Discovering Process Reference Models from Process Variants Using Clustering Techniques

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, success of an enterprise increasingly depends on its ability to react to 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, has led to large number of process and service variants derived from the same model, but differs in structures due to the applied changes. This paper provides a sophisticated approach which fosters learning from past process changes and allows for determining such process variants. As a result we obtain a generic process model for which the average distances between this model and the process variants becomes minimal. By adopting this generic process model in the PAIS, need for future process configuration and adaptation will decrease. The mining method proposed has been implemented in a powerful proof-of-concept prototype and further validated by a comparison between other process mining algorithms.

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 (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 [9, 28, 4]. 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 [5].

The ability to adapt and configure process models at the different levels will result in a collection of model variants (i.e., configurations [1]). Such variants are 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 [24].

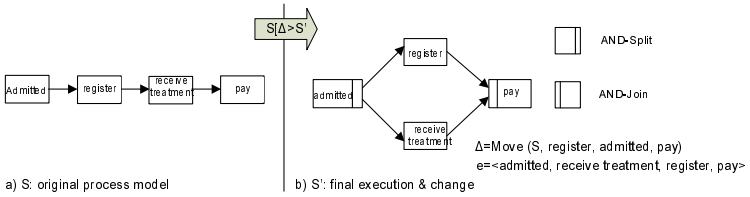


Fig. 1. Original Process Model S and Process Variant S'

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 [3], which aggravates maintenance significantly.

Although some efforts on process flexibility have been made to make the process configuration and customization easier [9, 13, 4, 2].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.

Based on the two assumptions that: (1) process models are block-structured [9] 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 generic reference model out of them, such that the average distance between the reference model and the process variants becomes minimal?

The distance between two process models is measured by the number of change operations needed to transform one process model into another one. Therefore, this figure can directly represent the effort for process adaptation and customization. (further explanation of distance is available in Section 2). Clearly, when the process variants are given, setting different model as the reference model would result in different average distance. The challenge is to find the best reference model, i.e. the one with minimal average distance to the process variants.

The reminder of this paper is organized as follows: Section 2 gives background information needed for understanding this paper. Section 3discusses major goals of process variants mining and shows why it is different from traditional process mining. In Section 4 we shows a method to represent a process model using a matrix called order matrix. After that, shows the algorithm to perform process varints mining at Section 5 and further validate the algorithm at Section 6. We further extend it to handle mining with different activity set at Section 7. The algorithm and prototype are afterwards given in Section 8.1. This paper concludes with a summary ad an outlook in Section 10.

2 Backgrounds

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

Process Model Let $\mathcal{P}$ denote the set of all correct process models. A particular *process model* $S = (N, E, \ldots) \in \mathcal{P}$ is defined as a well-structured Activity Net [9]. N constitutes a set of activities a_i and E is a set of control edges linking them. To limit the scope, we assume Activity Net to be block structured. Examples are provided by Fig 1.

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/or their order relations. Thus, each application of a change operation results in a new process model. We define *high-level change operations* on a process model as follows:

Definition 1 (Process Change and Process Variant). *Let $\mathcal{P}$ denote the set of possible process models and $\mathcal{C}$ 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 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* $S_i[\Delta\rangle S_{i+1}$ *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 the position of an activity and thus the structure of the process model. For example, operation $move(S, \texttt{A, B, C})$ 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 [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 [5]. 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 [10]. 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 [5].

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 [12]. We omit further details and refer to [12].

Here we use high-level change operations rather than change primitives (basic changes like adding/removing nodes and edges) to measure the distance between

[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 [10, 12] for details.

process models. The reason is that using high-level change operations can guarantee soundness and provide more meaningful measure of the distance. Due to the limited space, we omit the details are refer readers to [27].

Trace

Definition 3 (Trace). *Let $S = (N, E, \ldots) \in \mathcal{P}$ be a process model. We can define t as a trace of S iff:*

- *$t \equiv < a_1, a_2, \ldots, a_k >$, $a_i \in N$, be a sequence of activities, which is a valid and complete execution sequence based on the control flow in S. We define $\mathcal{T}_S$ be a set which contains all the traces process model S can produce.*
- *$t(a \prec b)$ is denoted as precedence relationship between activities a and b in trace $t \equiv < a_1, a_2, \ldots, a_k >$ iff $\exists i < j : a_i = a \wedge a_j = b$.*

Here, we only consider traces composing 'real' activities, but no events related to silent activities (i.e. activity nodes which contain no operation and exist only for control flow purpose). A detailed discussion about silent activity can be found in [12]. At this stage, we consider two process models as being the same if they are *trace equivalent*, i.e. $S \equiv S'$ iff $\mathcal{T}_S \equiv \mathcal{T}_{S'}$. The stronger notion of bi-similarity [11] is not required in our context.

3 Mining Process Variants: Goals and Comparison with Process Mining

In this section, we first motivate the major goal behind mining process variants, namely to derive a reference 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 reference model. The letter is of particular importance if we want to learn from process instance deviations, derive an optimized process model from them, and migrate the already running instances to the newly derived model afterwards [28]. We measure the efforts for respective process configurations by the number of high-level change operations needed to transform the reference model into the respective model variant. The challenge is to find a reference 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. Process mining based on execution log has been researched intensively during the past few years [6]. An overview on existing techniques is given in [16]. In a nutshell, these approaches are trying to discover the underlining process model from execution logs which typically record the starting/completion events of each process activities. 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 [15] 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 (c.f. Def. 3). Therefore, two process models can be compared based on the difference between their trace sets [6, 15, 16]. 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 [12]. 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. 2 shows two examples.

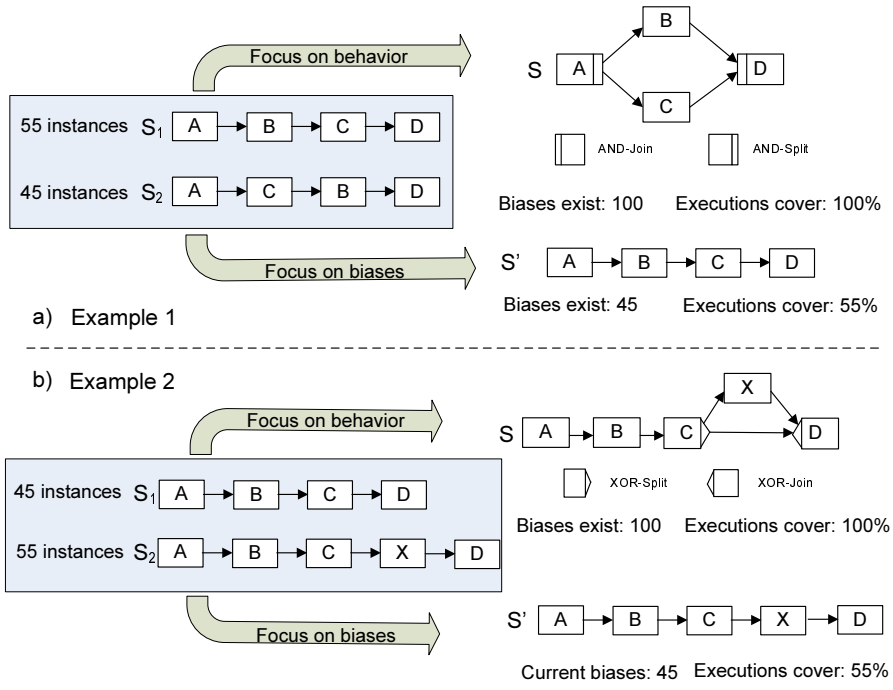


Fig. 2. Mining focusing either on Behavior or on Minimization of Biases

Consider Example 1 (c.f. Fig. 2a)) 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 reference process model such that the efforts for configuring the 100 process instances out of the reference model become minimal. If we focus on behavior, like existing process mining algorithms do, 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 $\sigma_1 = move(S, \texttt{B}, \texttt{A}, \texttt{C})$ (c.f. Def. 1 for detail parameterization) 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 [12].

