Data & Knowledge Engineering 70 (2011) 409–434

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Editorial

Mining business process variants: Challenges, scenarios, algorithms

Chen Li [a,*], Manfred Reichert [b], Andreas Wombacher [c]

[a] Information Systems Group, University of Twente, The Netherlands
[b] Institute of Databases and Information Systems, Ulm University, Germany
[c] Database Group, University of Twente, The Netherlands

ARTICLE INFO

Article history:
Received 25 April 2010
Accepted 18 January 2011
Available online 24 February 2011

Keywords:
Process mining
Process configuration
Process change
Process variant

ABSTRACT

During the last years a new generation of process-aware information systems has emerged, which enables process model configurations at buildtime as well as process instance changes during runtime. Respective model adaptations result in a large number of model variants that are derived from the same process model, but slightly differ in structure. Generally, such model variants are expensive to configure and maintain. In this paper we address two scenarios for learning from process model adaptations and for discovering a reference model out of which the variants can be configured with minimum efforts. The first one is characterized by a reference process model and a collection of related process variants. The goal is to improve the original reference process model such that it fits better to the variant models. The second scenario comprises a collection of process variants, while the original reference model is unknown; i.e., the goal is to "merge" these variants into a new reference process model. We suggest two algorithms that are applicable in both scenarios, but have their pros and cons. We provide a systematic comparison of the two algorithms and further contrast them with conventional process mining techniques. Comparison results indicate good performance of our algorithms and also show that specific techniques are needed for learning from process configurations and adaptations. Finally, we provide results from a case study in automotive industry in which we successfully applied our algorithms.

1. Introduction

In today's dynamic business world success of an enterprise increasingly depends on its ability to react to environmental changes in a quick, flexible and cost-effective way [1–3]. To increase flexibility of Process-Aware Information Systems (PAIS) different approaches have been suggested for adapting pre-modeled processes (i.e., to add, delete or move process activities) [4,5]. Corresponding adaptations are not only needed for customizing a given reference process model to a particular context at buildtime [6,7], but also become necessary for tailoring process instances during runtime in order to deal with exceptional situations and changing needs [4,8]. As example consider medical guidelines [2]. Such guidelines need to be customized to fit the particular healthcare environment in which they are applied. Additional adaptations become necessary when applying a tailored guideline in the context of a particular patient [2]. Generally, respective process model adaptations lead to large collections of process model variants (*process variants* for short) that are derived from the same process model, but slightly differ in structure [1,9]. In case studies we conducted in healthcare and automotive engineering, we identified scenarios with dozens up to hundreds of variants [10].

* Corresponding author. Tel.: +31 53 489 5334; fax: +31 53 489 2927.
E-mail addresses: lic@cs.utwente.nl (C. Li), manfred.reichert@uni-ulm.de (M. Reichert), a.wombacher@utwente.nl (A. Wombacher).

0169-023X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2011.01.005

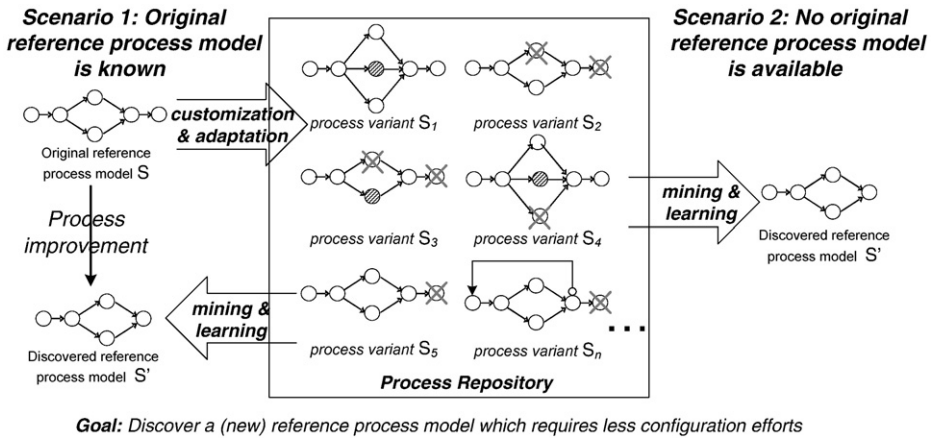


Fig. 1. Different scenarios for discovering reference process models.

1.1. Problem statement

Though considerable efforts have been made to ease process configuration and adaptation [5–8], most existing approaches have not utilized information about such structural adaptations yet [11]. Fig. 1. describes the overall goal of our research. We want to learn from process model adaptations in order to discover a (new) *reference process model* covering the given variants best. By adopting the discovered model in the PAIS, need for future process adaptations and costs for change will decrease. Generally, finding such reference model is by far not trivial when considering control flow patterns like sequence, parallel branching, conditional branching, and loops. Furthermore, "dramatic" changes of the current reference process model might be not always preferred due to implementation costs or social reasons. Fig. 1. further differentiates between two scenarios. In the *first scenario* the process variants are derived by configuring a known reference process model. When mining a new reference process model without considering the current one, however, we might be confronted with significant structural differences between old and new reference models. Process engineers should therefore have the flexibility to control to what degree they want to maximally modify the original reference model to better fit to the given variant collection. Consequently, closeness of the new reference model to the old one and closeness of this model to the variant models act as "counterforces".

The *second scenario* we consider is based on a collection of related process variants, but does not presume knowledge about the original reference process model these variants were derived from. Here we want to discover a reference process model by "merging" these variants without considering the "counterforces" described above. Due to this simplification, we want to design an optimized solution approach (e.g., running faster or discovering better results) than the one suggested in the first scenario.

1.2. Contribution

Based on the assumptions that (1) process models are block-structured and (2) all activities in a process model have unique labels[1], this paper deals with the following research questions:

1. Given the original reference process model and a collection of related process variants, how can we derive a new reference process model that fits "better" to the variants? — In this scenario we want to control the evolution of the reference process model, i.e., to enable process engineers to control to what degree the new reference model "differs" from the original one and how "close" it is to the process variants.
2. Given a collection of process variants without knowledge about the process model they were derived from, how can we discover a suited reference process model in such a way that distance between this reference model and the process variants becomes minimal?
3. Which algorithms foster these two scenarios best and what are their differences? How do these algorithms for mining process variants differ from traditional process mining algorithms focusing on execution behavior?

As input for our analysis we solely require a collection of *process variant models* (and a *reference process model* in the first scenario). In particular, we do NOT presume the existence of *process change logs* which comprise information about the change patterns that were applied when configuring the variants out of the original reference model [13]. Furthermore we measure closeness (or distance) between reference process model and process variant in terms of the number of high-level change operations (e.g., to insert, delete or move activities [8]) needed to transform the reference process model into the respective variant. Clearly, the shorter this distance is, the less the efforts for configuring this variant are.

[1] The block-structure constraint is discussed in Section 2. Regarding unique activity labeling, we refer to [12] for an approach that matches activities with different labels.

In our *first scenario*, we discover a new reference model by performing a sequence of change operations on the original one. In this context, process engineers have the flexibility to control similarity between the original reference model and the newly discovered one, i.e., to specify how many change operations shall be maximally applied to the old reference model when discovering the new one. As benefits, we can control the efforts for updating the reference process model, and we can avoid Spaghetti-like model structures, which is a common challenge in the field of process mining [14,15]. Clearly, most relevant changes (i.e., changes which significantly contribute to reduce average distance between discovered reference model and variants) should be considered first and less relevant ones last. In our *second scenario*, we only need to "merge" the variants without considering an original reference process model as "counterforce". By utilizing this simplification we provide another algorithm which shall perform better than the one designed for the first scenario. To elaborate this we systematically compare these two algorithms for process variants mining.

Finally, we compare the suggested mining techniques with process mining algorithms [16]. The latter aims at discovering a process model by analyzing execution behavior of completed process instances as captured in execution logs [15–18]. The latter typically documents the start/end of each activity execution and therefore reflects behavior of the implemented processes. In principle, process mining techniques can be applied in our context as well. However, they discover models which cover *behavior* best; i.e., their goal is different from ours.

This paper significantly extends our work presented in [19]. It handles more process patterns (e.g., loops), provides more technical and formal details, adds another mining algorithm, and discusses results from a case study. Finally, we compare our algorithms for process variants mining with existing process mining algorithms based on different criteria.

The remainder of this paper is organized as follows. Section 2 gives necessary background information and Section 3 introduces a running example. Section 4 deals with important measures for evaluating process variants in different aspects. Section 5 describes heuristic algorithm we designed for Scenario 1, while Section 6 presents the clustering algorithm for handling Scenario 2. We compare the two algorithms with each other as well as with traditional process mining algorithms in Section 7. Results of a case study in the automotive domain are presented in Section 8. Section 9 discusses related work and Section 10 concludes with a summary.

2. Backgrounds

2.1. Process model

Let $\mathcal{P}$ denote the set of all sound process models. We denote a process model as sound if its execution cannot lead to deadlocks, livelocks or other undesirable states (e.g., unreachable activities) [8,20]. Further, a *process model* $S = (N, E, \ldots) \in \mathcal{P}$ is defined in terms of an Activity Net [8], where N constitutes a set of labeled activities and E a set of control edges (i.e., precedence relations) linking these activities.[2] We consider following process patterns: Sequence, AND-split, AND-join, XOR-split, XOR-join, and Loop [21]. These patterns constitute the core of any process specification language and cover many of the process models we can find in practice [22,23]. Based on them we are able to compose more complex process structures if required (e.g., OR-split can be mapped to AND- and XOR-splits [24]). When only using these basic process patterns, we obtain better understandable and less erroneous models [25]. Fig. 3a depicts a simple example of an Activity Net.

2.2. Block structuring

We assume Activity Nets to be block-structured, i.e., sequences, branchings, and loops are represented as blocks with well-defined *start* and *end* nodes. These blocks may be nested, but must not overlap; i.e., their nesting must be regular [8,26]. In a process model S, a block can be a single activity, a self-contained fragment of S, or S itself. As example consider process model S from Fig. 3. Here {A}, {A,B}, {C,F}, and {A,B,C,D,E,F,G} describe possible blocks contained in S. Note that we can represent a block B as activity set, since the block structure itself becomes clear from process model S; e.g., block {A,B} corresponds to the parallel block with AND-split and AND-join nodes in S. The concept of block-structuring can be found in process specification languages like WS-BPEL and XLANG. Furthermore, process management systems like AristaFlow BPM Suite [27] and CAKE2 [28] emerged, which are applied in a variety of application domains and whose process modeling language is block-structured. When compared with non-block-structured process models, block-structured ones are easier, understandable and have lower error probability [25,29,30]. If a process model is not block-structured, in most cases we can transform it into a block-structured one [25,26,31]. For example, in a case study we analyzed 214 process models from different domains and being expressed in different languages (e.g., Event Process Chains and UML Activity Diagrams). More than 95% of them were block-structured or could be transformed into a block-structured representation [32]. For these reasons, we consider our mining algorithms for block-structured process variant models as highly relevant.

2.3. Process change

A process change is accomplished by applying a sequence of high-level change operations to a given process model S [8]. Such operations structurally modify the initial process model by altering its set of activities and their order relations.

[2] An Activity Net contains more components than node set N and edge set E, which are factored out here.

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 changes performed on initial model S. Then:

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

Examples of high-level change operations include *insert activity*, *delete activity*, and *move activity* as implemented in our ADEPT change framework [8]. While *insert* and *delete* modify the activity set of a process model, *move* changes activity positions and thus model structure. A formal semantics of these change patterns can be found in [33]. For example, *move*(S,A,B,C) shifts activity A from its current position within process model S to the one after activity B and before activity C; *delete*(S,A), in turn, deletes activity A from S. Finally, *insert*(S,A,B,C) adds activity A between B and C. Issues concerning the correct use of these change operations, their generalization, and pre-/post-conditions are described in [8,34]. Though the depicted change operations are discussed in relation to ADEPT, they are generic in the sense that they can be applied in connection with other process meta models as well [4,33]; e.g., a process change as realized in ADEPT can be mapped to the concept of life-cycle inheritance known from Petri Nets [20]. We refer to ADEPT since it covers by far most high-level change patterns and change support features when compared to other adaptive PAIS eChangePatternJournal. Furthermore, with AristaFlow BPM Suite [27], an industrial-strength version of the ADEPT technology has emerged.[3] Based on the given change operations, we define *distance* and *bias* as follows:

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'. We define $d_{(S,S')} = min\{\,|\sigma| \,|\, \sigma \in \mathcal{C}^* \wedge S[\sigma\rangle S'\}$. Furthermore, a sequence of change operations σ with $S[\sigma\rangle S'$ and $|\sigma| = d_{(S,S')}$ is denoted as *bias* $B_{(S,S')}$ between S and S'. □

Distance between process models S and S' corresponds to the minimal number of high-level change operations needed for transforming S into S'. The corresponding sequence of change operations is denoted as *bias* $B_{(S,S')}$ between S and S'.[4] Usually, distance measures complexity for model configuration. As example take Fig. 2. Here, distance between process model S and process variant S_4 is *four*, since we minimally need to apply four change operations to transform S into S' [35]. In general, determining bias and distance between two process models has complexity at $\mathcal{NP}-hard$ level [35]. Note that we consider high-level change operations instead of change primitives (i.e., elementary changes like adding or removing nodes or edges) to measure model distance. This enables us to guarantee soundness of process models and also provides a more meaningful measure for distance [4,35]. Finally, we define the notion of trace:

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

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

We only consider trace logging events about 'real' activities, but no events related to silent ones, i.e., nodes within a process model having no associated action and only existing for control flow purpose [35]. Fig. 2 depicts some examples. At this stage, we consider two process models as being the same if they are *trace equivalent*, i.e., $S \equiv S'$ iff: $\mathcal{T}_S \equiv \mathcal{T}_{S'}$. Like most process mining approaches, the stronger notion of bi-similarity [36] is not considered here.

3. Running example

Fig. 2 depicts an illustrating example of an original reference process model S and 6 process variants $S_i \in \mathcal{P}$ that were configured out of S through structural adaptations. All models are based on patterns like AND-split, AND-join, XOR-split, XOR-join and Loop [21]. Note that the variants do not only differ in structure, but also in respect to their activity sets: e.g., activity X appears in 5 of the 6 variants (except S_2), while Z only appears in S_5. Furthermore variants are weighted. In our context, we define the *weight* w_i of a process variant S_i as the number of process instances executed on the basis of S_i; e.g., 25 instances were executed based on S_1, while 20 instances ran on S_2. If we only know the variants, but have no runtime information about related instance executions, we assume variants to be equally weighted; i.e., every process variant has weight of 1.

We can compute the distance (cf. Def. 2) between original reference model S and each process variant S_i. For example, when comparing S with S_1 we obtain 5 as distance (cf. Fig. 2); i.e., we need to apply five high-level change operations to transform S into S_1: *delete(loop)*, *move*(S,H,I,D), *move*(S,I,J,*endFlow*), *move*(S,J,B,*endFlow*), and *insert*(S,X,E,B) (cf. Def. 1). Based on weight w_i of each variant S_i, we can compute *average weighted distance* between a reference model S and its variants.

[3] Visit www.aristaflow-forum.de for more information and screen casts.

[4] Generally, it is possible to have more than one minimal set of change operations to transform S into S', i.e., a bias of S and S' does not need to be unique. See [20,35] for adetailed discussion of this issue.

