[13] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[14] 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, uary.

[15] 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 Knowl. Eng.*, 47(2):237–267, 2003.

[16] 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.

[17] B. Weber, S. Rinderle, and M. Reichert. Change patterns and change support features in process-aware information systems. In *CAiSE'07*, pages 574–588, 2007.

[18] 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.

[19] L. Wen, J. Wang, and J. Sun. Detecting implicit dependencies between tasks from event logs. In *APWeb'06*, pages 591–603, 2006.