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 reference 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' cannot cover all traces process variants S_1 and S_2 can produce, adapting S' rather than S as the new reference 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 (c.f. Fig. 2), activity `X` is only present in S_2, but not in S_1. When applying traditional process mining, 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. 2 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, see Sec. 5.

Our discussions on the difference between behavioral and structural similarity also demonstrate that current process mining algorithms do not consider structural similarity based on bias and change distance (we will quantitatively 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 to use these traces as input source for process mining algorithms based on process logs. 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 $(5 \times 5)!/(5!)^5 = 623360743125120$.

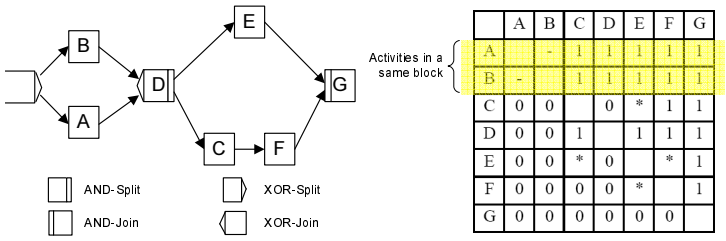
4 Represent Process Model Logic as an Order Matrix

After showing the goal for process variant mining, we start describing the approach to perform the mining. The theoretical backgrounds of high-level change operations have been discussed in ADEPT technique which is based on WSM net [9], and life-cycle inheritance which is based on Petri net [10]. 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 the predecessors of this activity, and also not their control relationships. To incorporate this feature in our approach, rather than only looking at direct predecessor-successor relationships between two activities (i.e. control flow edges), we consider the transitive control dependencies between all pairs of activities; i.e. for every pair of activities $a_i, a_j \in N \bigcap N'$, $a_i \neq a_j$, their execution order compared to each other is examined. Logically, we check the execution orders by considering all traces a process model can produce (c.f. Sec. 2). Results can be formally described in a matrix $A_{n \times n}$ with $n = |N \bigcap N'|$. Four types of control relations can be identified (cf. Def. 4):

Definition 4 (Order matrix). *Let $S = (N, E, \ldots) \in \mathcal{P}$ be a process model with $N = \{a_1, a_2, \ldots, a_n\}$. 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 relation between different activities a_i,$a_j \in N$ 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 is predecessor 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 activity a_i and a_j, a_i always appears AFTER *a_j, then we denote A_{ij} as a '**0**', i.e. a_i is successor 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 at least one other 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.*

Figure 3 shows an example. Besides control flow edges which shows direct predecessor-successor relationship, process model S also contains four types of control flows: AND-Split, AND-Join, XOR-Split and XOR-join. The order matrix can represent all these relationship in the matrix. For example activity `A` and activity `B` would never appear in one trace since they are in two different



	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	

Fig. 3. Order matrix for process model S

branches of an XOR block. It is denoted as '-' in the matrix at A_{ab} and A_{ba}. Similarly, we can denote all the execution orders between every pair of activities. The main diagonal in the matrix is empty since we do not compare an activity with itself. As one can see, elements A_{ij} and A_{ji} can be derived from each other. If activity a_i is a predecessor of activity a_j, (i.e. $A_{ij} = 1$), we can always conclude that $A_{ji} = 0$ holds. Similarly, if $A_{ij} \in \{'*','-'\}$, then we will obtain $A_{ji} = A_{ij}$. Please note that order matrix is different from the adjacency matrix as used in graph theory [18]. The reasons include that it differentiates AND and XOR block, does not contain unnecessary silent activities, and handle loop differently [12].

Under certain constraints, an order matrix A can uniquely represent the process model, based on which it was built on. This is stated by Theorem 1. Before giving this theorem, we need to define the notion of *substring of trace*:

Definition 5 (Substring of trace). *Let t and t' be two traces. We define t is a 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 1. *Let $S, S' \in \mathcal{P}$ be two process models, with same set of activities $N = \{a_1, a_2, \ldots, a_n\}$. 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 $t'_2)$.*

According to Theorem 1, there is a one-to-one mapping between a process model S and its order matrix A, if the substring constraint is met. A proof of Theorem 1 can be found in [12]. The substring constraint is also not very strong since we can easily detect and handle it [12]. Thus, analyzing order matrix (cf. Def. 4) would be sufficient for analyzing a process model, since an order matrix can uniquely represent the process model.

The order matrix can also help on determining blocks: if the internal execution orders of the activities in the same block are ignored, these activities should have the same execution orders to the rest of activities. Take the order matrix in Fig. 3 for example. If we ignore the internal relationship between activities `A` and `B`, the execution orders between `A` and the rest of activities are the same

as the execution orders between B and the rest of activities (as marked up in yellow in Fig. 3, the first two rows are the same if we ignore the execution order between A and B). Therefore, without knowing the process model, we can already determine a block with activities A and B, which are in different branches of an XOR block.

We therefore can apply the similar idea on designing the mining algorithm. When two activities have the same or similar execution orders to the rest of activities, we therefore can cluster them together as a block. And if we repeat it again and again, then activities will form a small blocks, and small blocks will form bigger ones, and we can therefore mine a process model out by clustering activities or blocks. The following sections will give a detail explanation of our method.

5 Discovering Generic Process Reference Model

The goal of this paper is to mine the process variants in order to derive a new reference model which is easier configurable. Since we restrict ourselves on handling block-structured process models, we can build a process model by enlarging blocks, i.e. some activities can form blocks, and the block is enlarged by merging other blocks. If all the activities and blocks are linked together, we therefore can get a process model which represent the new reference process models.

Therefore, our general approach for mining process variants is as follows:

1. Represent process variants by a Type-level Order Matrix (Section 5.1).
2. Determine which activities should be clustered together to form a block, based on the type-level order matrix. (Section 5.3).
3. Make respective change of the type-level order matrix after building a block in step 2 (Section 5.4).
4. Repeat 2,3 until all activities (blocks) are clustered together.
5. Evaluation the fitness of the resulting process model (Section 5.5).

An illustrative example is given in Fig. 4. It contains five process variants $S_i \in \mathcal{P}, \rangle = \infty, \in, \ldots \bigtriangledown$ as well as the weight of each variance based on number of execution. In our example, 30% of instances were executed according to process variant S_1, while 15% instance according to variant S_2. In another scenario, for example, when we only have the process variants derived from the same reference model, but no information about the instance executions, we can assume the variants are un-weighted. i.e. every process variant will have the same weight $1/n$, where n is the number of process variants in the system. So far, our example is given based on the assumption that each process variant has the same activity set. See Sec. 7 for a relaxation of this constraint.