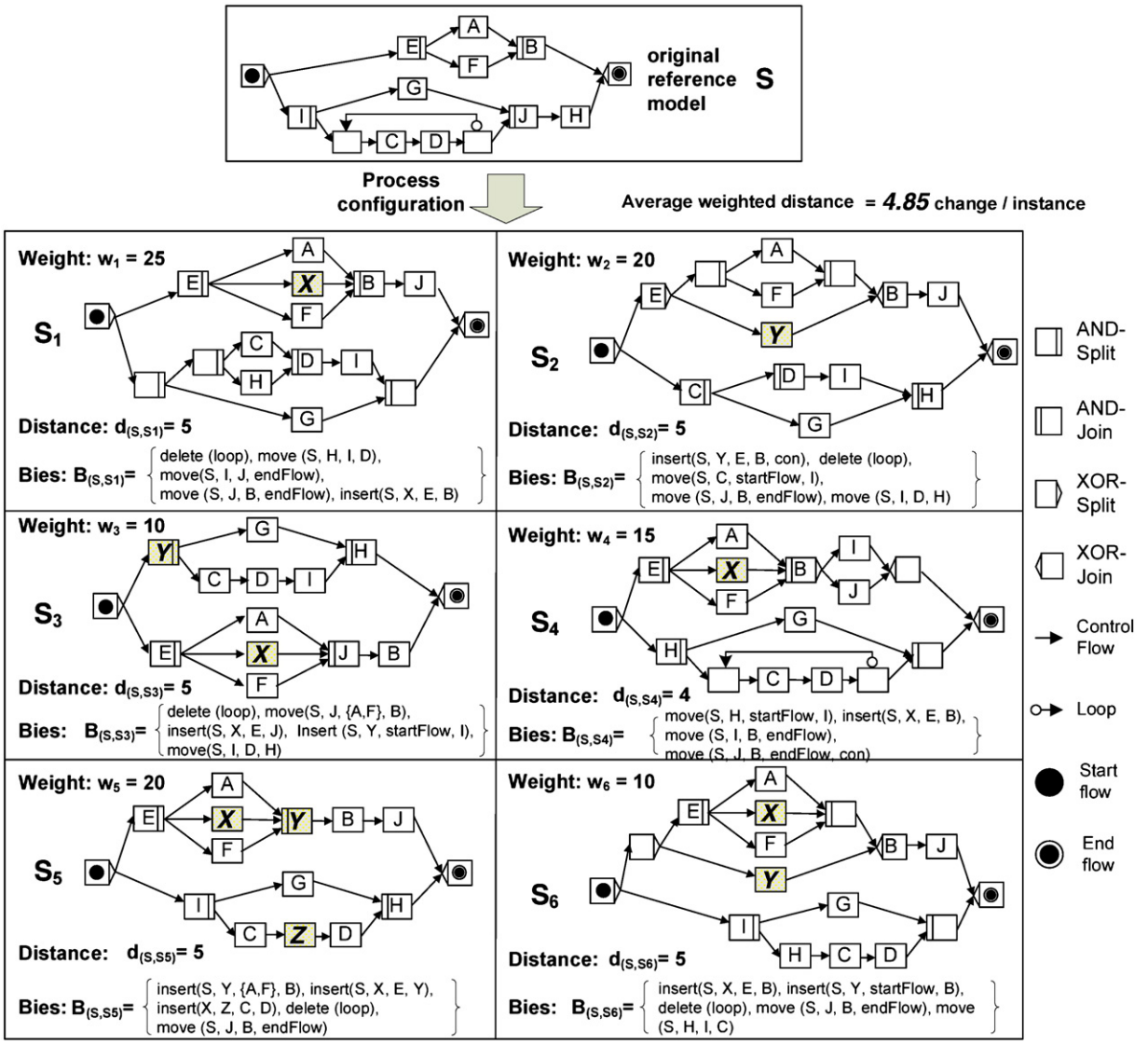


Fig. 2. An illustrating example of a reference process model and related process variants.

Definition 4. *Average weighted distance*. Let $S = (N, E, \ldots) \in \mathcal{P}$ be a reference process model. Let further $\mathcal{M}$ be a set of process variants $S_i = (N_i, E_i, \ldots) \in \mathcal{P}, i = 1, \ldots, n$, with w_i representing the number of process instances that were executed on basis of S_i. The *average weighted distance* $D_{(S,\mathcal{M})}$ between S and $\mathcal{M}$ can be computed as follows:

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

□

The complexity to compute average weighted distance is $\mathcal{NP}-hard$ since the complexity to compute the distance between two variants is $\mathcal{NP}-hard$ (cf. Def. 2). Regarding our example, the distance between S and S_4 is 4, while the distances between S and S_i ($i \neq 4$) correspond to 5, (cf. Fig. 2). When considering variant weights, we obtain as average weighted distance: $(5\times25+5\times20+5\times10+4\times15+5\times20+5\times10)/(25+20+10+15+20+10) = 4.85$. This means we need to perform on average 4.85 high-level change operations to configure a process variant (and related instance respectively) out of the reference process model. Generally, average weighted distance between a reference model and its variants expresses how "*close*" they are.

Our goal is to discover a reference model with shorter average weighted distance to a given collection of (weighted) process variants than the current reference model (Scenario 1) or minimal average weighted distance if the original reference model is unknown (Scenario 2).

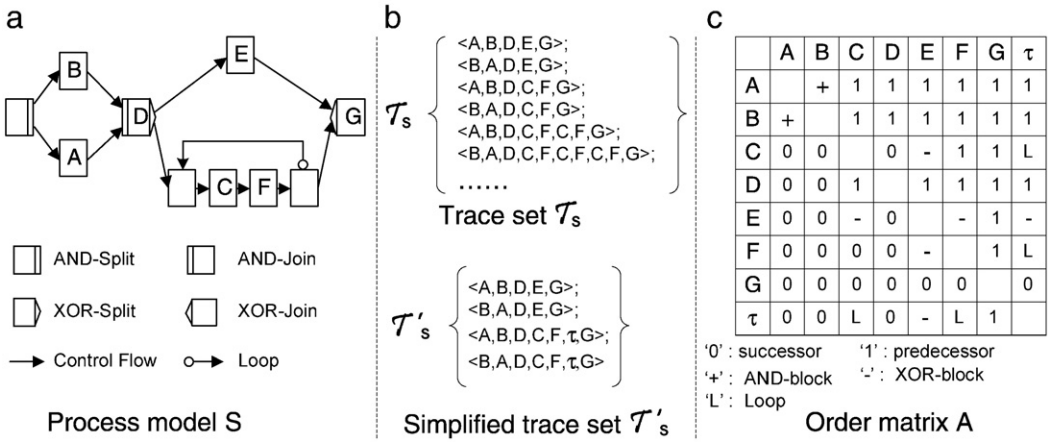


Fig. 3. a) Process model, b) (simplified) Trace set, and c) related order matrix.

4. Matrix-based representations of process models and process variant collections

As basic input for our mining algorithms, we take a collection of process variants and optionally the original reference model they were derived from. In this section, we show how we represent this information in our mining framework.

4.1. Representing block-structured process models as order matrices

One key feature of any change framework is to maintain the structure of unchanged parts of a process model [5,8,27]; e.g., when deleting an activity this neither influences its successors nor predecessors and therefore also not their order relations [33,37]. To incorporate this in our approach, rather than only looking at direct predecessor–successor relations (i.e. control edges), we consider transitive control dependencies for each pair of activities; i.e., for a given process model $S = (N, E, \ldots) \in \mathcal{P}$ and activities a_i, $a_j \in N$, $a_i \neq a_j$ we examine structural order relations (including transitive ones). Logically, we determine order relations by considering all traces producible by model S (cf. Def. 3). Fig. 3a shows an example of a process model S. Here, $\mathcal{T}_s$ constitutes an infinite trace set due to the presence of the loop-block in S (cf. Fig. 3b).

4.1.1. Simplification of infinite trace sets

Such infinite number of traces precludes us to perform any detailed analysis of the trace set. Therefore we need to transform it into a finite representation before. One common approach to describe a string with infinite length is to represent it as a finite set of n-gram lists [38]. The general idea behind is to represent a single string as ordered list of substrings with length n (so-called *n-grams*). In particular, only the first occurrence of an n-gram is considered, while later occurrences of the same n-gram are omitted. Thus, an n-gram list represents a collection of strings having different lengths. In particular, an infinite language can be represented as a finite set of n-gram lists; e.g., string $<ababababab>$ can be represented as 2-gram $<\$a, ab, ba, b\#>$, where \$ (#) represents the start (end) of the string. Such an approach is commonly used for analyzing loop structures in process models [39] or – more generally – in the context of text indexing for substring matching [40]. Inspired by this, we define the notion of *simplified trace set* as follows:

Definition 5. *Simplified trace set.* Let S be a process model and $\mathcal{T}_s$ denote the trace set producible on S. Let B_k, $k = (1,\ldots,K)$ be loop-blocks in S, and $\mathcal{T}_{B_k}$ denote the set of traces producible by the body of loop block B_k. Let further $(t_{B_k})^m$ be a sequence of $m \in \mathbb{N}$ traces $<t^1_{B_k}, t^2_{B_k}, \ldots, t^m_{B_k}>$ with $t^j_{B_k} \in \mathcal{T}_{B_k}$, $j \in \{1,\ldots,m\}$. We additionally define $(t_{B_k})^0 \equiv <>$ as empty sequence. If we only consider the activities corresponding to B_k, in any trace $t \in \mathcal{T}_s$ producible on S, t either has no entries [5] or must have structure $<t^*_{B_k}, (t_{B_k})^m>$, with $t^*_{B_k} \in \mathcal{T}_{B_k}$ representing the first loop iteration and $m \in \mathbb{N}_0$ being the number of additional iterations and loop-block B_k was executed according to trace t. We can simplify this structure by using $<t_{B_k}, \tau_k>$ instead, where τ_k refers to $(t_{B_k})^m$. When simplifying trace set $\mathcal{T}_s$ this way, we obtain a finite set of traces $\mathcal{T}'_s$ which we denote as Simplified Trace Set of process model S. □

In our simplified representation of a trace $t \in \mathcal{T}_s$, we only consider the first occurrence of trace $t^*_{B_k}$ producible by loop-block B_k, while omitting others that occur later within t. Instead, we represent such repetitive entries by a silent activity τ_k, which has no associated action, but solely exists to indicate omission of other t_{B_k} appearing later in trace t; i.e., τ_k represents the iterative execution of loop-block B_k as captured in t.[6] When omitting repetitive entries within trace set $\mathcal{T}_s$, we obtain a finite trace set $\mathcal{T}'_s$ for further analysis. Note that when dealing with nested loops (i.e., a loop-block B_k contains another loop-block B_j), we first need to

[5] This means the loop-block B_k has not been executed at all.

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

analyze B_j and then B_k; i.e., we need to first define τ_j to represent the iterative execution of loop-block B_j as captured in trace t and then we define τ_k to represent loop-block B_k. As example consider process model S in Fig. 3a. Let B^* denote the loop-block comprising activities C and F. The trace set this block can produce corresponds to {<C,F>}. Therefore, any trace $t \in \mathcal{T}_S$ containing C and F has structure <C, F, $(C,F)^m$> with $m \in \mathbb{N}_0$ depending on the number of times the loop iterates; e.g., <C,F>, <C,F,C,F> and <C,F,C,F,C,F> are all valid traces producible by the loop-block. Let us define a silent activity τ corresponding to block B^*. Then we can simplify these traces by <C,F,τ> where τ refers to the sequence of the traces producible on B^*. This way, we can simplify infinite trace set $\mathcal{T}_S$ to finite set $\mathcal{T}_S' = \{\langle A,B,D,E,G\rangle, \langle B,A,D,E,G\rangle, \langle A,B,D,C,F,\tau,G\rangle, \langle B,A,D,C,F,\tau,G\rangle\}$ (cf. Fig. 3b).

4.1.2. Representing a process model as order matrix

For process model S, the analysis results concerning its trace set $\mathcal{T}_S$ are aggregated in a so-called *order matrix* A, which considers five types of order relations (cf. Def. 6):

Definition 6. *Order matrix.* Let $S = (N, E, \ldots) \in \mathcal{P}$ be a process model with activity set $N = \{a_1, a_2, \ldots, a_n\}$. Let further $\mathcal{T}_S$ denote the set of all traces producible on S and let $\mathcal{T}_S'$ be the simplified trace set of S according to Def. 5. Finally let B_k, $k = (1,\ldots,K)$ denote loop-blocks in S and for every B_k let τ_k, $k = (1,\ldots,K)$ be a silent activity representing the iterative structure producible by B_k in $\mathcal{T}_S'$. Then:

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

- A_{a_i,a_j} = '1' iff: $(\forall t \in \mathcal{T}_S' \text{ with } a_i, a_j \in t \Rightarrow t(a_i \prec a_j))$
  If for all producible traces containing activities a_i and a_j, a_i always appears BEFORE a_j, we set A_{a_i,a_j} to '**1**', i.e., a_i always precedes a_j in the flow of control.
- A_{a_i,a_j} = '0' iff: $(\forall t \in \mathcal{T}_S' \text{ with } a_i, a_j \in t \Rightarrow t(a_j \prec a_i))$.
  If for all producible traces containing activities a_i and a_j, a_i always appears AFTER a_j, we set A_{a_i,a_j} to '0', i.e. a_i always succeeds a_j in the flow of control.
- A_{a_i,a_j} = '+' iff: $(\exists t_1 \in \mathcal{T}_S' \text{ with } a_i, a_j \in t_1 \wedge t_1(a_i \prec a_j)) \wedge (\exists t_2 \in \mathcal{T}_S' \text{ with } a_i, a_j \in t_2 \wedge t_2(a_j \prec a_i))$.
  If there exists at least one producible trace in which a_i appears before a_j and another one in which a_i appears after a_j, we set A_{a_i,a_j} to '+'; i.e., a_i and a_j are contained in different parallel branches.
- A_{a_i,a_j} = '-' iff: $(\neg \exists t \in \mathcal{T}_S' : a_i \in t \wedge a_j \in t)$.
  If there is no producible trace containing both activities a_i and a_j, we set A_{a_i,a_j} to '−', i.e. a_i and a_j are contained in different branches of a conditional branching.
- A_{a_i,a_j} = 'L', iff: $((a_i \in B_k \wedge a_j = \tau_k) \vee (a_j \in B_k \wedge a_i = \tau_k))$.
  For any activity a_i in a loop-block B_k, we define order relation A_{a_i,τ_k} between it and the corresponding silent activity τ_k as '**L**'. □

The first four order relations {1,0,+,−} specify the precedence relations between activities as captured in the trace set, while the last one {L} indicates loop structures within the trace set. As example consider Fig. 3c which depicts the order matrix of process model S. Since S contains one Loop-block, a silent activity τ is added to this order matrix. Note that this order matrix contains all five order relations from Def. 6. For example, activities E and C will never appear in the same trace of the simplified trace set since they are contained in different branches of an XOR block. Therefore, we assign '−' to matrix element A_{EC}. Since in all producible traces, which contain both B and G, B always appears before G, we further obtain order relations A_{BG} = '1' and A_{GB} = '0'. Special attention should be paid to the order relations between silent activity τ and the other activities. The order relation between τ and activities C and F is set to 'L', since both C and F are contained in the loop-block; in respect to all remaining activities, τ has the same order relations as C or F have. Note that the main diagonal of an order matrix is empty since we do not compare an activity with itself.