5.1 Type-level Order Matrix

When given a set of process variants, we first need to compute the order matrices for these process variants. To our case, we need to draw five order matrices

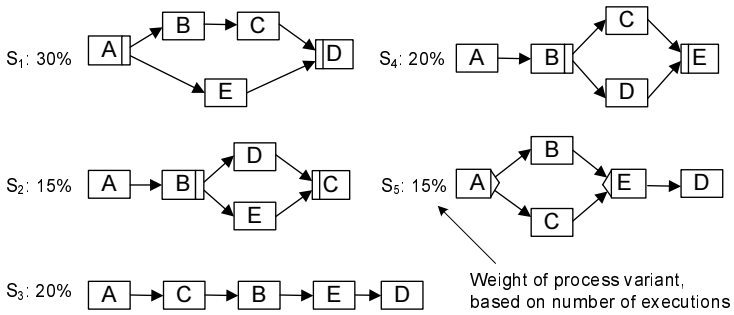


Fig. 4. Illustrative example for our mining approach

according the five process variants in the system (c.f. Fig. 5). Afterwards, we can analyze the execution orders between every pair of activities shown in different order matrices. As the execution order between two activities may not be unique in all order matrices, this is not a fixed relationship but a distribution value of the four types of execution orders. For example, activity `C` is 65% successor of activity `B` (as in S_1, S_2, S_4), 20% predecessor of `B` (as in S_3) and 15% in different XOR-split branching with `B` (as in S_5). We therefore can define the execution order between two activities `a` and `b`, as a four dimensional vector, $V_{ab} = (v_{ab0}, v_{ab1}, v_{ab*}, v_{ab-})$, which corresponds to the frequency of the four types of execution orders ('0', '1', '*', '-') as specified in Def. 4. For example, $v_{\texttt{CB}1}$ shows the percentage where activities `C, B` have relationship '1', i.e. activity `C` is a predecessor of `B`. Clearly, $v_{ab0} + v_{ab1} + v_{ab*} + v_{ab-} = 1$. To our case, $V_{CB} = (0.65, 0.2, 0, 0.15)$.

We can formally define a type-level order matrix as follows:

Definition 6 (Type-level Order Matrix).
Let $S_i = (N_i, \ldots) \in \mathcal{P}$, $i = (1, 2, \ldots, n)$ be a set of process variants. A_i is the order matrix for S_i and w_i represents the number of instances executed according to S_i. The Type-level Order Matrix for the process models is defined as a 3-dimensional matrix $V_{m \times m}$ with $m = |\bigcup N_i|$ and $V_{jk} = (v_{jk0}, v_{jk1}, v_{jk*}, v_{jk-})$ being a four dimensional vector. $v_{jk0} = \sum_{i=(1,\ldots,n), A_{i_{jk}}='0'} w_i / \sum_{i=(1,\ldots,n)} w_i$ for $\forall S_i \in \mathcal{P} : a_j, a_k \in N_i \wedge j \neq k$. Similarly we can compute v_{jk1}, v_{jk*} and v_{jk-}.

In a type-level order matrix, V_{jk} shows the percentage value of the four types of execution orders in all the process models containing activity a_j and a_k ($j \neq k$). Please note that we do not necessarily constrain all process models to have the same activity set. We will further describe how we handle such situation in Section 7. To our example, the type-level order matrix V for the process variants shown in Fig. 4 is depicted in Fig. 5.

In the type level order matrix, the main diagonal is empty since we do not specify the execution order between an activity and itself. For the rest of elements, when the value in a certain dimension is empty, it means it is 0.

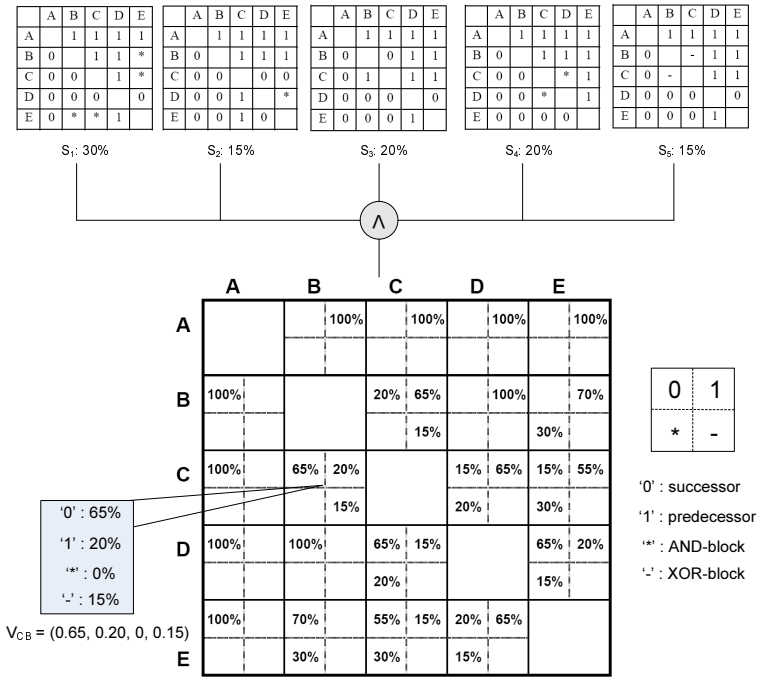


Fig. 5. Type-level order matrix

In Sec. 4, we have shown that we can use order matrix to determine blocks in a process model. i.e. two activities can be clustered together to form a block if the execution order between them and the rest of activities are same (c.f. Sec. 4). Similar idea can be applied, when analyzing type-level order matrix. The following sections will describe the methods in details.

5.2 Two Measures for Block Detection: Cohesion and Separation

A naive approach would be to make an order matrix based on the type-level order matrix by setting the execution orders to the one with the highest frequency in the vector. For example, because $V_{CB} = (0.65, 0.2, 0, 0.15)$ the execution order of `C, B` should be successor, since V_{AB0} is 0.65 which is the highest among the four. If we apply it to every pair of execution order, we can then generate an 'order matrix', based on which we can build one process model. Unfortunately it will not work, since an 'order matrix' determined in this way does not necessarily represent a valid process model. Take the order matrix in Fig. 3 for example, if we change A_{AE} from 1 to 0, there would be no process model representing such order matrix. This has triggered us to think of other methods.

As described in the general process at the beginning of this section, whenever we need to determine a block, we need to know which activities (blocks) should be in such block, and what execution order should the activities in the blocks

be? In this section, we will introduce two measures: *cohesion* and *separation* as the answer of these two questions.

Before we define cohesion and separation, we first need to define a function $f(\alpha, \beta)$ which shows the closeness between two vectors $\alpha = (x_1, x_2, ..., x_n)$ and $\beta = (y_1, y_2, ..., y_n)$:

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

$f(\alpha, \beta)$ computes the cosine value of the angle θ between the two vectors α, β in an Euclid space. The value range of if this function is $[0, 1]$ where 1 means two vector α, β precisely match in their directions and 0 when they are precisely vertical.

Cohesion *Cohesion* measures how closely related the objects in a cluster are. In our context: it indicates how strong the relationship between two activities is.

In the type level order matrix, if, for instance, we decide to cluster activity `B` and `E` together to form a block, the relationship between the two activities `B, E` in the block, is determined by the vector V_{BE}. This vector shows the execution order between `B` and `E`. We then can compare V_{BE} with the four benchmarking vectors (which corresponding to the four axes in the four-dimensional space): $V_0 = (1, 0, 0, 0)$, $V_1 = (0, 1, 0, 0)$,$V_* = (0, 0, 1, 0)$,$V_0 = (0, 0, 0, 1)$. To which of the four axes V_{BE} is closest, we can define it as the execution order between `B` and `E`. And the credibility is therefore determined by the cosine value between these two vectors. For example, $V_{BE} = (0, 0.7, 0.3, 0)$, so the closest vector is V_1, and the closeness which is evaluated by the cosine value equals 0.919.

Since cohesion is determined by angle between a vector and its closest axis, the value range of the cohesion value is not [0,1] but to some degree smaller. We can also normalize cohesion to a value between 0 and 1. It is not difficult to find that when a vector equals to $(0.25, 0.25, 0.25, 0.25)$, the cohesion equals to the minimal one which is 0.5. Therefore the cluster cohesion, which shows the credibility of the relationship between the two activities (blocks) in a block, equals:

$$Cohesion(A, B) = 2 \times max(f(V_{AB}, V_0), f(V_{AB}, V_1), f(V_{AB}, V_*), f(V_{AB}, V_-)) - 1$$

In this way, we can determine and measure the internal relationship between two activities (blocks) by a value between 0 to 1. Regarding to the example given above, $Cohesion$(`B,E`) therefore equals 0.838, and the execution order between activities `B,E` is *predecessor*.

Separation *Separation* measures how distinct or well-separated a cluster is from other clusters. In our context, we indicate how well two activities (blocks) are suited to be clustered together, i.e. to form a block.

This factor $Seperation(A, B)$ is determined by how similar the execution orders of activity `A` and `B` when compared with the rest of activities. As to our case shown in Fig. 4, $Seperation(A, B)$ is determined by the closeness (measured by the cosine value) of $f(V_{AC}, V_{BC})$, $f(V_{AD}, V_{BD})$ and $f(V_{AE}, V_{BE})$. Therefore, we can define cluster separation equals:

$$Separation(A, B) = \frac{\sum_{x \in N \setminus \{A,B\}} f^2(V_{Ax}, V_{Bx})}{|N| - 2}$$

In the formula of separation, N is the set of activities. The reason to square the cosine value is to emphasize the differences between the two compared vectors, like most clustering algorithms do [14]. And dividing the formula by $|N| - 2$ can normalize the value to a range between [0, 1]. As to our example in Fig. 4, $Seperation(A, B) = 0.905$.

5.3 Detecting Activities to Form a Block

After introducing Cohesion and Separation in Section 5.2, we can apply these two measures to determine: which two activities should be clustered together, and what the relationship between them should be.

The idea is similar to Agglomerative Hierarchical Clustering as introduced in the field of data ming [14]. Similar to clustering algorithm in data ming which measures the distance between each pair of nodes in a dataset, we measure the distance between each activity pair by computing the separation value between them. The higher the separation value is, the more likely two activities tend to be clustered together. Regarding to our case in Fig. 4, the separation values between every pair of activities are shown in Fig. 6. We denote this table as *separation table*.

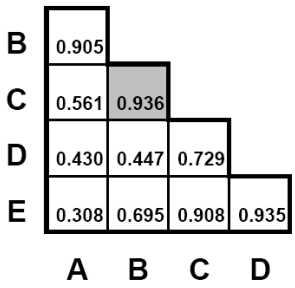


Fig. 6. The separation table of the type-level order matrix in Fig. 5

Regarding the separation table, it becomes clear that activity `B` and `C` have the highest separation value (marked up in grey). To be more precise, the separation between `B` and `C` is 0.936. Since it is the highest one, we can already

determine the activities in our first block: activity B and C. However, when dealing with complex examples, there can several maximal separation values. For this case, we also need to compute the cohesion between the pairs with the highest separation values. The pair with highest cohesion shall be selected, since the relationship of the two activities (measured by the cohesion) is most significant. If separation and cohesion are both the same for two blocks, we do not differentiate which one should be first.

After we determined the activities to form a block, we need to determine which execution order should activity B and C have, as well as the cohesion value which indicates how strong the relationship is (c.f. Cohesion). $Cohesion(B, C) = 0.867$, and the closest axis is V_1 which identifies that activity B is a predecessor of activity C.

5.4 Re-compute the Type-level Order Matrix

After we cluster activities B and C, we need to decide the relationship between this new block and the result of activities. In the traditional clustering techniques, the distance (separation, as to our case) between this block and the node out side this block is either the maximal, or the minimal or the average of the distances between that node and the nodes in the cluster [14]. For example, according to Agglomerative hierarchical clustering [14], after activities B and C are clustered together, the distance (separation) between this block and another activity (e.g. A) is either the minimal (0.561) or the maximal (0.905) or the average (0.733) of $separation$(A,B) = 0.905 and $separation$(A,C) = 0.561. Unfortunately, such technique will not work in our context because we do not simply consider each activity as a high dimensional vector. When two activities are clustered together, we actually moved the positions of the activities in the process variants so that they could be linked together to form a block. Therefore, rather than simply modify the separation values, we need to re-compute the execution orders between the activities in the block and the activities outside the block. We do it by compute the means. For example, activity B is 100% predecessor of D, while activity C is 15% successor, 65% predecessor and 20% AND-Split with C (c.f. Fig. 5). After we cluster activity B and C, the execution order of block (B,C) to activity D, turn to be 7.5% successor, 82.5% predecessor and 10% AND-split. We can formally describe it as follows: after activity a, b are clustered together, the new type-level order matrix $V'_{(n-1)\times(n-1)}$ equals:

1. $V'_{(a,b)x} = 1/2(V_{ax} + V_{bx})$ and $V'_{x(a,b)} = 1/2(V_{xa} + V_{xb})$ for all $x \in N \setminus \{a, b\}$
2. $V'_{xy} = V_{xy}$ for all $x, y \in N \setminus \{a, b\}$

The new type-level order matrix V' after clustering activity B and C is shown in Fig. 7.

5.5 Mining Result and Evaluation

After we get a new type-level order matrix, we can repeat the following two process as described in Section 5.3 and Section 5.4, i.e. first determine the activities

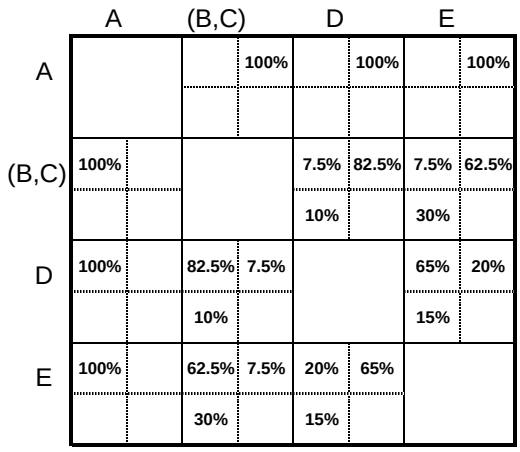


Fig. 7. The new type-level order matrix after clustering `B` and `C`

(blocks) to be cluster together, and then re-compute the type-level order matrix. The iteration will continue until all activities and blocks are clustered together (the number of iteration equals the number of activities minus two). The final result after all iteration is shown in Fig. 8.

Figure 8 shows the process model S' we mined out. The result does not only show a process model, but also shows how the process model been built up in each iteration (shown as a number at the right-bottom corner of each block) and what the credibility of the control flow is. For example, the credibility of the successor relationship between activity `E` and block with activity `C` and `D` is 0.792, which is relatively low when compared to the other control flows. It reflects that: based on the instances described in Fig. 4, the relationship between activity `C` and `E` has been configured more than others. And the credibility of activity `C` is a predecessor of activity `E` is not very strong.

Since we follow a strict block structure so far, the result can also be visualized in an expression tree as proposed in [25]. Similar tree structure can also be found in hierarchical clustering algorithm [14]. The tree structure of the process model in Fig. 8 is depicted in Fig. 9.