Generally, it is not a good idea to first enumerate all traces of a process model and then to analyze the order relations captured by them. Note that the trace set of a process model can become extremely large, particularly if the model contains multiple AND-blocks or even infinite if it contains loop-blocks. In [41] we introduced algorithms for transforming a block-structured process model into its corresponding order matrix and vice verse. Complexity of these two algorithms is $\mathcal{O}(2n^2)$, where n equals the number of activities plus the number of loop-blocks contained in the process model. In [41] we have further proven that such order matrix constitutes a unique representation of a block-structured process model; i.e., if we transform a process model into an order matrix and then transform this matrix back into a process model, the two process models are trace equivalent; i.e., they cover the same behavior [36].

4.2. Representing a collection of process variants as aggregated order matrix

Based on the notion of Order Matrix (cf. Def. 6), we introduce *Aggregated Order Matrix* to represent a collection of process model variants. First, we compute the order matrix of each process variant. Regarding our running example from Fig. 2, we need to compute six order matrices (cf. Fig. 4). Due to space limitations, Fig. 4 only shows a partial view of them here (i.e., activities H,I,J,X,Y,Z as well as silent activity τ representing the Loop-block). Following this, we analyze the order relation for each pair of activities based on all derived order matrices. As the order relation between two activities might be not the same in all order matrices, this analysis does not result in a fixed relation, but in a distribution for the five types of order relations (cf. Def. 6). Regarding our example, in 60% of all cases H succeeds I (as in S_2, S_3, S_5 and S_6), in 25% of all cases H precedes I (as in S_1), and in 15% of all cases H and I are contained in different branches of an XOR block (as in S_4) (cf. Fig. 4). Generally, for a given variant collection we define the order relation between activities a and b as a 5-dimensional vector $V_{ab} = (v_{ab}^0, v_{ab}^1, v_{ab}^+, v_{ab}^-, v_{ab}^L)$. Each field corresponds to the relative frequency of the respective relation type ('0','1','+' or '−','L') as specified in Def. 6. Take our running example and consider Fig. 4:

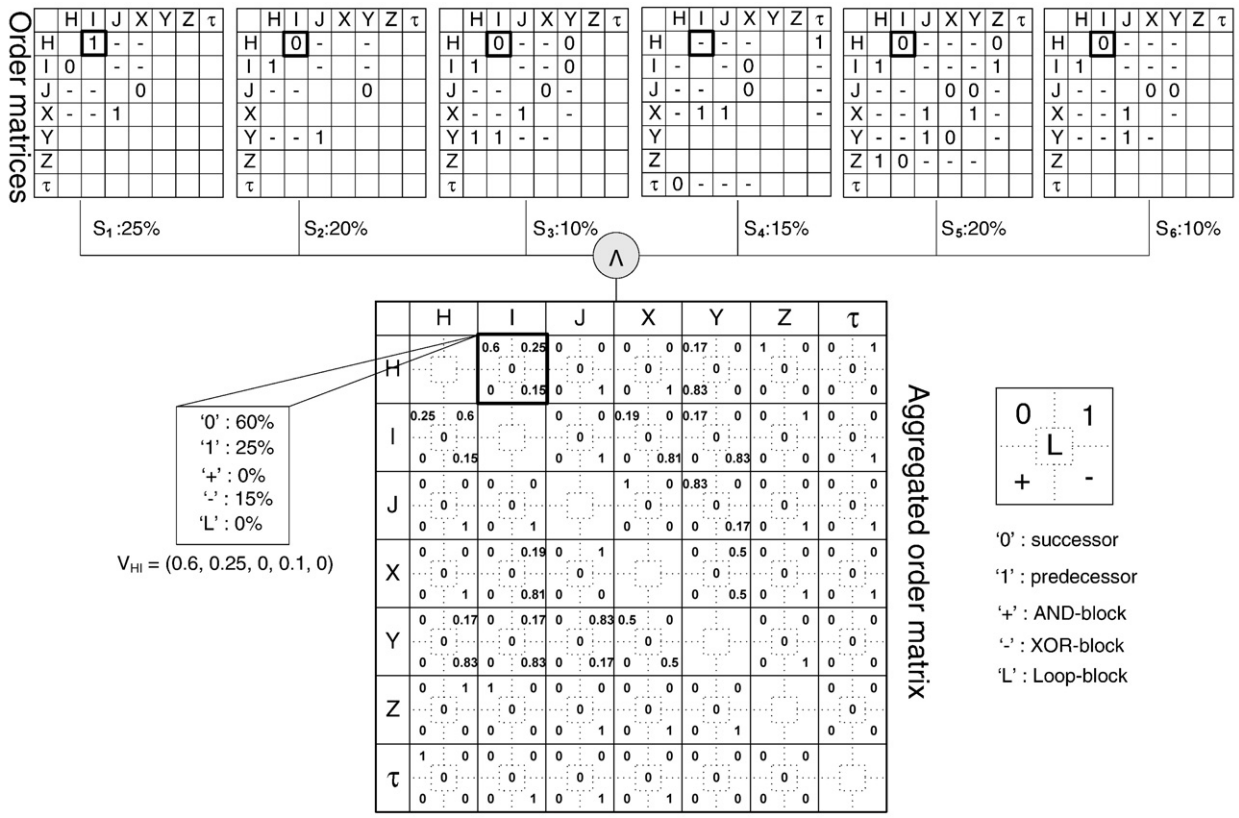


Fig. 4. Aggregated order matrix based on process variants.

$v^1_{HI} = 0.25$ corresponds to the frequency of all order matrices with activities H and I having order relationship '1', i.e., all cases for which H precedes I; we obtain $V_{HI} = (0.6, 0.25, 0, 0.15, 0)$.

Definition 7. *Aggregated order matrix.* Let $S_i = (N_i, E_i, \ldots) \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants with activity sets N_i. Let further A_i be the order matrix of S_i, and w_i be the number of process instances that were executed on S_i. The *Aggregated Order Matrix* of all process variants is defined as 2-dimensional matrix $V_{m \times m}$ with $m = |\cup N_i|$ and each matrix element $v_{a_j a_k} = (v^0_{a_j a_k}, v^1_{a_j a_k}, v^+_{a_j a_k}, v^-_{a_j a_k}, v^L_{a_j a_k})$ being a 5-dimensional vector. For $\diamond \in \{0, 1, +, -, L\}$, element $v^{\diamond}_{a_j a_k}$ expresses to what percentage, activities a_j and a_k have order relation $\diamond$ within the given variant collection $S_1, \ldots, S_n$. Formally: $\forall a_j, a_k \in \cup N_i, a_j \neq a_k$:

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

□

Fig. 4 partially shows the aggregated order matrix V for the process variants from Fig. 1. Due to space limitations, we only consider order relations for activities H,I,J,X,Y,Z, and silent activity τ which represents the Loop-block.

4.3. Measuring the activities frequencies within variant collection

Generally, the order relations computed by an aggregated order matrix may be not equally important. For example, relationship V_{HI} between H and I (cf. Fig. 4) is more important than relation V_{HZ}, since H and I appear together in all six process variants while H and Z only co-occur in S_5 (cf. Fig. 1). We introduce *co-existence matrix CE* in order to represent the importance of the different order relations occurring within an aggregated order matrix V.

Definition 8. *Coexistence matrix.* Let $S_i = (N_i, E_i, \ldots) \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants with activity sets N_i. Let further w_i represent the number of instances that were executed based on S_i. The *coexistence matrix* of variant collection $\{S_1, \ldots, S_n\}$ is then defined as 2-dimensional matrix $CE_{m \times m}$ with $m = |\cup N_i|$. Each matrix element $CE_{a_j a_k}$ corresponds to the relative frequency with which activities a_j and a_k appear together within the given collection of variants. Formally: $\forall a_j, a_k \in \cup N_i, a_j \neq a_k$:

$$CE_{a_j a_k} = \frac{\sum_{S_i : a_j, a_k \in N_i} w_i}{\sum_{i=1}^{n} w_i} \tag{3}$$

□

	H	I	J	X	Y	Z	τ
H		1	1	0.8	0.6	0.2	0.15
I	1		1	0.8	0.6	0.2	0.15
J	1	1		0.8	0.6	0.2	0.15
X	0.8	0.8	0.8		0.4	0.2	0.15
Y	0.6	0.6	0.6	0.4		0.2	0
Z	0.2	0.2	0.2	0.2	0.2		0
τ	0.15	0.15	0.15	0.15	0	0	

Fig. 5. Coexistence Matrix.

Fig. 5 shows the coexistence matrix for our running example. Again, we only depict the coexistence matrix for activities H,I,J,X, Y,Z, and silent activity τ; e.g., we obtain $CE_{\mathrm{HI}=1}$ and $CE_{\mathrm{HZ}}=0.2$. This indicates that order relation between H and I is more important than the one between H and Z.

For a given variants collection, we can further measure how frequent each activity a_i appears using *activity frequency*:

Definition 9. *Activity frequency.* Let $S_i = (N_i, E_i, \ldots) \in \mathcal{P}$, $i = 1,2,\ldots,n$ be a collection of variants with weights w_i representing the number of instances that were executed on S_i. For each $a_j \in \cup_{i=1}^{n} N_i$, we define $g(a_j)$ as the relative frequency with which a_j appears within the given variant collection. Formally:

$$g\left(a_j\right) = \frac{\sum_{S_i : a_j \in N_i} w_i}{\sum_{i=1}^{n} w_i} \tag{4}$$

□

Table 1 shows relative frequency of activities contained in the process variants of our running example (cf. Fig. 2); e.g., activity X is present in 80% of the variants (i.e., in S_1, S_3, S_4, S_5, and S_6), while Z only occurs in S_5 (i.e., 20% of the variants). Since S_4 contains a loop-block, we obtain 15% as frequency with which silent activity τ occurs (cf. Def. 6).

5. Scenario 1: evolving reference process models by learning from past model adaptations: a heuristic approach

As discussed, measuring the distance between two models (cf. Def. 2) is an $\mathcal{NP}-hard$ problem, i.e., the time for computing distance is exponential to the size of the process models. Consequently, the problem set out in our research question (i.e., finding a reference process model with minimal average weighted distance to the variants) is an $\mathcal{NP}-hard$ problem as well. When encountering real-life cases (i.e., dozens up to hundreds of variants with complex structure), finding "the optimum" would either be too time-consuming or simply be not feasible. In this section, we present a *heuristic search algorithm* for mining process variants, while controlling the maximal distance between old and new reference process model (cf. Scenario 1).

Heuristic algorithms are widely used in fields like Artificial Intelligence [42], Data Mining [43] and Machine Learning [44]. A problem employs heuristics when "it may have an exact solution, but the computational cost of finding it may be prohibitive" [42]. Although heuristic algorithms do not aim at finding the "real optimum" (i.e., it is neither possible to theoretically prove that the discovered result is the optimum nor can we say how close it is to the optimum), they are widely used in practice. Usually heuristic algorithms provide a nice balance between goodness of the discovered solution and computation time needed for finding it [42]. Basically, our heuristic algorithm for process variants mining works as follows:

1. Use original reference model S as starting point.
2. Search for all neighboring process models with distance 1 to the currently considered reference process model. If we are able to find a better model S' among these candidate models (i.e., one with lower average weighted distance to the given variant collection compared to S), we replace S by S'.
3. Repeat Step 2 until we either cannot find a better model or the maximally allowed distance between original and new reference process model is reached. S' then corresponds to our discovered reference model.

Generally, most important for any heuristic search algorithm are two aspects: the *heuristic measure* and the *algorithm* that uses heuristics to search the state space. Section 5.1 introduces our *structural fitness function* which measures the quality of a particular candidate model, and Section 5.2 then introduces a *best-first search* algorithm to search the state space.

Table 1
Relative frequency of each activity within the given variant collection.

Activity	A	B	C	D	E	F	G	H	I	J	X	Y	Z	τ
$g(a_j)$	1	1	1	1	1	1	1	1	1	1	0.8	0.6	0.2	0.15

5.1. Fitness function

Generally, the fitness function of any heuristic search algorithm should be quickly computable. Since search space may become very large, we must be able to make a quick decision on which path to choose next. As discussed, average weighted distance (cf. Def. 4) would be not a good choice since complexity for computing it is $\mathcal{NP}-hard$. This section introduces a *fitness function*, which can be used to approximately measure "closeness" between a candidate reference model and the given collection of variants. In particular, it can be computed in polynomial time. Like in most heuristic search algorithms, the chosen fitness function is a "reasonable guessing" rather than a precise measurement. Therefore, in Section 5.4 we investigate correlation between fitness function and average weighted distance.

5.1.1. Activity coverage

For a candidate reference process model $S_c = (N_c, E_c, \ldots) \in \mathcal{P}$, we first measure to what degree its activity set N_c covers the activities that occur in the variant collection. Note that we also consider silent activities τ_k representing loop-blocks B_k of S_c in the corresponding order matrix A_c (cf. Def. 6). We denote this measure as *activity coverage* $AC(S_c)$ of S_c.

Definition 10. *Activity coverage.* Let $S_i = (N_i, E_i, \ldots)\ i = 1, \ldots, n$ be a collection of process variants. Let further $M = \cup_{i=1}^{n} N_i$ be the set of activities that are present in at least one of the variants. Let further N_c be the activity set of candidate process model S_c. Given activity frequency $g(a_j)$ of each $a_j \in M$, we compute activity coverage $AC(S_c)$ of model S_c as follows:

$$AC(S_c) = \frac{\sum_{a_j \in N_c} g\left(a_j\right)}{\sum_{a_j \in M} g\left(a_j\right)} \tag{5}$$

□

Obviously, $AC(S_c) \in [0,1]$ holds. Let us take original reference model S as candidate model. It contains activities A, B, C, D, E, F, G, H, I, J, and τ (which represents the loop-block). Its activity coverage *AC(S)* represents to what degree S covers the activities in the variant collection; *AC(S)* corresponds to $\frac{10.15}{11.8} = 0.860$.

5.1.2. Structure fitness of a candidate process models

$AC(S_c)$ measures how representative activity set N_c of candidate model S_c is in respect to the given variant collection. However, it does not state anything about the structure of candidate model S_c (i.e., order relations). We therefore introduce *structure fitting* $SF(S_c)$ as the second important metrics. It measures to what degree S_c structurally fits to the given variants collection. We use *aggregated order matrix* (cf. Def. 7) and *coexistence matrix* (cf. Def. 8) for this purpose.