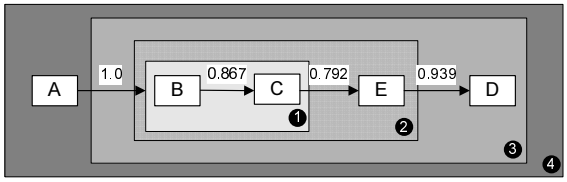


Fig. 8. The resulting process model S'

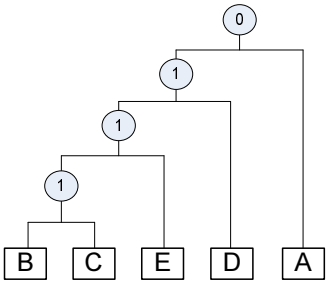


Fig. 9. The binary tree representation of process model S'

The rectangle in Fig. 9 represents the activities and the circle in the root represents the execution order. The execution order shows the relationship between the left subtree and the right sub tree. For example, the relationship between activity `B` and `C` is 1, which means `B` is a predecessor of `C`. The tree can grow as the blocks grow in our algorithm. Whenever a block is created, the new activities being included in the block will become the new right subtree of old one. For example, after activity `B` and `C` are clustered with the relationship of predecessor, the existing tree has two subtrees `B` and `C` and a root with execution order "1". After the first iteration, the algorithm decided to cluster the block with `B` and `C` with the activity `E`. The new activity will then be listed as the right subtree of the new tree with the execution order of "1" shown on the root.

If we perform an inorder traversal (details is available in [25]) of the tree, we can also represent the process model in a linear structure like an arithmetic expression. For this case the expression is: `(((B 1 C) 1 E) 1 D) 0 A)`. In this expression, 0,1 represent the execution order just like "+" "-" in arithmetics. Therefore `B 1 C` means activity `B` is a predecessor of `C`.

5.6 Two Evaluation Measures

Besides the cohesion which evaluate the local fitness of a certain block, we can also define global evaluation measures which show the "goodness" of the process model we mined out. The measures include: *accuracy* and *precision*.

We first can build an order matrix A' based on the process model S' which we mined out (c.f. Fig. 8). After that, we can also build a type-level order matrix V' only based on A'. For instance, if $A'_{ij} = 0$, then the corresponding element in the type-level order matrix is a vector $V'_{ij} = (1, 0, 0, 0)$. Or, if $A'_{ij} = 1$, we can determine the vector to be $V'_{ij} = (0, 1, 0, 0)$. Based on the comparison between the type-level order matrix V (c.f. Section 5.1) and V', the *accuracy* and *precision* can be defined as follow:

$$Accuracy(S') = \frac{\sum_{a_i,a_j \in N, i \neq j, V_{ij} = V'ij} 1}{|N| \times (|N| - 1)}$$

Accuracy(S') measures how accurate the mined out process model S' is when compared to the process variants. It is determined by how many execution or-

ders have been perfectly satisfied compared with all the activities pairs. We define two vectors V_{ij} and V'_{ij} perfectly match if $V_{ij} = V'_{ij}$. To our example, 10 execution orders satisfy this relationship out of 20 activity pairs. Therefore, $Accuracy(S') = 0.5$.

$$Precision(S') = \frac{\sum_{i,j \in \mathcal{N}, i \neq j, f(V_{ij} \neq V'ij)} f^2(V_{ij}, V'_{ij})}{\sum_{i,j \in \mathcal{N}, i \neq j, V_{ij} \neq V'ij} 1}$$

Precision(S') shows how precise the execution orders in S' are when compared to the process variants in the system. When computing the precision, only the none-perfect match cases are counted. Precision(S') equals the average square of the similarities (measured by the f function in Section 5.2) between the none-perfect match element in V and V'. We define two vectors V_{ij} and V'_{ij} not perfectly match if $V_{ij} \neq V'_{ij}$. The reason to square the cosine measure is to emphasize the different, as most data mining measures do [14]. In our case, $precision(S') = 0.837$.

Clearly, a more precise way to evaluate the resulting model is to really compute the distances between the mined model and the process variants. However, as computing the distance between two given process models is at $\mathcal{NP}$ level [12], it is not very sufficient to do so. Therefore, we introduce the two measures *accuracy* and *precision* to give an approximate result. In the following section, we will also compare these two different methods for measuring the goodness of the resulting model.

6 Validation

The complexity of the process variants mining algorithm described above is $\mathcal{O}(n^3)$. Like other clustering algorithms in data mining [14], this algorithm is trying to solve a complex combinatory optimization problem in polynomial time. This leads to the benefit that it can solve a problem with large scale, as well as the disadvantage that it only searches for local optimal but not global optimal. Therefore, it is not possible to systematically prove that the algorithm really does what we want, i.e. reduce the number of biases in the system by improving the reference process model.

However, we can compare the process model we mined out with some other models. The comparison is based on: how many biases would exist in the system if we set different models as the new reference process model. To emphasize again, the biases are evaluated by the number change operations need to transform the reference model to the variants. It is also same to the largest common divider as defined in [13] i.e., how many activities we need to remove from the process variants to make it as a life-cycle inheritance of the reference process model [10]. For example, if we compare the e S_1 in Fig. 4 with the process model S' as we mined out in Fig. 8. We only need to move *one* activity `E` to the position after `C` and before `D` (i.e. $move(S_1, \texttt{E, C, D})$ in ADEPT [9]) in order to transform S_1 to S'. If we apply the life-cycle inheritance as Aalst defined in Petri-net world

[13, 10] we only need to remove activity E in S_1 so that S_1 could be an life-cycle inheritance process model of S'. In this way, we can define that if we set the process model S' as the new process template, there is only one bias existing in each instance running according to process model S_1. By applying the similar technique, we can compute the biases in other variants if we set S' as the new type level order matrix.

There are two groups of process models that can be the candidates for the new reference process model. The first group contains all the process models we already know. Clearly, the five process variants S_1, S_2, S_3, S_4, S_5 shown in Fig. 4 belong to this group. In addition, we can also assume that there exists an original reference process model. Lets assume it is S in Fig. 10. Comparing these existing models with the one we obtained through our approach, 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).

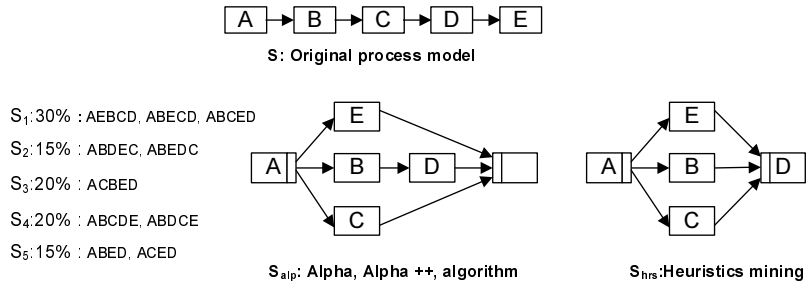


Fig. 10. The candidate process models

The second group of process models includes the process models that we discovered using different algorithms. Clearly, the process model S' from Fig. 8, as obtained using our process variants algorithm is in this group. In addition, we can compare the process models discovered using traditional process mining algorithms based on traces [16]. 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, and use these traces as the input of the traditional mining algorithms (c.f. Fig. 10 for all the traces). The mining algorithms we applied here includes Alpha algorithm [20], Alpha++ algorithm [21], Heuristics mining [22] and Genetic mining [23]. (These are some of the most well-known algorithms for discovering process models from execution logs). 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: genetic mining resulted in a complex model with six silent activities (and the distances to each process variant is higher than *three*).

Given the two groups of process models, the comparison is also made based on two methods. Clearly, the first option is to compute the distances between

the process candidates and the five process variants. The average weighted distance can therefore represent how close each process candidate is to the process variants. This method would be very precise since it directly matches the goal of the algorithm, i.e., to reduce the biases in the system. However, this method is also very expensive since computing the distance between two process models is an $\mathcal{NP}$ problem [12]. The comparison result is shown in Fig. 11.

An alternative way is to compare the *accuracy* and *precision* of each process model (c.f. Section 5.5), since these two measures can represent the closeness between a process model and a type-level order matrix which represents a group of process variants. The advantage of comparing accuracy and precision is that they can be computed in polynomial time, so the comparison would be fast and scalable. However, the disadvantage is that the approximative comparison may not be very precise.

The comparison result for the two groups of process models based on two methods described above is shown in Fig. 11.

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
0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	Accuracy
0.422	0.1	0.837	0.679	0.672	0.521	0.679	0.372	0.596	Precision

Fig. 11. The number of biases when adapting different models

The matrix in Fig. 11 shows how many biases would exist if we set different process models as the new reference process model. For example, if we set S_1 as the new reference model, we therefore can find biases existing in S_2, S_3, S_4, and S_5, and the distances between S_1 to them all equal 2. When given the percentage of each variants as the weight, we can compute the average weighted distances between S_1 and the five variants in the repository. The number is 1.4 as shown in Fig. 11. We can interpret it as follows: if setting S_1 as the new reference model, we need to perform on average 1.4 changes to the reference process model in order to configure it to the process variants in the repository. Similarly, we can compute the average distance when setting other models as the

reference model. The result shows that S' (c.f. Fig. 8), the process model mined out using the method we suggested, has the shortest average weighted distance to the variants. It means, setting S' as the new reference process model would require the least amount effort for process configuration, i.e. we only need to perform on average 1.15 changes to configure S' into the five process variants. Please note that the process models S_{alp} and S_{hrs}, which are produced by the process mining algorithms based on traces, are in fact the two of the worst ones concerning the number of biases in the system. This result also corresponds the analysis we shown in Section 3.

When comparing the accuracy and precision of each process models, the mined out process model, S', is still the best one. Although the accuracy for each process model all equals to 0.5, S' has the highest precision value, 0.837 as shown in Fig. 11, which shows that S' fits closer to the process variants in the repository. In addition, according to the results in Fig. 11, we can generally claim that the higher the precision value is, the shorter the average distance it has compared to the process variants. However, it does not always hold, for instance, S_3 has both higher precision value and average weighted distance when compared to S_1. Such occasional in-consistency is not surprising due to the $\mathcal{NP}$ nature of the distance measure.

Clearly, it is not possible to enumerate all process models available, since it can be infinite. Therefore we can not prove that the process model we mined out are the best one. It is also possible that there exists better ones since the algorithm only search for local optimal as most data mining algorithms do [14]. However, as identified In Fig. 11, the mined out process model is at least better than all the process models currently known and the process models which can be produced by process mining algorithms based on executions. Keeping our search at local optimal will also make our approach applicable to really case, since we therefore can limit the complexity at polynomial level.

7 Mining Process Variants with Different Activity Set

In Section 5 we introduced our method to mine process variants with the same set of activities. In practise, it is often the case that different process variant has different set of activities. In this section, we will discuss the technique to handle this situation. We explain our method by using two examples E_1 and E_2 as described in Fig. 12. We will show how we mine the process variants with different activity set and why we do it in the proposed way.

In Fig. 12, we shows two examples. In example E_1, there are two process variants S_{11} and S_{12} which corresponding to 40% and 60% of the instances in the system. There are two activities: activity `B` and activity `C` that do not appear together in any of the instances. Therefore, there is no process variant which can specify the execution order between activity `B` and `C`. However, when compared to the rest of activities, the execution order of `B` in S_{11} are the same as the execution order of `C` in S_{12}.

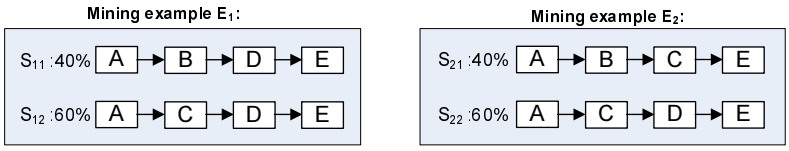


Fig. 12. Two example cases wit different set of activities

In example E_2, there are also two activities, `B` and `D` which do not appear together in any of the process variants. Therefore, there is no direct indication of which execution order between `B` and `D` should be. However, the execution order has been indirectly specified since activity `B` is a predecessor of `C` while activity `D` is a successor of `D`.

As different process variants have different activity sets, we first need to determine which activities appear more often in the process variants so that can ignore the trivial activities. In addition, we also need to see how often a pair of activities appear together in the process variants, since whenever it happens, we can determine the execution order between them. The result based on E_1 is shown in Fig. 13. As the matrix is symmetric, only the lower triangle together with the main diagonal are shown.

In this matrix, the main diagonal shows how frequent each activity appears in the instances. For example,in example E_1 activity `A, B, E` exists in all the instances while activity `B` and `C` only appear in 40% and 60% of the instances. We therefore can define a threshold value to determine which activities should be included in the reference process model. In this way, we can ignore trivial activities in the process variants. For example, only if one activity appears in more than 40% of the instance, we will consider it as an activity in the type level process model. Clearly, determining the threshold value would involve end users, since users would know more about the process model and whether a lot of new activities would or would not be preferable.

Besides the main diagonal, the lower triangle shows how often one pair of activities appears together in the instances. For example, activity `A` and `B` appears together in all the instances, while activity `A` and `C` only appears together in 60% of the instances. We can use this value to compute the type-level order matrix as described in Section 5.1. Since not all the activities will appear in all the instances, the execution order counted in each instance will be discounted by the occurrence value. For example, although activity `A` is a predecessor of activity `B` in only 40% of the instances, such execution order should be discounted since activity `A` and `B` only appear together in 40% of the instances. Therefore, the execution order between `A` and `B` should be $V_{AB} = (0, 1, 0, 0)$, since in all the instances containing both `A` and `B`, activity `A` is always a predecessor of `B`. This is also reflected in the definition of Type-level order matrix (c.f. def 6).

To discount the execution order by the appearance value will not threaten our mining method as discussed in Section 5. The reason is that we are trying to mine a process model out which only represent the *structure* of all the activities.

It is possible that one activity does not appear very often in the instances, so that the appearances between this activity and the rest of activities together are low. However, if the appearances of such activity is higher than the threshold value, we therefore only care where we need to put such activity, i.e. how such activity should be incorporated in the process model. And where the activity should be only depend on its execution order compared to others. Therefore, it is possible that the appearance of one activity is low while the position of such activity is rather stable.

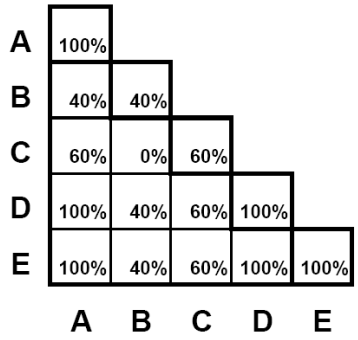


Fig. 13. Occurrence matrix

7.1 Unclear Execution Orders

It is possible that two activities never appear in any process instances so that is not execution order defined in the type-level order matrix. For example, in example E_1, activity `B` and `C` never appears together in one instances so that their appearances value is 0. Therefore it is not possible to determine the relationship between them. There are several ways to handle it:

1. Option 1. We can define the execution order of these activity pairs in the type-level order matrix. We do it by setting the un-specified execution order to $(0.25, 0.25, 0.25, 0.25)$. Which means the execution order between two activities is not clear (i.e. the cohesion is 0 in this case). As $v_0+v_1+v_*+v_- = 1$, we can still re-compute the new type-level order matrix by taking the mean, if new block has been created (c.f. Section 5.4).
2. Option 2. As no execution order has been specified, we can leave the unclear execution order to $(0, 0, 0, 0)$. However, $f(\alpha, \beta)$ (c.f. Section 5.2) is invalid if any of the vector equals to $(0, 0, 0, 0)$. Therefore, we can define $f(\alpha, \beta) = 0$ if α or $\beta = (0, 0, 0, 0)$. In addition, if we need to re-compute the type-level order matrix after a block been created (c.f. Section 5.4), we then do not take such unclear execution order into account. For example, if V_1 and V_2 are to be merged, and $V_1 = (0, 0, 0, 0)$, then the new execution order after merging V_1, V_2 equals V_2.