Since we can represent a candidate process model S_c by its corresponding order matrix A_c (cf. Def. 6), we determine structure fitting $SF(S_c)$ between S_c and the variants by measuring how similar order matrix A_c and the aggregated order matrix V (representing the variants) are. Take original reference model S as candidate process model S_c (i.e., $S_c := S$). Obviously, A_{HI} = '0' holds, i.e., H succeeds I in S (cf. Fig. 1). Consider now the aggregated order matrix V of the variants (cf. Fig. 3). Here order relation between H and I is represented by the 5-dimensional vector $V_{HI} = (0.6, 0.25, 0, 0.15, 0)$. If we now want to compare how close A_{HI} and V_{HI} are, we first need to build an aggregated order matrix V^c purely based on our candidate process model S_c (S in our case). Trivially, as order relation between H and I in V^c, we obtain $V^c_{HI} = (1,0,0,0,0)$. We then compare V_{HI} (which represents the variants) with V^c_{HI} (which represents the reference model). We use Euclidean metrics $f(\alpha,\beta)$ to measure closeness between two vectors $\alpha = (x_1, x_2, \ldots, x_n)$ and $\beta = (y_1, y_2, \ldots, y_n)$:

$$f(\alpha,\beta) = \frac{\alpha \cdot \beta}{|\alpha| \cdot |\beta|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}} \in [0,1] \tag{6}$$

$f(\alpha,\beta)$ computes cosine value of the angle θ between vectors α and β in Euclidean space. If $f(\alpha,\beta) = 1$ holds, α and β exactly match in their directions; $f(\alpha,\beta) = 0$ means, they do not match at all. Regarding our running example, we obtain $f(V_{HI}, V^c_{HI}) = 0.899$. This indicates high similarity between the order relation of H and I in the candidate process model with those captured by the variants. Based on Euclidean metrics, which measures *similarity* between the order relations, and Coexistence matrix *CE* (cf. Def. 8), which measures *importance* of the order relations, we formally define structure fitness $SF(S_c)$ of a candidate model S_c as follows:

Definition 11. *Structure fitness.* Let $S_i = (N_i, E_i, \ldots) \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants with activity sets N_i. Let further *CE* be the coexistence matrix and V be the aggregated order matrix of the variants collection. For candidate model S_c, let

$m = |N_c|$ correspond to the number of activities in S_c. Finally let V^c be the aggregated order matrix of S_c. Then we compute *structure fitness* $SF(S_c)$ as follows:

$$SF(S_c) = \frac{\sum_{j=1}^{m}\sum_{k=1,k\neq j}^{m}\left(f\left(V_{a_j a_k}, V^c_{a_j a_k}\right) \times CE_{a_j a_k}\right)}{m \times (m-1)} \in [0,1] \quad (7)$$

□

For every pair of activities $a_j, a_k \in N_c, j \neq k$, we first compute similarity of corresponding order relations (as captured by V and V_c) in terms of $f(V_{a_j a_k}, V^c_{a_j a_k})$. Second we determine importance of these order relations by calculating $CE_{a_j a_k}$. Structure fitness $SF(S_c)$ of candidate model S_c then equals average of the similarity multiplied by importance of every order relation. For our example from Fig. 2 we obtain $SF(S) = 0.632$ when choosing S as candidate model.

5.1.3. Fitness function

Based on $AC(S_c)$ and structure fitting $SF(S_c)$, we can compute fitness $Fit(S_c)$ of a candidate model S_c as follows:

Definition 12. *Fitness.* Let $AC(S_c)$ be activity coverage of candidate model S_c and $SF(S_c)$ be its structure fitting. We compute *fitness* $Fit(S_c)$ of S_c as follows: $Fit(S_c) = AC(S_c) \times SF(S_c)$. □

As $AC(S_c) \in [0,1]$ and $SF(S_c) \in [0,1]$, $Fit(S_c) \in [0,1]$ holds as well. $Fit(S_c)$ indicates how "close" candidate model S_c is to the given collection of variants. If $Fit(S_c) = 1$, S_c will perfectly fit to the variants; i.e., no further adaptation is needed. Generally, the higher $Fit(S_c)$ is, the closer S_c is to the variants and the less configuration efforts are required. In our example from Fig. 2, original reference model S has $Fit(S) = AC(S) \times SF(S) = 0.860 \times 0.632 = 0.543$. As fitness of candidate model S_c is evaluated by multiplying activity coverage $AC(S_c)$ with structure fitting $SF(S_c)$, a high value for $Fit(S_c)$ does not only mean that S_c structurally fits well to the process variants, but also that a reasonable number of activities is considered in the candidate model.

5.2. Constructing the search tree

We now show how to find candidate process models. We present a *best-first* algorithm for constructing a search tree to find the best candidate model in the search space.

5.2.1. The search tree

Remember the general overview we gave on our heuristic search approach at the beginning of Section 5. Starting with the current candidate model S_c, in each iteration we search for its direct "neighbors" (i.e., process models which have distance 1 to S_c). Thereby we try to find a better candidate model S'_c with higher fitness value. Generally for a given process model S_c, we construct a neighbor model by applying *ONE* insert, delete, or move operation to S_c. All activities $a_j \in \cup N_i$ which appear in at least one variant are candidate activities for change. While an insert operation adds an activity $a_j \notin N_c$ to S_c, the other two operations delete or move an activity $a_j \in N_c$ already present in S_c.

Generally, numerous process models can result when changing one particular activity a_j on S_c. Note that the positions where we can insert ($a_j \notin N_c$) or move ($a_j \in N_c$) activity a_j can be numerous. Section 5.2.2 provides details on how to find all process models resulting from the change of one particular activity a_j on S_c. First of all, we assume that we have already identified these neighbor models, including the one with highest fitness value (denoted as the *best kid* S^j_{kid} of S_c when changing a_j). Fig. 6 illustrates our search tree. Our search algorithm starts with setting the original reference model S as initial state, i.e., $S_c = S$ (cf. Fig. 6). We further define AS as *active activity set*, which contains all activities that might be subject to change. At the beginning $AS = \{a_j | a_j \in \cup_{i=1}^{n} N_i\}$ contains all activities that appear in at least one variant S_i. For each activity $a_j \in AS$ we then determine the corresponding best kid S^j_{kid} of S_c. If S^j_{kid} has higher fitness value than S_c, we mark it; otherwise, we remove a_j from AS (cf. Fig. 6). Afterwards, we choose the model with highest fitness value S^j_{kid} among all best kids S^j_{kid}, and denote this model as *best sibling* S_{sib}. We then set S_{sib} as the first intermediate search result and replace S_c by S_{sib} for further search. Finally, we remove a_j from AS.

The described search method continues iteratively until termination condition is met, i.e., we either cannot find a better model or *allowed search distance* is reached. Consequently, the process engineer obtains the flexibility to control to what degree the discovered reference process model shall differ from the original one. The final search result S_{sib} corresponds to our discovered reference model S' (the node marked by a bull's eye and circle in Fig. 6). Please see Appendix A for an algorithm formally describing the above steps.

5.2.2. Changing one particular activity

Section 5.2.1 showed how to construct a search tree by comparing best kids S^j_{kid}. We now discuss how to find such best kid S^j_{kid}, i.e., how to find all "neighbors" of candidate model S_c by performing *one* change operation in relation to a particular activity a_j. Consequently, S^j_{kid} is the model with highest fitness value among all considered ones. Regarding an activity a_j, we consider three types of basic change operations: *delete, move* and *insert*. The neighbor model resulting from the deletion of $a_j \in N_c$ can be easily determined by removing a_j from the process model and its order matrix [35]; movement of a_j can be simulated by deleting a_j and sub-sequently re-inserting it at the desired position. Thus, the basic challenge in finding neighbors of candidate model S_c is to apply one activity *insertion* such that *block structuring* and *soundness* of the resulting model are preserved. Obviously, the positions

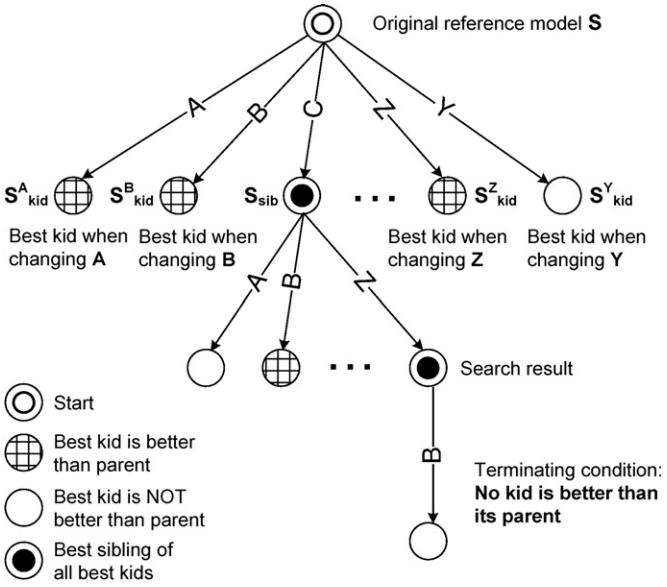


Fig. 6. Constructing the search tree.

where we can (correctly) insert a_j into S_c are our subjects of interest. Fig. 7 provides an example. Given model S_c, we want to find all process models that may result when inserting X into S_c. We apply two steps to "simulate" our activity insertion.

Step 1 block-enumeration. First, we enumerate all possible blocks, candidate model S_c contains. A block can be an atomic activity, a self-contained fragment, or S_c itself. Let $S = (N, E, \ldots) \in \mathcal{P}$ be a process model with $N = \{a_1, \ldots, a_n\}$. Let further textitA be the order matrix of S. Two activities a_i and a_j can form a block iff: $\forall a_k \in N \setminus \{a_i, a_j\} : A_{a_i a_k} = A_{a_j a_k}$, i.e., two activities can form a block iff they have same order relations in respect to remaining activities. As example consider Fig. 7a. Here activities C and D can form a block, since they have same order relations to activities G, H, I, and J. As extension, two blocks B_j and B_k can be merged to a bigger one iff $[(a_\alpha, a_\beta, a_\gamma) \in B_j \times B_k \times (N \setminus B_j \cup B_k) : A_{a_\alpha a_\gamma} = A_{a_\beta a_\gamma}]$ holds; i.e., all activities $a_\alpha \in B_j$, $a_\beta \in B_k$ show same order relations to the activities outside the

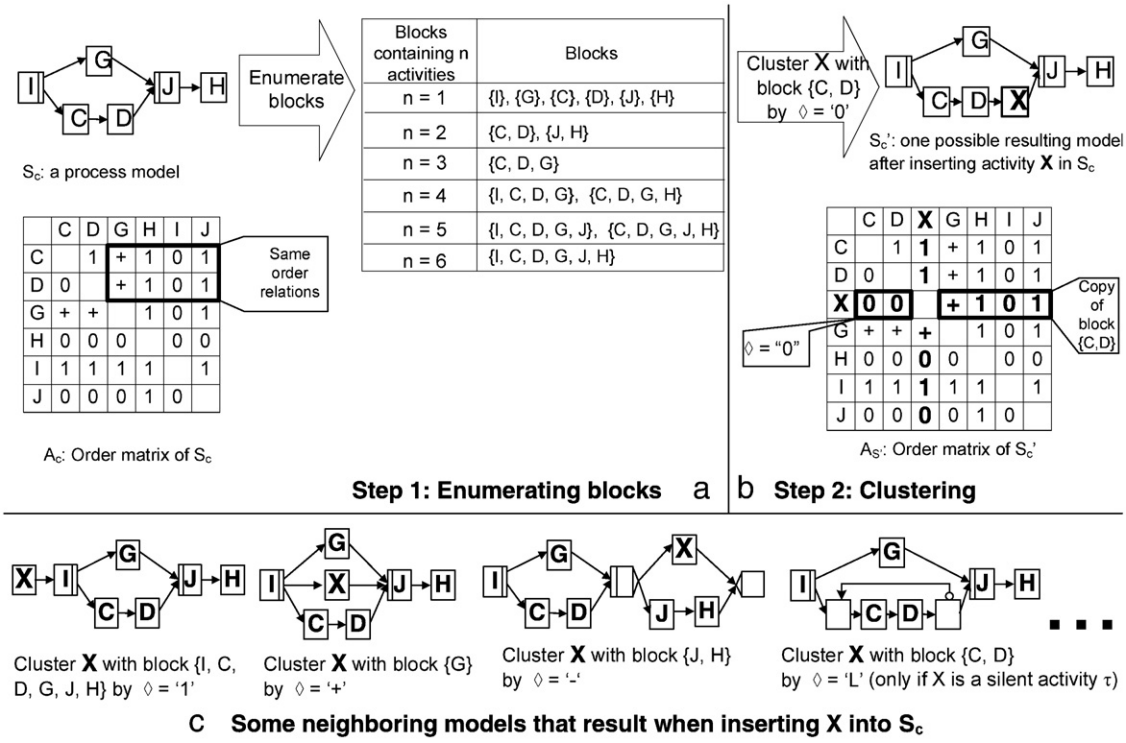


Fig. 7. Finding the neighboring models by inserting X into process model S.

two blocks; e.g., blocks {C,D} and {G} show same order relations in respect to H,I and J; therefore they can form a bigger block {C,D, G}; i.e., we can determine a block containing x activities by merging two disjoint blocks containing j and k activities respectively ($x=j+k$) (cf. Fig. 7). Based on this, we are able to enumerate all blocks with different sizes contained in a process model. Please see Appendix B for an algorithm formally describing the above steps.

Step 2 cluster inserted activity with one block. After having enumerated all possible blocks for a given candidate model S_c, we can insert activity a_j in S_c such that we obtain a sound and block-structured model again. Assume that we want to insert X in S (cf. Fig. 7). To ensure block structure of the resulting model, we "cluster" X with an enumerated block, i.e., we replace one of the previously determined blocks B by a bigger block B' containing both B and X. In this context, we set order relation between B and X to $\diamond \in \{0, 1, +, -\}$ or $\diamond = L$ if X is a silent activity τ representing a loop-block, i.e., the order relations between X and all activities of B are defined by $\diamond$. One example is given in Fig. 7b: added activity X is clustered with block {C,D} using order relation $\diamond =$ "0", i.e., X becomes a successor of the sequence block containing C and D. To realize this clustering, we have to set order relations between X on the one hand and block activities C and D on the other hand to "0". Further, order relations between X and remaining activities are the same as for C and D. Finally the three activities form a new block {C,D,X} replacing the old one (i.e., {C,D}). This way, we obtain a sound and block-structured process model S'.

Every time we cluster an activity with a block, we actually add this activity to the position where it can form a bigger block together with the selected one, i.e., we replace a self-contained block of a process model by a bigger one. Fig. 7b shows one resulting model S' we obtain when adding X to S_c. Obviously, S' is not the only neighboring model since we can insert X at different positions; i.e., we can cluster each block enumerated in Step 1 with X by any one of the four order relations $\diamond \in \{0, 1, +, -\}$, or by 'L' if X is a silent activity representing a loop-block. In our example from Fig. 7, S_c contains 14 blocks. Consequently, the number of models that may result when adding X in S_c is $14 \times 4 = 56$ (or $14 \times 1 = 14$ if X is a silent activity τ); i.e., we can obtain 56 (14) potential models. Fig. 7c shows some neighboring models of S_c. Note that the resulting models are not necessarily unique, i.e., it is possible that some of them are the same. However, this is not an important issue in our context since $Fit(S_c)$ can be quickly computed; i.e., some redundant information does not significantly decrease performance of our algorithm.

5.3. Search result for running example

Fig. 8 presents the search result we obtain when applying our heuristic algorithm to the example from Fig. 2. We do not set any limitation on the number of search steps in order to find the best reference model. Fig. 8 shows the evolution of the original reference model S. First operation $\Delta_1 = move(S, J, B, endFlow)$ changes S into intermediate model R_1, which shows highest fitness value in comparison to all other neighbor models of S. Using R_1 as next input for our algorithm, we discover R_2 by applying $\Delta_2 = delete(R_1, Loop)$, and then R_3 using $\Delta_3 = insert(R_2, X, E, B)$. Finally, we obtain R_4 by applying $\Delta_3 = move(R_2, I, D, H)$ to R_3. Since we cannot find a "better" process model by changing R_4 anymore, we obtain R_4 as final result. Note that if we set constraints on allowed search steps (i.e., we only allow to change original reference model S by maximum d change operations), the final search result will be as follows: R_d *if* $d \leq 4$ *or* R_3 *if* $d > 4$. Table 2 further compares S and all (intermediate) search results.

It is not surprising that fitness value increases with continuing search since we use fitness to guide it. However we need to examine whether the discovered process models are indeed getting better by computing their average weighted distance to the variants, which is a precise measurement in our context. From Table 2, iterative improvement of average weighted distances becomes clear, i.e., it drops monotonically from 4.85 to 2.4, which indicates that the algorithm performs as expected in the given example.