3. Option 3 is similar to option 3, but we define the unclear relationship to 1, which is another extreme. i.e. $f(\alpha, \beta) = 1$ if $\alpha = (0, 0, 0, 0)$ or $\beta = (0, 0, 0, 0)$.
4. Option 4. We can still leave the unclear execution order to $(0, 0, 0, 0)$, but unlike option 2 and 3, we then ignore this execution order in our computation. i.e. whenever we have such execution order, we define the cohesion to 0 and ignore such vector in computing the separation value (c.f. cohesion, separation in Section 5.2). We also ignore such unclear execution order when we re-compute the type-level order matrix similar to option 2.

We therefore can compare the results if we adept different options in our mining algorithm. Fig. 14 shows the comparison results.

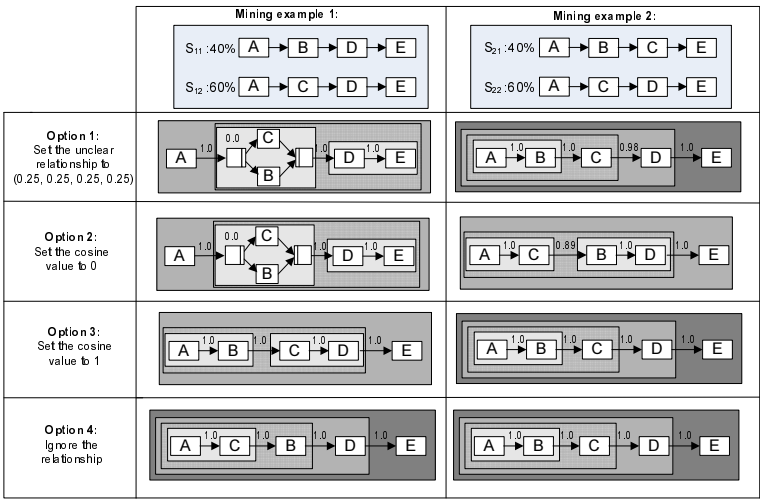


Fig. 14. Different strategy to handle unclear relationship

We therefore can compare the different results when handling the unclear relationship according to the 4 options listed above.

1. If we apply option 1, i.e. set the unclear execution order to (0.25, 0.25, 0.25, 0.25), E_1 and E_2 all fit well since the mined out model confirm to our understanding. In E_1, the relationship between `B` and `C` are unclear (the cohesion is 0.0, which means any type of the execution order is possible), well the relationship towards other's are fixed (i.e. the cohesion to other activities are 1). For E_2, the in-direct relationship between `B` and `D` are also mined out with the cohesion of 0.98. However, it is not 1.0 since it is an indirect relationship. In both examples E_1 and E_2, the average distance between the mined out model and the variants equal to 1.
2. If we apply option 2, i.e. set $f(\alpha, \beta)$ to 0 if any of the vector equals to $(0, 0, 0, 0)$. E_1 works the same as if we apply option 1, but the second one

does not work as we expected. To be more precise, activity `A` is immediate clustered with `C`. And this result in an increase in the average distance to 1.4. The reason is that: since the cosine value is set to 0, the closeness between `B` and `D` is therefore considered as very low. So, adapting option 2 could lead to missing indirect relationship as specified through other activities.

3. If we apply option 3, i.e. set $f(\alpha, \beta)$ to 1 if any of the vector equals to $(0, 0, 0, 0)$, E_2 works as we expected while E_1 does not provide us a model we expected. As activity `A` and `B` are cluster and activity `C` and `D` are cluster, the relationship between these two blocks is very strong, with cohesion of 1.0. However, it should not be the case since the relationship between `B` and `C` are actually unclear. In fact, the strong relationship between the two blocks containing `A,B` and `C,D` only shows the execution order between activity `A` and `D`. The reason is that: since we set the cosine value to 1, activities will be easily clustered to its neighbors, which leads to incorrect cohesion values. So, adapting option 3 will exaggerate the relationship between unclear activities therefore reduce the credibility of the mining result.
4. If we apply option 4, i.e. ignore unclear relationship, the result is similar as option 3. In this case, activities with unclear relationship will be easily clustered with their neighbors, and the relationship between two unclear activities will be also be exaggerated as explained in option 3.

Please note that, here we deliberately give examples process variants in sequential structure (needless to say, the proposed method is also available if process variants contain complex control flow like AND or XOR blocks). The reason is to make difference between process mining based on execution clearer. For example, If we apply mining algorithm based on execution, E_1 will mine a process model similar to our result only with activity `B` and `C` in two branches of an XOR block. In fact, our method can also mine such process model out since the cohesion between the block with activity `B` and `C` is 0, which means any four types of execution orders is possible. However, as our method is focusing on mining process variants rather than traces which only have sequential structure, we do not make such assumption that if two activities never appear together in a log, then they should be in an XOR block. The reason is that two activities does not appear together in an instance does not necessary mean only one of them can be executed. Instead, we do it by setting the cohesion value which shows the credibility of a certain structure to 0. Please also refer to the detailed comparison between execution and biases as discussed in Section 3.

8 Algorithm and Prototype

8.1 Algorithm

To give an overview of the method we described in Section 5 and Section 7. We can formally define the algorithm to perform process variants mining as follows:

The mining starts by deciding the set of activities to be included in the type level process model. If the appearance of an activity is larger than a given

Data: a set of process variants
Result: new type level process model
Do statistics of the appearance of each activity;
if *appearanceOfActivityI* > *threshold value* **then**
  Set activity I in the type level order matrix;
end
Compute order matrix for each process variance;
Build type-level order matrix based on these selected activities;
Do statistics to the appearance of each pair of activities;
if $appearanceA_{ij} \neq 0$ **then**
  Compute execution order of V_{ij} based on process variants;
else
  Set A_{ij} to $(0.25, 0.25, 0.25, 0.25)$.
end
while {*Iteration* < *numberOfActivity -1*} **do**
  Compute similarity table;
  Determine block, execution order and cohesion;
  Generate the block in the out put model;
  Recompute the type-level order matrix;
end
compute precision;
compute accuracy;

Algorithm 1: Process mining based on process variants

threshold, we will include this activity in the reference model. Then we build the type-level order matrix based on the order matrices of each process variants. If the execution order between a pair of activity is not define, we set it to the value of (0.25, 0.25, 0.25, 0.25). After we build the order matrix, we continue cluster activities (blocks) until all the activities are clustered together. In each of these iterations, we need to compute the similarity table to decide which activity to be clustered together, as well their execution order and cohesion. After two activities are blocked together, we need to re-compute the type-level order matrix for the next iteration. The iteration will continue until all activities and blocks are clustered together. After the process model is build up, we can evaluate the model by computing the precision and accuracy of that model.

8.2 Prototype

The algorithm has been tested and implemented using Java. Figure 15 shows an screen shot of the prototype.