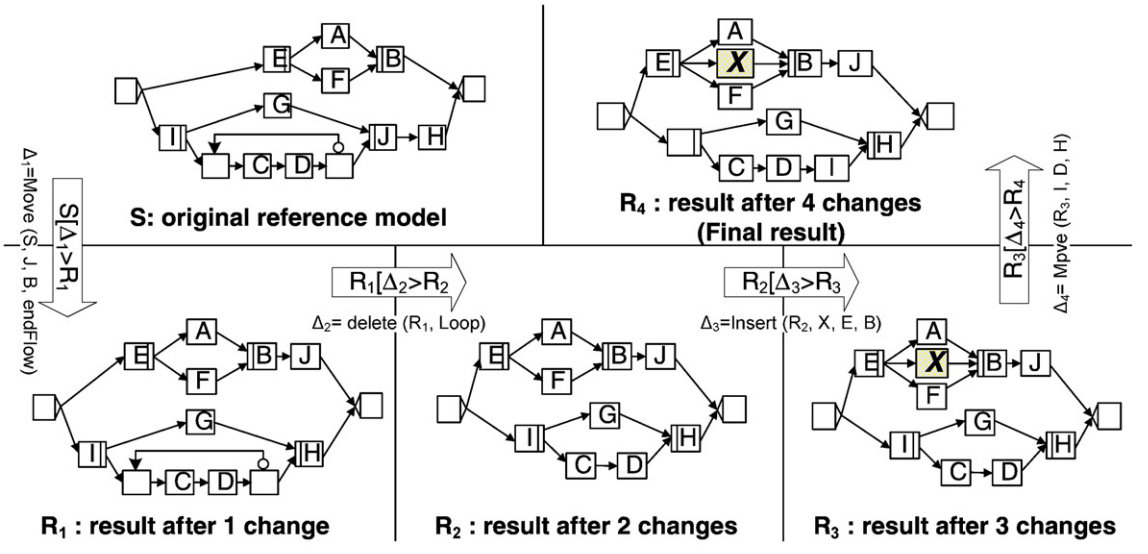


Fig. 8. Search result by every change operation.

Table 2
Search result by every change.

	S	R_1	R_2	R_3	R_4
Fitness	0.543	0.687	0.805	0.844	0.859
Average weighted distance	4.85	3.95	3.25	2.65	2.4
Change operation		Move	Delete	Insert	Move
Delta-fitness		0.143	0.118	0.039	0.009
Delta-distance		0.9	0.7	0.6	0.25

One design goal for our heuristic search algorithm was to be able to only consider most relevant change operations, i.e., most important changes (reducing average weighted distance between reference model and variants most) should be discovered at the beginning. Therefore, we additionally evaluate *delta-fitness* and *delta-distance*, which indicate relative improvement of fitness values and reduction of average weighted distance after each change; e.g., first operation Δ_1 changes S into R_1, which improves fitness value (delta-fitness) by 0.143 and reduces average weighted distance (delta-distance) by 0.9. Similarly, Δ_2 reduces average weighted distance by 0.7, Δ_3 by 0.6 and Δ_4 by 0.25. Obviously, delta-distance is monotonically decreasing with increasing number of change operations. This indicates that most important changes are performed at the beginning of the search, while less important ones are performed at the end.

Another important feature of our heuristic search is its ability to automatically decide which activities shall be included in the reference model; i.e., a predefined threshold or filtering of less relevant activities are not needed. In our example, X is automatically added, while the Loop block is automatically deleted. The only optimization we want to achieve is to reduce average weighted distance, i.e., change operations (insert, move, and delete) are automatically balanced based on their contribution to reduce average weighted distance.

5.4. Performance evaluation based on simulations

Using one example to measure performance of our heuristic mining algorithm is far from being enough. Since computing average weighted distance is at $\mathcal{NP}-hard$ level, fitness function is only an approximation of it. Therefore, we have to analyze *to what degree delta-fitness is correlated with delta-distance?* Furthermore, we are interested in to what degree important change operations are performed at the beginning. If biggest distance reduction can be achieved with the first changes, setting search limitations or filtering out the change operations performed at the end, does not constitute a problem. Therefore, we want to know: *To what degree important change operations are positioned at the beginning of our heuristic search?*

We try to answer these questions using *simulation*; i.e., by generating thousands of data samples, we can provide a statistical answer for them. In our simulation, we identify several parameters (e.g., size of the model and similarity of the variants) for which we investigate whether or not they influence performance of our heuristic mining algorithm (see [45] for details). For example, size of process variants ranged from 10 to 75 activities while their similarity to the reference process model ranged from 10% to 30%. In addition, 8 different scenarios concerning which activity to change and where the activities are changed are discussed as well. By adjusting these parameters, we generate 72 groups of datasets (7272 models in total) covering different scenarios. Each group contains a randomly generated *reference process model* and a collection of *100 different process variants*. We generate each variant by configuring the reference model according to a particular scenario. When performing our heuristic mining to discover new reference models, we do not set constraints on search steps, i.e., the algorithm only terminates if no better model can be discovered. *All (intermediate) process models* are documented (see Fig. 8 as example). We compute *fitness* and *average weighted distance* of each intermediate process models. We further compute *delta-fitness* and *delta-distance* in order to examine the influence of every change operation (see Table 2 for an example).

Correlation of delta-fitness and delta-distance. One important issue we want to investigate is how delta-fitness is correlated to delta-distance. Every change operation leads to a particular change of the process model, and consequently creates a delta-fitness x_i and delta-distance y_i. In total, we performed *284* changes in our simulation when discovering reference models. We use Pearson correlation to measure correlation between delta-fitness and delta-distance [46]. Let X be delta-fitness and Y be delta-distance. We obtain n data samples (x_i, y_i), $i = 1,\ldots,n$. Let $\bar{x}$ and $\bar{y}$ be the mean of X and Y, and let s_x and s_y be the standard deviation of X and Y. Pearson correlation r_{xy} then equals $r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y}$ [46]. Results are summarized in Table 3. All correlation coefficients are

Table 3
Correlation analysis.

	Correlation analysis				Correlation comparison		
	# of activity per variant	# of data	Correlation	Significant?	Pairwise Comparison	Probability being same	Significant?
Small-sized	10–15	33	0.762	Yes	Small vs. Medium	0.130	Yes
Medium-sized	20–30	74	0.589	Yes	Medium vs. Large	0.689	Yes
Large-sized	50–75	177	0.623	Yes	Small vs. Large	0.170	Yes

significant and *high* (>0.5). The high positive correlation between delta-fitness and delta-distance indicates that when finding a model with higher fitness value, we have very high chance to also reduce average weighted distance. We additionally compare these three correlations. Results indicate that they do not show significant difference from each other, i.e., they are statistically the same (see [45]). This implies that our algorithm provides search results of similar goodness *independent* from the number of activities contained in the process variants.

Importance of top changes. We analyze to what degree our algorithm applies more important changes at the beginning. For this purpose, we measure to what degree the top *n%* changes reduced average weighted distance. As example consider search results from Table 2. We performed in total 4 change operations and reduced average weighted distance by 2.45 from 4.85 (based on S) to 2.4 (based on R_4). Among the four change operations, the first one reduced average weighted distance by 0.9. When compared to overall distance reduction of 2.45, the top 25% changes accomplished 0.9/2.45 = 36.73% of overall distance reduction. This number indicates how important changes at the beginning are. We therefore evaluate distance reduction by analyzing the top 33.3% and 50.0% change operations. On average, the top 33.3% change operations contribute to 63.80% distance reduction while the top 50.0% have achieved 78.93%. Consequently, the changes at the beginning are *a lot more important* than the ones performed later.

6. Scenario 2: discovering reference process model by mining process model variants: a clustering approach

We now present a clustering algorithm for mining a collection of process variants without knowledge about the original reference model. 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 continues until all activities and blocks respectively are merged into one single block. This block and its internal structure represent the newly discovered reference process model.

Based on the aggregated order matrix representing a collection of variants, our clustering approach for mining process variants works as follows:

1. Determine the activity set to be considered in the new reference process model.
2. Determine the activities (blocks) to be clustered in a block.
3. Determine the order relation the clustered activities (blocks) shall have within this block.
4. After having built a new block in Steps 2 and 3, adjust the aggregated order matrix accordingly.
5. Repeat Steps 2–4 until all activities are clustered; i.e., until the new process model is constructed by enlarging blocks.

6.1. Determining the activity set of the reference process model

One fundamental challenge is to decide which activities shall be considered in the new reference model. As basis for this decision we choose *activity frequency* (cf. Def. 9). The user may set a threshold such that only activities with activity frequency higher than this threshold are considered in the reference process model. This way we can exclude activities with low frequency if we only want to consider activities with frequency greater than 60% in our example, for instance, activities Y and Z as well as silent activity τ will be excluded from the reference process model (excluding τ means the loop structure will not be considered). Generally, process engineers have to set a threshold depending on whether they want to add more or fewer activities to the reference process model. In the following, we use 60% as threshold.

6.2. Determining activities to be clustered

Taking an order matrix (cf. Def. 6), two activities can form a block if they have the same order relations with respect to the remaining activities (cf. Section 5.2). We can apply similar idea when analyzing an aggregated order matrix. However, the relationship between two activities in an aggregated order matrix is expressed as 5-dimensional vector showing the distribution of the order relations over all process variants. When determining pairs of activities that can be clustered as a block, it would be too restrictive to require precise matching as in the case of an order matrix. To deal with this, we re-apply function $f(\alpha,\beta)$ (cf. Eq. 6) which expresses closeness between two vectors $\alpha=(x_1,x_2,\ldots,x_n)$ and $\beta=(y_1,y_2,\ldots,y_n)$. Using $f(\alpha,\beta)$ we introduce *Separation* metrics. It indicates to what degree two activities of an aggregated order matrix are suited for being clustered to a block. More precisely, $Separation(a,b)$ expresses how similar order relations of activities a and b are when compared to the other activities. In our example from Fig. 2, $Separation(\mathrm{A},\mathrm{B})$ is determined by the closeness (measured in terms of the cosine value) of $f(v_{AC},v_{BC})$, $f(v_{AD},v_{BD})$, …, $f(v_{AJ},v_{BJ})$, and $f(v_{AX},v_{BX})$. We define cluster separation as follows:

$$Separation(a,b) = \frac{\sum_{x\in N\setminus\{a,b\}} f^2(v_{ax},v_{bx})}{|N|-2} \in [0,1] \tag{8}$$

N corresponds to the set of activities. Like most clustering algorithms [43], we square the cosine value to emphasize the differences between the two compared vectors. Finally, dividing this expression by $|N|-2$ normalizes its value to a range between [0, 1]. The higher $Separation(a,b)$ is, the better activities a and b are separable from others, and the more probable a and b should. For our example from Fig. 1 we obtain $Separation(\mathrm{A},\mathrm{B})=0.776$. We determine the pair of activities best suited to form a block by

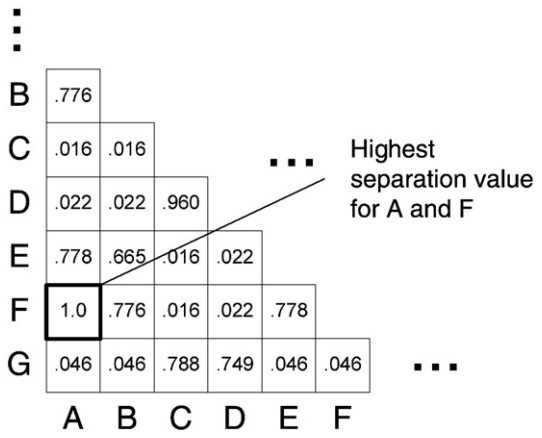


Fig. 9. Separation table.

computing separation value for each activity pair. Fig. 9 depicts separation values for our running example from Fig. 2. We denote this table as *separation table*. Obviously, A and F have the highest separation value of 1.0. We therefore choose A and F to form our first block. Since *separation*(A, F) = 1 holds, A and F can form a block in all six variants, i.e., we obtain the same results when directly analyzing the variants (cf. Fig. 2).

6.3. Determining internal order relations

After clustering A and F in the first block, we need to determine the order relation between these activities. For this purpose, we introduce *Cohesion* to measure how significant particular order relations between two activities of the same cluster are. In the aggregated order matrix of our example, the relationship between activities A and F is depicted as 5-dimensional vector $v_{AF} = (0,0,1,0,0)$. It shows distribution values of the five types of order relations. When building a reference process model, only one of the five order relations can be chosen. Therefore, we want to choose the most significant order relation. Regarding our example, significance of each order relation can be evaluated by the closeness vector v_{AF} and having the five axes in the 5-dimensional space. These axes can be represented by five benchmarking vectors: $v^0 = (1,0,0,0,0)$, $v^1 = (0,1,0,0,0)$, $v^+ = (0,0,1,0,0)$, $v^- = (0,0,0,1,0)$, and $v^L = (0,0,0,0,1)$. We can compute significance of each order relation using $f([\alpha,\beta])$ (cf. Eq. 6). In our example, the closest axis to v_{AF} is v^+ (with $f(v_{AF}, v^+) = 1$). Therefore, we decide that A and F shall be organized in parallel within the newly derived block (cf. Def. 6). We use Cohesion to evaluate how good our choice is:

$$Cohesion(a,b) = \frac{max_{\diamond \in \{0,1,+,-,L\}} \left\{ f\left(v_{ab}, v^{\diamond}\right) \right\} - 0.4472}{1 - 0.4472} \quad \in [0,1] \tag{9}$$

Cohesion(a,b) equals *one* if there is a dominant order relation, i.e., v_{ab} is on one of the five axes; or equals *zero* if $v_{ab} = (0.2, 0.2, 0.2, 0.2, 0.2)$ holds (i.e., no order relation is more significant than the others). In our example, *cohesion*(A, F) = 1 holds; this indicates that A and F have order relation '+' in all six process variants; we can obtain the same results by directly analyzing the variants (cf. Fig. 2).

6.4. Recomputing the aggregated order matrix

We have discovered the first block of our reference process model which contains A and F having order relation '+'. We now have to decide on the relationship between the newly created block and the remaining activities. We accomplish this by adapting the aggregated order matrix.[7] For this purpose, we compute the means of the order relations between {A, F} and remaining activities; e.g., since $v_{AI} = (0, 0.15, 0, 0.85, 0)$ and $v_{FI} = (0, 0.15, 0, 0.85, 0)$, the order relation between new block {A,F} and activity I correspond to $(v_{AI} + v_{FI})/2 = (0, 0.15, 0, 0.85, 0)$.[8] Such computation is applied to all remaining activities outside this block. Generally, after clustering activities a and b, aggregated order matrix V' can be re-calculated as follows:

$$\forall x \in N \setminus \{a,b\} : \begin{cases} v'_{(a,b)x} = (v_{ax} + v_{bx})/2 \\ v'_{x(a,b)} = (v_{xa} + v_{xb})/2 \end{cases} \tag{10}$$

$$\forall x,y \in N \setminus \{a,b\} : v'_{xy} = v_{xy} \tag{11}$$

[7] Our approach is different from traditional clustering algorithms [43], which only re-compute distances, but not the original dataset.

[8] This approach is an unweighted one; i.e., we simply take the average of the two vectors without considering their importance (e.g., how many activities are included in the block). This way, we can ensure that when merging two blocks of different sizes, the order relations of the resulting block are not too much dominated by the bigger one. Such unweighted approach is widely used in other clustering approaches [43].