We use ADEPT2 Process Template Editor [26] as the tool to generate process variants. For each process model, the editor can generate an XML document with all the information of a process model (like nodes, edges, blocks) been marked up. We therefore store all the process variants in the folder *"instances"* (c.f. Figure 15) for mining.

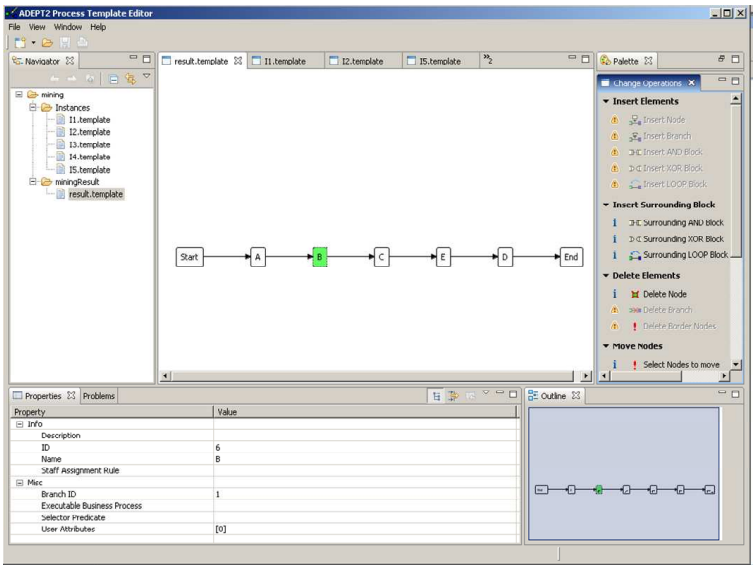


Fig. 15. Screen shot of the prototype

The mining algorithm is developed as an 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 therefore been visualized using the editor. The advantage of using ADEPT2 process template editor is that it can handle the layout of a process model automatically. Therefore, we can just focus on the structure of a process model.

9 Related Work

A variety of techniques for process mining has been suggested in literature [16, 22, 23, 20]. As illustrated in this paper, traditional process mining is different from process variants mining due to its different goal and inputs. [2] presents a method to mine configurable process model based on event logs, however, it still focusing on discovering process model through even log rather than reducing the effort for process configuration. In addition, 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 [14] and maximal subgraph mining [17] as known from graph theory [18]; 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, [7] 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. [24] presents a method for retrieving process variants using a query model but not mining process variants. None of the discussed approaches aims at creating a reference process model, which allows for easy and optimized configuration for process variants.

10 Summary and Outlook

In this paper, we provide an approach to mine block-structured process variants with the goal of reducing the number of biases in the system. The first contribution of this paper is to provide one new process mining direction which is *to reduce the number of process variants* so that the supporting information system for process models is easy manageable. Based on quantitative comparisons, we have shown that this mining goal is different to the one of traditional process mining techniques which try to discover a process model representing all the execution behaviors shown from the execution log [16, 6].

In addition, we provides one polynomial algorithm to perform the mining. Like most of data mining algorithm, the algorithm we suggested also only looks for local optimal. However, in this way we can solve a complex combinatory problem in polynomial time, which provides us the opportunity to scale up on solving real-life problem. We also validate our result in Section 6 that the mined out model is better than all the process models existing in the system, as well as the models derived from the existing process mining algorithms [16]. This paper is, to our knowledge, the first paper to provide algorithm support of process variants mining with the goal of minimizing biases.

The approach looks promising but there are still several questions left open. First is to include more sentential control flow types, like synchronization edge or loop back edge as proposed in ADAPT technique [9] or BEPL [19]. In addition, it is also interesting to design search or heuristic algorithm to limit the difference between the original process model and the process model we mined out. In this way, the process re-engineering effort would be limited so that the mined out process model would be relatively easier to be acceptable.

References

1. G. Halmans, K. Kohl: *Communicating the Variability of a Software Procut Family to Customers*. Software and Systems Modeling, 2(1): 15-36, 2003
2. M. Jansen-Vullers, W.M.P. van der Aalst, M. Rosemann: *Mining Configurable Enterprise Information Systems*. Data and Knowledge Engineering, V56, n3, pp 195-244. 2006
3. D.L. Parnas (1994): *Software Aging*. Proc. 16th Int'l Conf. on Software Engineering, Sorrento, Italy, May 1994, pp. 279-287

4. M. Rosemann, W.M.P. van der Aalst: *A configurable reference modeling language*. Information Systems, 32(1):1-23, 2007
5. B. Weber, S. Rinderle, M. Reichert: *Change Patterns and Change Support Features in Process-Aware Information Systems*. Proc. 19th Int'l Conf. Advanced Information Syst. Eng. (CAiSE'07), LNCS 4495, Norway, 2007, pp. 574-588
6. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm and A. J. M. M. Weijters: *Workflow mining: a survey of issues and approaches*. Data & Knowledge Engineering. Volume 47, Issue 2. pp 237–267. 2003.
7. C.W. Günther, S. Rinderle, M. Reichert, and W.M.P. van der Aalst. Change Mining in Adaptive Process Management Systems. in proceedings of CoopIS 2006, LNCS 4275, pp 309-326. 2006.
8. S.Rinderle. *Schema Evolution in Process Management Systems*. Ph.D thesis, Univ. of Ulm, 2004
9. M. Reichert and P. Dadam. *ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control*. Journal of Intelligent Information Systems, 10(2):93–129, 1998.
10. 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
11. 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
12. C.Li, M.Reichert, A.Wombacher. *Process Similarity Based on High Level Change Operations*. CTIT Technical Report, University of Twente, The Netherlands.
13. W. M. P. van der Aalst, T. Basten. *Identifying Commonalities and Differences in Object Life Cycles Using Behavioral Inheritance*. In proceedings of ICATPN '01. LNCS 2075. pp 32-52. 2001
14. P.N. Tan,M. Steinbach, V. Kumar : *Introduction to Data Mining*. Addison-Wesley, 2006
15. A.Wombacher, M.Rozie: Evaluation of Workflow Similarity Measures in Service Discovery. Service Oriented Electronic Commerce 2006: 51-71
16. http://ga1717.tm.tue.nl/wiki/
17. J. Huan, W. Wang, J. Prins, J. Yang: *SPIN: mining maximal frequent subgraphs from graph databases*. KDD '04: Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining. pp 581–586. 2004
18. K.H.Rosen: *Discrete mathematics and its application*. 5th edition. McGraw-Hill. 2003
19. M.Weske: *Business process management*. Springer. 2007
20. 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.
21. L. Wen, J. Wang, and J.G. Sun. *Detecting Implicit Dependencies Between Tasks from Event Logs*. In APWeb 2006, LNCS 3841, pp 591C603, 2006.
22. 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.
23. A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2006.
24. R.Lu, S.W.Sadiq: *Managing Process Variants as an Information Resource*. BPM06, LNCS 4102, pp 426-431, 2006

25. R.L.Kruse, A.J.Ryba: *Data Structures And Programming Design in C++*. Prentice Hall. 1999.
26. M.Reichert, S.Rinderle, U.Kreher, P.Dadam: *Adaptive Process Management with ADEPT2*. ICDE05: pp 1113-1114, 2005.
27. C. Li, M. Reichert, A. Wombacher: *Issues in Process Variants Mining*. CTIT Technical Report TR-CTIT-08-10. University of Twente. ISSN 1381-3625. 2008.
28. 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,
29. J.Bae, L. Liu, J. Caverlee, W. B. Rouse: *Process Mining, Discovery, and Integration using Distance Measures* ICWS06, pp. 479-488, 2006