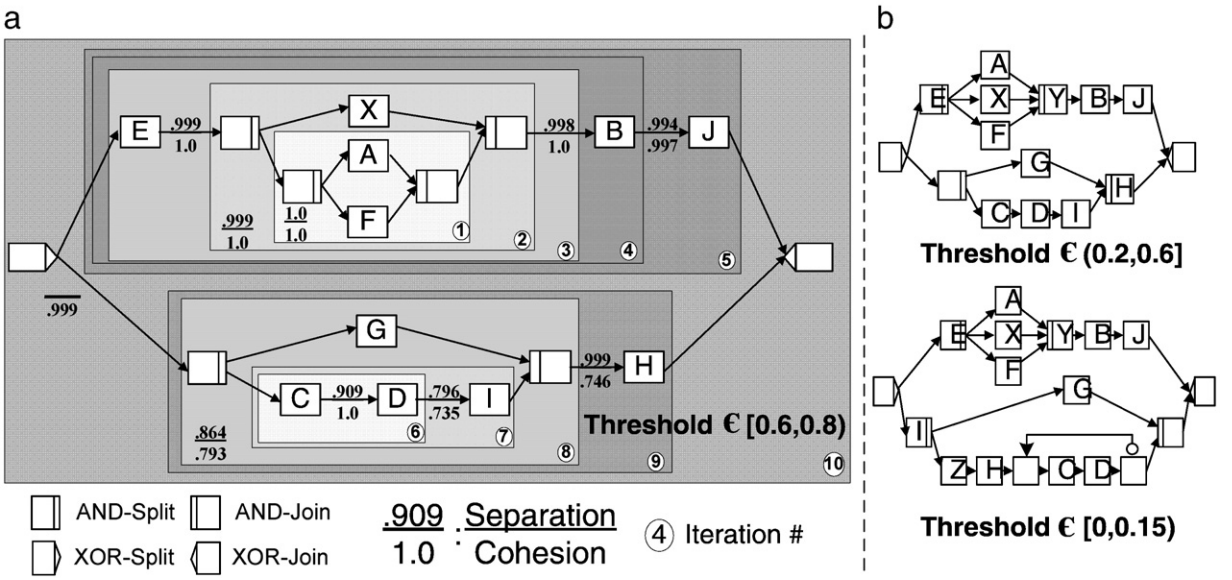


Fig. 10. Reference process model discovered by clustering algorithm.

Since A and F are replaced by one block, the matrix resulting from this re-computation is one dimension smaller than *V*. Afterwards, we treat this block like a single activity, but keep its internal structure in order to build up the new reference process model at the end.

6.5. Applying the clustering algorithm to our example

We re-apply the steps described in Sections 6.2–6.4 until all activities and blocks respectively are clustered together. Fig. 10a shows the reference process model we can discover for our example. It further depicts the blocks as constructed in each iteration as well as the two evaluation measures *Separation* and *Cohesion*. Using separation and cohesion, we can evaluate how each part of the reference process model fits to the variants. For example, it is clear that activities A and F can always form a block in the six variants (high separation) and the order relation between A and F is also very consistent (high cohesion). By contrast, activity I does not often succeed block {C,D} (low separation and cohesion). We can draw similar results if we have another look at the process variants (cf. Fig. 2). Fig. 10b further shows two other reference process models we obtain when setting other threshold values for determining the activity set (cf. Section 6.1).

6.6. Proof-of-concept prototype

We implemented and tested both the heuristic and the clustering algorithm using Java. Fig. 11 depicts a screenshot of our prototype. We use the ADEPT2 Process Template Editor [34] as tool for creating process variants. For each process model, the editor can generate an XML representation with all relevant information being marked up. We store created variants in a repository which can be accessed by our mining procedure. The mining algorithms were developed as stand-alone Java program, independent from the process editor. This program can read the original reference model (if available) as well as all process variants. It then generates the result models and stores them as accessible XML schemas. All intermediate search results are also stored.

7. Algorithm comparisons

We now compare our heuristic algorithm (cf. Section 5) with our clustering approach (cf. Section 6). In Section 7.2, we then compare the two algorithms with process mining techniques [16], i.e., algorithms that discover process models from execution logs.

7.1. Comparing the two algorithms for process variant mining

7.1.1. Qualitative comparison

Inputs and Goals. Fig. 12 illustrates how our heuristic mining algorithm differs from the clustering one in respect to goals and inputs. It represents each process variant S_i as single (white) node in the two dimensional space. Our heuristic algorithm tries to discover a new reference process model by performing a sequence of change operations on the original one. In particular, it

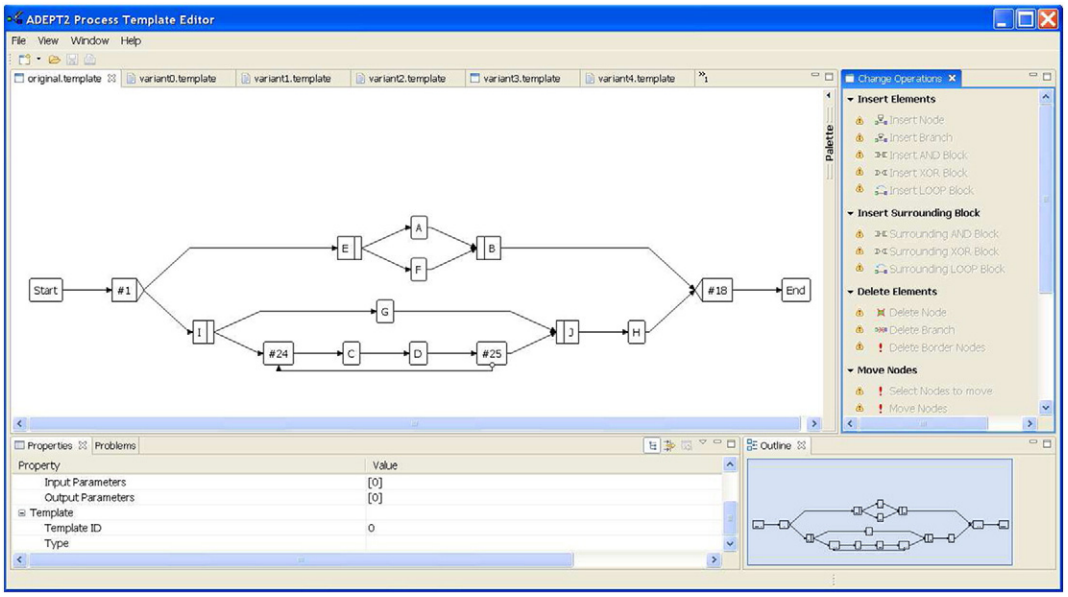


Fig. 11. Screenshot of the prototype.

balances two "forces": one is to bring the new reference model S_c closer to the variants (i.e., to the bull's eye S_{nc} at the right); the other force is to not "move" it too far away from original reference model S, i.e., S_c should not differ too much from S. Our heuristic algorithm provides such flexibility by allowing process engineers to set a maximum search distance. Our simulations (cf. Section 5.4) showed that the change operations that are applied first to the (original) reference model are more important than the ones positioned at the end i.e., they reduce distance between reference model and variants more. Consequently, when ignoring less relevant changes, we do not influence overall distance reduction too much. While the above scenario presumes knowledge of the original reference model, we also must cope with cases in which there exists only a collection of process variants, but no original reference process model is known. The goal of our clustering approach therefore purely is to discover the "center" of the variants, i.e., a reference process model with shortest average weighted distance to them. In particular, no knowledge about the original reference model is required. In principle, it is also possible to apply our heuristic algorithm in this scenario. We just need to start with an "empty" model S and do not set any search limitation. However, since we do not need to balance the two forces and to perform the important change operations at the beginning of the search, the clustering algorithm is expected to be faster or can provide additional information on the search result. We discuss this in Section 7.1.2.

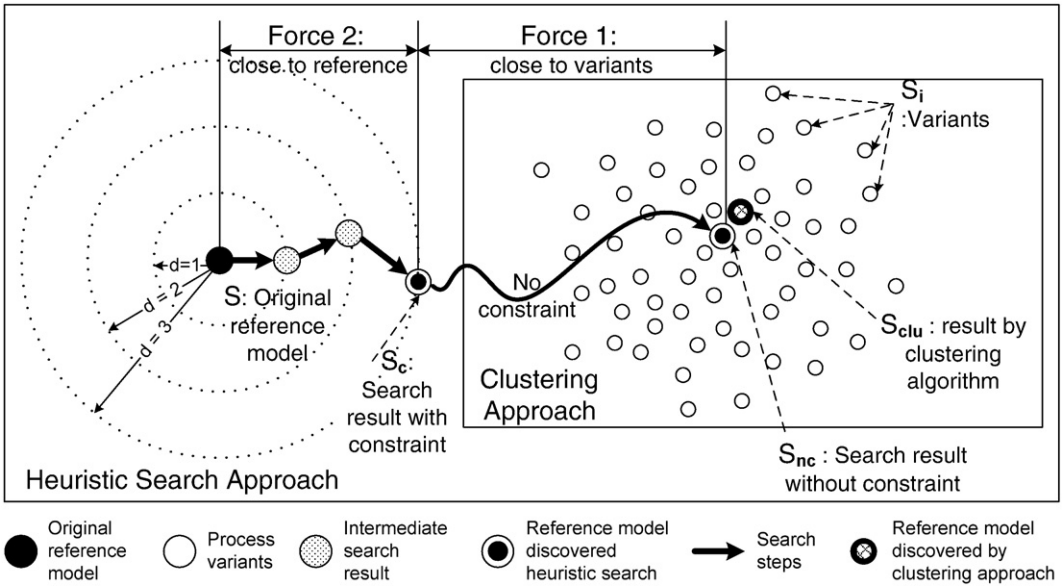


Fig. 12. High-level overview of the two algorithms.

Table 4
Qualitative comparison between clustering algorithm and heuristic algorithm.

	Clustering algorithm	Heuristic algorithm
Input	Collection of process variants.	Collection of process variants + original reference process model
Goal	Discover reference process model with shortest average weighted distance to the variants	Discover better reference process model within certain distance of the original one
Design principle	Local view: Discover reference process model by enlarging blocks	Global view: Discover reference process model by searching better candidate models
Complexity	$\mathcal{O}(n^2m + n^3)$ (n: # of activities; m: # of variants)	$\mathcal{NP}-hard$ (in worst case scenario)
Pros & Cons	1. Runs very fast 2. Provides local view on how each part of the reference process model fits to the variants 3. Activity set can be flexibly chosen by user	1. Can automatically decide on activity set 2. Can control the distance between the discovered and original reference process models 3. Applies more important change operations at the beginning

Design principles and complexity. Our heuristic algorithm discovers a better reference model by applying a sequence of change operations to the original one. To enable quick decisions on a large search space (cf. Section 5.2), we use a fitness function to evaluate how well a candidate model fits to the variants. This fitness function only provides a global evaluation, but does not show how each part of the candidate model fits to the variants. On the contrary, the clustering algorithm discovers a reference process model by enlarging blocks. By evaluating separation and cohesion, we are able to determine how well each part of the discovered reference model fits to the variants; i.e., due to its different design the clustering algorithm returns more information than the heuristic one. Complexity of the two algorithms differs as well. Despite polynomial complexity of computing the fitness of a candidate model, worst case, enumerating all blocks in a candidate model has $\mathcal{NP}-hard$ complexity.[9] On the contrary, our clustering algorithm has polynomial complexity since computing separation and cohesion is both polynomial. To be more precise, if n is the number of activities and m the number of variants, complexity of the clustering algorithm is $\mathcal{O}(n^2m + n^3)$. This implies that the clustering algorithm can quickly compute the reference process model of a large collection of process variants, while the heuristic algorithm may take considerably longer.

Pros and cons. The differences between the two algorithms are summarized in Table 4. Additional attention should be paid to the pros and cons of the two algorithms. Since the clustering algorithm has polynomial complexity, it runs significantly faster than the heuristic algorithm. Using Separation and Cohesion, we obtain information about how each part of the discovered reference process model fits to the variants. However, our clustering algorithm cannot control the discovery procedure or distinguish important changes from less relevant ones as our heuristic algorithm does. Though for our running example, our clustering algorithm discovered same process model (cf. Fig. 10) as our heuristic algorithm (cf. Fig. 8). In many other cases the discovered model was less optimal for the clustering algorithm since its search space is considerably smaller.

7.1.2. Quantitative comparison

We now compare our two algorithms quantitatively by analyzing how fast they run and how good discovered models are. We use the same data for this comparison as for the evaluation of our heuristic algorithm (cf Section 5.4). We generated 72 groups of datasets representing different scenarios. Each group contains 1 reference process model and 100 process variants. Based on this, with each algorithm we discovered a new reference process model and documented execution time and distance reduction (between discovered model and the original one). Results are summarized in Table 5, which indicates that the clustering algorithm runs significantly faster than the heuristic one. However, results obtained with the clustering algorithm are less optimal compared to the heuristic algorithm.

7.2. Comparison with existing process mining algorithms

Process mining has been extensively studied in literature. Its key idea is to discover a process model by analyzing *execution behavior* of process instances as captured in execution logs [16]. The latter typically documents the start/end of each activity execution, and therefore reflects behavior of implemented processes. In principle, process mining techniques [15–18] can be applied in our context as well. Consider our example from Fig. 4. For each process variant S_i, we could first obtain its trace set $\mathcal{T}_{S_i}$ by enumerating all traces producible by S_i [47]. If a process model contains loop structures (i.e., it can generate infinite number of traces), without loss of generality, we assume that a loop-block is executed either once or twice. Despite this simplification, however, the number of traces producible by a process model can be extremely large; e.g., if a parallel branching contains five branches, of which each contains five activities, the number of producible traces is $(5\times5)!/(5!)^5 = 623360743125120$. This explains why we conduct the comparison only in small scale.

The trace sets generated for the variants are merged into one trace set $\mathcal{T}$ taking the weight of each variant into account. As S_1 accounts for 25% of the variants, for example, we ensure that each trace producible by S_1 has the same number of instances and the

[9] Worst-case, complexity of this algorithm is 2^n where n corresponds to the number of activities. This worst-case scenario will only occur if any combination of activities may form a block (like a process model for which all activities are ordered in parallel to each other). During our simulation, in most cases we were able to enumerate all blocks of a process model within milliseconds. This indicates low complexity in practice.

Table 5
Performance comparison between clustering algorithm and heuristic algorithm.

	Average execution time			Average distance reduction	
	# of activity per variant	Clustering Algorithm	Heuristic Algorithm	Clustering Algorithm	Heuristic Algorithm
Small-sized	10–15	0.013	0.184	6.93%	19.73%
Medium-sized	20–30	0.022	4.568	11.14%	22.59%
Large-sized	50–75	0.181	805.539	−8.97%	11.70%

sum of all instances producible by S_1 accounts for 25% of the instances in $\mathcal{T}$ as well. We consider $\mathcal{T}$ as execution log since it fully covers behavior of the given variant collection.

Since all activities captured in execution log will be included in the discovered process model by process mining algorithms (same as our clustering algorithm), we introduce two additional datasets. In the first one, we filter out all activities a_j whose activity frequency $g(a_j)$ is lower than 0.2 in the variant collection (cf. Def. 9); i.e., in our example activity Z in S_5 as well as silent activity τ (representing the loop in S_4) are ignored. For this extended data set, we can determine trace set $\mathcal{T}_{0.2}$. In the second additional dataset, we filter out activities whose activity frequency is lower than 0.6. Regarding our example, besides Z and τ, we then additionally filter out activity Y in S_2, S_3, S_5, and S_6. Consequently, we obtain trace set $\mathcal{T}_{0.6}$ which contains all traces producible by the reduced variants. Note that $\mathcal{T}_{0.6}$ has the same activity set as the model discovered by our heuristic algorithm (cf. R_4 in Fig. 5). The enumerated trace sets $\mathcal{T}$, $\mathcal{T}_{0.2}$ and $\mathcal{T}_{0.6}$ are imported into the ProM framework, which is one of the most popular tools for process mining and process analysis [48]. In our comparison, we consider alpha algorithm [16], heuristic miner [17], genetic algorithm [15] and multi-phase miner [18]. These are well-known algorithms for discovering process models from execution logs.[10]

7.2.1. Evaluation criteria

Our algorithms focus on the *structural* perspective of process models, i.e., our goal is to configure the variant models out of a reference model with minimal efforts (i.e., with minimal number of high-level changes). On the contrary, traditional process mining focuses on process *behavior*, i.e., the discovered process model should cover the behavior of the variant models (shown by their trace sets) [15–18]. In the following, we compare our algorithms with existing process mining algorithms from both structural and behavior perspectives.

Since most existing process mining algorithms are based either on Petri Nets or EPCs, to enable comparison, we transform the process models discovered by the different algorithms either into Activity Nets, Petri Nets or EPCs (see [47,49] for respective transformation techniques). Particularly, such model transformation enables us to apply existing metrics [14,23] and tools [48] for process model evaluation instead of introducing new ones. We only briefly describe the metrices applied in this paper and refer [14,23,48] for details. In the following we first introduce three parameters to evaluate the structure of process models, namely *average weighted distance, structural appropriateness* and *# of splits/joins in EPC*.

1. *Average weighted distance* measures the efforts to configure the process variants out of the discovered reference process model. The lower this parameter is the easier the variants can be configured.
2. *Structural appropriateness* measures the complexity of a Petri Net by computing the ratio between labeled transitions and nodes (transitions and places) [14]. The value range of this parameter is [0,1]; the higher it is, the simpler a Petri Net is.
3. *# of Splits/Joins in EPC* measures the number of splits and joins contained in an Event Process Chain (EPC). It can be used to measure the complexity of an EPC [23]. The higher this parameter is, the more choices end users need to make when executing the process model and the more complex the respective EPC is.

We additionally use three parameters to evaluate the behavior of discovered process models, namely *behavior fitness, successful execution* and *proper completion*.

1. *Behavior fitness* evaluates whether the discovered process model (represented as Petri Net) complies with the behaviors captured in the execution log [14]. One way to investigate behavior fitness is to replay the log on the Petri net. This is done in a non-blocking way, i.e., if there are tokens missing to fire a transition in the discovered model, they are artificially created and replay proceeds [14]. The value range of this parameter is [0,1] and the higher behavior fitness is, the better the discovered model covers the trace set.
2. *Successful execution* measures percentage of traces in an execution log that can be successfully executed by the discovered process model [14]. The value range of this parameter is [0,1]; the higher it is, the more traces in the execution log can be reproduced based on the discovered model.
3. *Proper completion* measures percentage of traces in an execution log that leads to proper completion [14]. Compared to "successful execution" this parameter further requires that the analyzed process model reaches an end state after replaying a trace. The value range of this parameter is [0,1]; the higher it is, the more traces from the execution log result in proper completion.

[10] The enumerated trace sets $\mathcal{T}$, $\mathcal{T}_{0.2}$ and $\mathcal{T}_{0.6}$, as well as the process models discovered by different algorithms based on them are available at *http://wwwhome.cs.utwente.nl/lic/Resources.html*.

Table 6
Performance comparison with process mining algorithms.

Dataset	Algorithms	Structure measurement			Behavior measurement		
		Average weighted distance	Structural appropriateness	# of Joins/splits in EPC	Behavior fitness	Successful execution	Proper completion
$\mathcal{T}$	Heuristic Var.	2.4	0.481	6	0.876	0.353	0.353
	Clustering	4.75	0.468	8	0.737	0.120	0.120
	Alpha	8.55	0.441	15	0.646	0	0
	Heuristic	8.85	0.258	31	0.437	0.042	0
	Genetic	6.6	0.341	19	0.811	0.342	0.009
	Multi-phase	2245 arcs and 515 transitions		19	In theory, all equals 1		
$\mathcal{T}_{0.2}$	Heuristic Var.	2.4	0.481	6	0.886	0.382	0.382
	Clustering	2.6	0.482	6	0.784	0.133	0.133
	Alpha	6.9	0.466	12	0.706	0	0
	Heuristic	8.2	0.274	12	0.789	0.268	0
	Genetic	5.9	0.424	13	0.846	0.460	0.009
	Multi-phase	1534 arcs and 384 transitions		18	In theory, all equals 1		
$\mathcal{T}_{0.6}$	Heuristic Var. Clustering	2.4	0.481	6	0.851	0.327	0.337
	Alpha	6.85	0.5	7	0.814	0.407	0
	Heuristic	7.85	0.462	10	0.736	0.407	0
	Genetic	3.2	0.325	13	0.886	0.394	0.278
	Multi-phase	1266 arcs and 302 transitions		17	In theory, all equals 1		

7.2.2. Evaluation results

Our evaluation results are summarized in Table 6. To differentiate our heuristic algorithm from heuristic miner known from process mining [17], we denote our heuristic algorithm as "Heuristic var." in Table 6. It is not surprising that both our heuristic and our clustering algorithm can discover a process model of simple structure. Independent from which dataset we use, the process models discovered by our two algorithms have better scores for all three parameters relating to the structure of the discovered model; i.e., they have lower average weighted distance, higher structural appropriateness, and less number of splits/joins in the corresponding EPC model. Except multi-phase miner [18], none of the algorithms discovered a process model with behavior fitness being 1. Note that multi-phase miner was designed in a way that it always discovers a process model with fitness 1. Despite the fact that the models discovered by multi-phase miner are extremely complex, they also allow for more behavior not covered by the variants [14,18]; i.e., results are often overfitting.[11] Consequently, we consider that the cost of multi-phase miner for reaching behavior fitness 1 as too high. When excluding multi-phase miner, evaluation results show that even if we apply traditional process mining algorithms for discovering a process model that covers behavior of the variants best, the resulting model might NOT be able to support all behaviors captured by the variants. This indicates the necessity for process configurations: i.e., it is not sufficient to maintain only one model which covers all behaviors; instead we must enable process configurations at both run-time and build-time to obtain different process variants which support specific behaviors in different scenarios.

Surprisingly, behavior measurements of the process model discovered by our heuristic algorithm are also very good. Note that our heuristic algorithm discovers the same model (cf. R_4 in Fig. 5) for $\mathcal{T}$, $\mathcal{T}_{0.2}$ and $\mathcal{T}_{0.6}$. This model has the highest behavior fitness for trace sets $\mathcal{T}$ and $\mathcal{T}_{0.2}$, and only a few percent less than the genetic algorithm for $\mathcal{T}_{0.6}$. This is rather unexpected because our heuristic variant mining algorithm is focusing on structure rather than on behavior. Though our algorithm focuses on the discovery of a reference model out of which the process variants can be easily configured, this implies that behavior of the discovered model has not been sacrificed that much. Since structure fitness is not 1, however, we should apply process configurations to obtain suited process variants supporting the execution of different process instances best.

8. Automotive case study

Context. We conducted a case study in a large automotive company in which we analyzed variants of its product change management process. Basically, this process comprises several phases like specification of a change request, handling of this change request, change implementation, and roll-out. In the following we only consider the top-level process and comment on sub-processes later on. Usually, the change management process starts with the initiation of a Change Request (CR), which must then be detailed and assessed by different teams (e.g., from engineering and production planning). The gathered comments then

[11] In principle, it is possible to measure overfitting using parameter like *behavioral appropriateness* [14], however, due to the complexity of the discovered models, the conformance checker in Prom cannot measure some of models (besides Multi-Phase miner) in a reasonable time (e.g., within a couple of days), therefore we did not include it in our comparison.

have to be aggregated and approved by the CR board. In case of positive approval change implementation may start (e.g., detailing the planning and triggering the re-engineering of parts affected by the change).

Data source. We identified 14 process variants dealing with (product) change management. These variants were captured in separate process models being expressed in terms of UML Activity Diagrams and using standard process patterns like Sequence, AND-/XOR-Splits, AND-/XOR-Joins, and Loop. Size of the variants ranged from 5 to 12 activities and their weights from 2 to 15 according to the relative frequency of corresponding process instances. However, none of the process variants was dominant or significantly more relevant than others. All variant models were already block-structured or could be transformed into a behavior-equivalent block-structured process model.

Sources of variance. Though the variant models show structural similarities they comprise parts which are only relevant for a sub-collection of the variants. For critical changes, for example, the Quality Assurance Department needs to be involved in the appraisal and commenting of the change request, while this is not required for normal changes. Concerning low-cost changes, in turn, change implementation may start before the change request is finally approved. In this case, the implementation procedure will have to be aborted and compensated if the approval is withheld. Other points of variations concern the preparation of the approval task, the communication of implemented changes, and the triggering of secondary changes (raised by the requested one). The left of Fig. 13 exemplarily shows 4 variant models from our case.

Case study results. Since we did not know the original reference process model, this case corresponds to Scenario 2. Therefore, we first applied our clustering algorithm to "merge" the process variants. This way we obtained S' (cf. Fig. 13) as reference process model. As average weighted distance between S' and the variants we obtained 2.06. The time to find the model was negligible (0.031 s). We also applied our heuristic mining algorithm to the given case. Since there was no original reference process model, we used the most frequent variant (cf. S_1 in Fig. 13) as starting point of our search. We did not set any search limitations in this context such that our heuristic algorithm could discover the best model. As result we obtained S' (cf. Fig. 13) again as best reference process model (after performing one change on S_1). Though the heuristic algorithm ran longer than the clustering one to find the reference process model, overall search time was only 1.062 s. We discussed the discovered reference model with process engineers from the automotive company. They confirmed that it constitutes a good choice for representing the top level change management process. When further applying our mining algorithms to sub-processes relating to the different phases of the change management process (e.g., change implementation) and their variants we obtained good results as well.

The practical relevance and benefit of our variant mining algorithms further became evident in the context of another case study we conducted in a clinical center. Here we analyzed more than 90 process variants for handling medical orders and medical procedures respectively (e.g., X-ray inspections, cardiological examinations, and lab tests). By applying our algorithms to these 90 variants we obtained reference models that were closer to the variants than the old reference model.

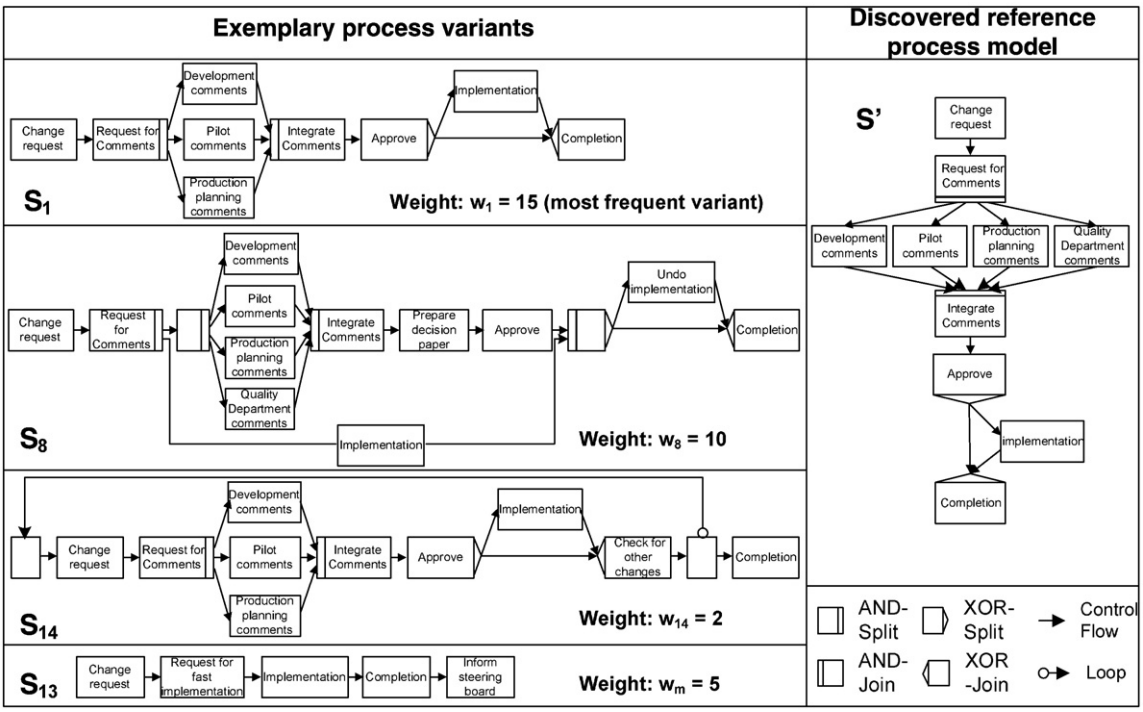


Fig. 13. Example process variants in the change management case study.

9. Related work

Though heuristic search algorithms and clustering algorithms are widely used in data mining [43], artificial intelligence [42], and machine learning [44], only few approaches apply heuristics or clustering for process variant management. In particular, only few solutions exist for learning from the adaptations that were applied when configuring a collection of process variants out of a process model.

Structural process changes during runtime and approaches for flexible process configuration are intensively discussed in literature [4,11]. A comprehensive analysis of theoretical and practical issues related to (dynamic) process changes is provided in the context of the ADEPT2 change framework [8]. Furthermore, there exist approaches for dynamic structural changes of Petri nets [20]. Based on such conceptual frameworks, the AristaFlow BPM suite [34] and tools for configurable process models [50] emerged. Further, there exist approaches which support management and retrieval of separately modeled process variants. As example, [51,52] allows storing, managing, and querying large collections of process variants within a repository. Graph-based search techniques are used for retrieving variants that are similar to a user-defined process fragment. Obviously, this requires profound knowledge about the structure of stored processes. Apart from this, no techniques for analyzing the different variants and for learning from their specific customizations are provided.

ProCycle enables change reuse at the process instance level to effectively deal with recurrent problem situations [11]. ProCycle applies case-based reasoning techniques to allow for semantic annotation as well as retrieval of process changes. Respective process adaptations can then be re-applied in a similar problem context to configure other process instances later on. If the reuse of a particular change exceeds a certain threshold, it becomes a candidate for adapting the process schema at type level. Though the basic goal of ProCycle is similar to our approach, its techniques are simpler and do not consider change variation.

There are few techniques which foster learning from process variants by mining recorded change primitives (e.g., to add or delete control edges). [53] measures process model similarity based on change primitives and suggests mining techniques using this measure. Similar techniques exist in the field of association rule mining [43] or frequent sub-graph mining [54] as known from graph theory [55]; here common edges between different nodes are discovered to construct a common sub-graph from a set of graphs. However, this approach does not consider important features of process meta model; e.g., it is unable to deal with silent activities, cannot differentiate between AND- and XOR-branchings or Loops.

To mine high level change operations, [13] presents an approach based on process mining techniques, i.e., the input consists of a change log, and process mining algorithms are applied to discover the execution sequences of the changes (i.e., the change meta process). However, this approach simply considers each change as individual operation such that the result is more like a visualization of changes rather than mining them. [56] shows a technique to rank activities based on their potential involvement in process configurations. In Configurable Workflow Models [50], all process variants are combined together into one reference process model based on inheritance rules known from Petri Nets [20]. Though questionnaire-based approaches can ease process configuration [57], the resulting model turns out to be complex and contains many decision points [58]. This approach becomes even more difficult when being confronted with a large collection of process variants, not being equally important. Here, an extremely large process model may result which contains too many decision points and cannot differentiate between important and trivial variants [59].

10. Summary and outlook

We presented challenges, scenarios and algorithms in respect to the mining of collections of process variants. In particular, we introduced, evaluated and compared two different algorithms for discovering a reference process model out of a collection of (block-structured) process variants. Adopting the discovered model as new reference process model makes (future) process configuration easier, since less efforts for configuring the variants are required. Our heuristic algorithm can take the original reference model into account such that the user can control to what degree the discovered model may differ from the original one. This way, we cannot only avoid spaghetti-like process models but also control how much changes we want to perform. Through a simulation of several thousand process models, we found out that the heuristic algorithm also applies important changes at the beginning of the search and its performance is able to scale up. The clustering algorithm does not presume knowledge of original reference process model based on which process variants are configured. By only looking at the variant collection, it can quickly discover a reference process model in polynomial time and provide additional information on how well each part of the discovered reference model fits to the variants. We successfully applied the suggested algorithms in a case study in the automotive domain. We further compared our algorithms with existing process mining algorithms. Results indicate good performance of our algorithms in both structure and behavior aspect. However, it would be useful to integrate our algorithms with existing process mining algorithms such that it can take both structure and behavior perspective into account in order to cover more general cases [16]. As learned from our case study, data-flow also constitutes an important part of process configurations. Therefore, it would be advantageous to additionally consider the data-flow perspective.

Acknowledgment

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

Appendix A. Heuristic search algorithm for variant mining

```
input  : A process model S; a collection of process variants S_i = (N_i, E_i, ...), i = 1, ..., n; allowed search
         distance d ;
output : Resulting process model S'
AS = ⋃_{i=1}^{n} N_i                                   /* Define AS as active activity set */;
S_c = S                                                /* Define initial candidate model */;
t = 1                                                  /* Define initial search step */ ;
while |AS| > 0 and t ≤ d do                            /* Search condition */;
    S_sib = S_c                                        /* Set S_c as initial S_sib */ ;
    Define a_s as the selected activity ;
    foreach a_j ∈ AS do
        S_kid = FindBestKid(S_c) ;
        if Fitness(S_kid) > Fitness(S_c) then
            if Fitness(S_kid) > Fitness(S_sib) then
                S_sib = S_kid ;
                a_s = a_j ;
        else
            AS = AS \ {a_j}                            /* Best kid not better than its parent; */
    if Fitness(S_sib) > Fitness(S_c) then
        S_c = S_sib ;                                  /* Initiate next iteration */ ;
        AS = AS \ {a_s} ;
    else
        break ;
    t = t+1 ;
```

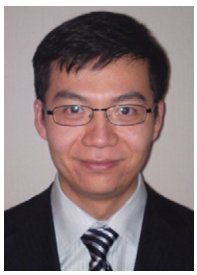
Appendix B. Block enumeration algorithm

```
input  : A process model S = (N, E, ...) and its order matrix A
output : A set BS with all possible blocks
Define BS_x be a set of blocks containing blocks with x activities. x = (1, ..., n);
Define each activity a_i as a block B, i = (1, ..., n) ;
BS_1 = {B_1, ..., B_n}.                                /* initial state */ ;
for i = 2 to n do                                      /* Compute BS_i */;
    let j = 1; let k = i;
    while j ≤ k do
        k = i - j      /* A block containing k activities can only be obtained by merging
        blocks containing i and j activities */;
        foreach (B_j, B_k) ∈ BS_j × BS_k do     /* judge whether B_j and B_k can form a block */
            merge = TRUE; if B_j ⋂ B_k = ∅ then                    /* Disjoint?  */
                foreach (a_α, a_β, a_γ) ∈ B_j × B_k × (N \ B_j ⋃ B_k) do
                    if A_{a_α a_γ} ≠ A_{a_β a_γ} then
                        merge = FALSE     /* two blocks con merge only if they show same
                        order relations to the activities out side the two blocks */;
                        break ;
            else
                merge = FALSE;
            if merge = TRUE then
                B_p = B_j ⋃ B_k;
                BS_i = BS_i ⋃ B_p;
        j = j + 1 ;
BS = ⋃_{x=1}^{n} BS_x
```

References

[1] B. Mutschler, M. Reichert, J. Bumiller, Unleashing the effectiveness of process-oriented information systems: problem analysis, critical success factors and implications, IEEE Trans. Sys. Man. Cybern. 38 (3) (2008) 280–291.
[2] R. Lenz, M. Reichert, IT support for healthcare processes — premises, challenges, perspectives, Data Knowl. Eng. 61 (1) (2007) 39–58.
[3] T.H. Davenport, *Mission Critical — Realizing the Promise of Enterprise Systems*, Harvard Business School, 2000.
[4] B. Weber, M. Reichert, S. Rinderle-Ma, Change patterns and change support features — enhancing flexibility in process-aware information systems, Data Knowl. Eng. 66 (3) (2008) 438–466.
[5] B. Weber, S. Sadiq, M. Reichert, Beyond rigidity — dynamic process lifecycle support: a survey on dynamic changes in process-aware information systems, Comput. Sci. R&D 23 (2) (2009) 47–65.
[6] A. Hallerbach, T. Bauer, M. Reichert, Managing process variants in the process lifecycle, *ICEIS'08*, Springer, 2008, pp. 154–161.
[7] M. Rosemann, W.M.P. van der Aalst, A configurable reference modelling language, Inf. Syst. 32 (1) (2007) 1–23.
[8] M. Reichert, P. Dadam, ADEPTflex — supporting dynamic changes of workflows without losing control, J. Intell. Inf. Syst. 10 (2) (1998) 93–129.
[9] M. Rosenmann, Potential pitfalls of process modeling: part b, BPM J. 12 (3) (2006) 127–136.
[10] A. Hallerbach, T. Bauer, M. Reichert, Capturing variability in business process models: the Provop approach, Softw. Process Improv. Pract. (2009).
[11] B. Weber, M. Reichert, W. Wild, S. Rinderle-Ma, Providing integrated life cycle support in process-aware information systems, Int' J. Coop. Inf. Syst. 19 (1) (2009), (IJCIS).
[12] R.M. Dijkman, M. Dumas, L. Garcia-Banuelos, R. Kaarik, Aligning business process models, *EDOC'09*, 2009, pp. 45–53.
[13] C.W. Günther, S. Rinderle-Ma, M. Reichert, W.M.P. van der Aalst, J. Recker, Using process mining to learn from process changes in evolutionary systems, Int. J. Bus. Process Int. Mgnt. 3 (1) (2008) 61–78.
[14] A. Rozinat, W.M.P. van der Aalst, Conformance checking of processes based on monitoring real behavior, Inf. Syst. 33 (1) (2008) 64–95.
[15] A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, NL, 2006.
[16] W.M.P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (9) (2004) 1128–1142.
[17] A.J.M.M. Weijters, W.M.P. van der Aalst, Rediscovering workflow models from event-based data using little thumb, Integr. Comp. Aided Eng. 10 (2) (2003) 151–162.
[18] B.F. van Dongen, W.M.P. van der Aalst, Multi-phase process mining: building instance graphs, *ER'04*, Springer, 2004, pp. 362–376, LNCS 3288.
[19] C. Li, M. Reichert, A. Wombacher, Discovering reference models by mining process variants using a heuristic approach, *BPM'09, LNCS 5701*, Springer, 2009, pp. 344–362.
[20] W.M.P. van der Aalst, T. Basten, Inheritance of workflows: an approach to tackling problems related to change, Theor. Comput. Sci. 270 (1–2) (2002) 125–203.
[21] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros, Workflow patterns, *Distributed and Parallel Databases*, 14, 1, 2003, pp. 5–51.
[22] M. zur Muehlen, J. Recker, How much language is enough? theoretical and practical use of the business process modeling notation, *CAiSE'08*, Springer, 2008, pp. 465–479, LNCS 5074.
[23] J. Mendling, *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction and Guidelines for Correctness*, volume 6 of *LNBIP*, Springer, 2008.
[24] J. Mendling, B.F. van Dongen, W.M.P. van der Aalst, Getting rid of or-joins and multiple start events in business process models, Enterp. Inf. Syst. 2 (4) (2008) 403–419.
[25] J. Mendling, H.A. Reijers, W.M.P. van der Aalst, Seven process modeling guidelines (7pmg), Inf. Softw. Technol. 52 (2) (2010) 127–136.
[26] B. Kiepuszewski, A.H.M. ter Hofstede, C. Bussler, On structured workflow modelling, *CAiSE'00*, Springer, 2000, pp. 431–445, LNCS 1789.
[27] P. Dadam, M. Reichert, The ADEPT project: a decade of research and development for robust and flexible process support — challenges and achievements, Comput. Sci. R&D 23 (2) (2009) 81–97.
[28] M. Minor, A. Tartakovski, D. Schmalenand, R. Bergmann, Agile workflow technology and case-based change reuse for long-term processes, Int. J. Intell. Inf. Technol. 4 (1) (2008) 80–98.
[29] H.A. Reijers, J. Mendling, Modularity in Process Models: Review and effects, Springer, 2008, pp. 20–35, LNCS 5240.
[30] C. Combi, M. Gambini, Flaws in the flow: the weakness of unstructured business process modeling languages dealing with data, *OTM Conferences*, 1, Springer, 2009, pp. 42–59, LNCS 5074.
[31] J. Vanhatalo, H. Volzer, J. Koehler, The refined process structure tree, Data Knowl. Eng. 68 (9) (2009) 793–818.
[32] L. Thom, M. Reichert, C. Iochpe, Activity patterns in process-aware information systems: basic concepts and empirical evidence, Int. J. Bus. Process Int. Mange. 4 (2) (2009) 93–110.
[33] S. Rinderle-Ma, M. Reichert, B. Weber, On the formal semantics of change patterns in process-aware information systems, *ER'08*, LNCS 5231, 2008, pp. 279–293.
[34] M. Reichert, S. Rinderle, U. Kreher, P. Dadam, Adaptive process management with ADEPT2, *ICDE'05*, IEEE, 2005, pp. 1113–1114.
[35] C. Li, M. Reichert, A. Wombacher, On measuring process model similarity based on high-level change operations, *ER'08*, Springer, 2008, pp. 248–262, LNCS 5231.
[36] J. Hidders, M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede, J. Verelst, When are two workflows the same, *CATS'05*, 2005, pp. 3–11, Darlinghurst, Australia.
[37] M. Reichert, S. Rinderle-Ma, P. Dadam, Flexibility in process-aware information systems, LNCS Trans. Petri Nets Other Models Concurrency 2 (2009) 115–135.
[38] P.F. Brown, V.J. Della Pietra, P.V. de Souza, J.C. Lai, R.L. Mercer, Class-based n-gram models of natural language, Comput. Ling. 18 (4) (1992) 467–479.
[39] A. Wombacher, M. Rozie, Evaluation of workflow similarity measures in service discovery, Serv. Oriented Electron. Commer. (2006) 51–71.
[40] R.A. Baeza-Yates, Text-retrieval: theory and practice, *IFIP'92*, North-Holland Co, 1992, pp. 465–476.
[41] C. Li, M. Reichert, and A. Wombacher. Representing block-structured process models as order matrices: Basic concepts, formal properties, algorithms. Technical Report TR-CTIT-09–47, University of Twente, 2009.
[42] G.F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, Pearson, 2005.
[43] P.N. Tan, M. Steinbach, V. Kumar, *Introduction to Data Mining*, Addison-Wesley, 2005.
[44] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc, 1993.
[45] C. Li, M. Reichert, A. Wombacher, A heuristic approach for discovering reference models by mining process model variants, Technical Report TR-CTIT-09-08, University of Twente, The Netherlands, March 2009.
[46] D.J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, CRC Press, 2004.
[47] A. Wombacher, P. Fankhauser, E. Neuhold, Transforming bpel into annotated deterministic finite state automata for service discovery, *Web Services, IEEE International Conference on*, 0:316, 2004.
[48] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, W.M.P. van der Aalst, The prom framework: a new era in process mining tool support, *ICATPN*, 2005, p. 3536, LNCS.
[49] J. Dehnert, R. Rittgen, Relaxed soundness of business processes, *CAiSE'01*, Springer, 2001, pp. 157–170, LNCS 2068.
[50] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, M. La Rosa, Configurable workflow models, Int. J. Coop. Inf. Syst. 17 (2) (2008) 177–221.
[51] R. Lu, S.W. Sadiq, Managing process variants as an information resource, *BPM'06*, 2006, pp. 426–431.
[52] R. Lu, S.W. Sadiq, On the discovery of preferred work practice through business process variants, *ER*, Springer, 2007, pp. 165–180.
[53] J. Bae, L. Liu, J. Caverlee, W.B. Rouse, Process mining, discovery, and integration using distance measures, *ICWS'06*, 2006, pp. 479–488, Washington, DC, USA.
[54] M. Kuramochi, G. Karypis, Frequent subgraph discovery, *ICDM'01*, IEEE, 2001, pp. 313–320.
[55] K.H. Rosen, *Discrete Mathematics and Its Application*, McGraw-Hill, 2003.
[56] C. Li, M. Reichert, A. Wombacher, What are the problem makers: ranking activities according to their relevance for process changes, *ICWS'09*, IEEE, 2009, pp. 51–58.

[57] M. La Rosa, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, Questionnaire-based variability modeling for system configuration, Softw. Syst. Model. 8 (2) (2009) 251–274.
[58] F. Gottschalk, T.A.C. Wagemakers, M.H. Jansen-Vullers, W.M.P. van der Aalst, M. La Rosa, Configurable process models: experiences from a municipality case study, *CAiSE'09*, 2009, pp. 486–500.
[59] J.S. Ash, M. Berg, E. Coiera, Some unintended consequences of information technology in health care: the nature of patient care information system-related errors, J. Am. Med. Inf. Ass. 11 (2) (2004) 104–112.



Chen Li is a Ph.D student at University of Twente, the Netherlands. Previously, he obtained his Master degree at Utrecht University, the Netherlands. His Ph.D research focuses on process variant management and process mining. In addition, his research interests also include business intelligence, simulation, statistic analysis and requirement engineering.

Manfred Reichert holds a PhD in Computer Science and a Diploma in Mathematics. Since January 2008 he has been appointed as full professor at the University of Ulm. Earlier, he was working as Associate Professor at the University of Twente (UT) in the Netherlands. His major research interests include next generation process management technology (e.g., adaptive processes, process lifecycle management, data-driven processes, mobile processes), service-oriented architectures (e.g., service interoperability, service evolution), and advanced applications for IT solutions (e.g., e-health, automotive engineering). Together with Peter Dadam he pioneered the work on the ADEPT process management system. Manfred has been participating in numerous research projects in the BPM area and contributed numerous papers. Further, he has co-organized international and national conferences and workshops. Manfred was PC-Co-Chair of the BPM'08 conference in Milan and General Co-Chair of the BPM'09 conference in Ulm.

Andreas Wombacher is an assistant professor at University Twente and co-coordinator of the strategic research objective Applied Science of Services for Information Society Technologies (ASSIST). He did his master and Ph.D. degree at the Technical University of Darmstadt. He gathered professional experience at IBM (Germany), the Integrated Publication and Information Systems Institute (IPSI) of GMD (Germany), the University of Twente (Netherlands), and the Swiss Federal Institute of Technology in Lausanne (EPFL). His research interests are in the area of service oriented architectures, distributed data management, and data processing with a focus on sensor networks always under the perspective of processes. He has been involved in several organization and program committees.