What are the Problem Makers: Discovering the Most Frequently Changed Activities in Adaptive Processes

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

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

Abstract. Recently, a new generation of adaptive Process-Aware Information System (PAIS) has emerged, which enables dynamic service changes (i.e., changes of instances derived from a composite service and process respectively). This, in turn, results in a large number of process variants derived from the same process model, but differing in their structure due to the applied changes. Since such process variants are expensive to maintain, the process model should evolve accordingly. It is therefore our goal to discover those activities that have been more often involved in process (instance) adaptations than others, such that we can focus on them when re-designing the process model. This paper provides two approaches to rank activities based on their involvement in process adaptations and process configurations respectively. The first approach allows to precisely rank the activities, but it is very expensive to perform since the algorithm is at $\mathcal{NP}$ level. We therefore provide as alternative approach an approximation ranking algorithm which computes in polynomial time. The performance of the approximation algorithm is evaluated and compared through a comprehensive simulation of 3600 process models. By applying statistical significance tests, we can also identify several factors which influence the performance of the approximation ranking algorithm.

1 Introduction

In today's dynamic business world, success of an enterprise increasingly depends on its ability to react to changes in its environment in a quick, flexible and cost-effective way [16]. Along this trend a variety of process and service support paradigms as well as corresponding specification languages (e.g., WS-BPEL, WS-CDL) have emerged. In addition, different approaches for flexible processes and

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

services respectively exist [17, 20]. Generally, adaptations of composite services and processes are not only needed for configuration purposes at buildtime, but also become necessary during runtime to deal with exceptional situations and changing needs; i.e., for single instances of composite services and processes respectively, it must be possible to dynamically adapt their structure (e.g. to insert, delete or move activities during runtime).

In response to this need adaptive process management technology has emerged [29]. It allows to adapt and configure process models at different levels. This, in turn, results in large collections of process model variants (*process variants* for short), which are created from the same process model, but slightly differ from each other in their structure. Fig. 1 depicts an example. The left hand side shows a high-level view on a patient treatment process as it is normally executed: a patient is *admitted* to a hospital, where he first *registers*, then *receives treatment*, and finally *pays*. In emergency situations, however, it might become necessary to deviate from this model, e.g., by first starting treatment of the patient and allowing him to register later during treatment. To capture this behavior in the model of the corresponding process instance, we need to move activity *receive treatment* from its current position to a position parallel to activity *register*. This leads to an instance-specific process model variant S' as shown on the right hand side of Fig. 1. Generally, a large number of process variants may exist in a Process-Aware Information System (PAIS) at both the process type and process instance level [14].

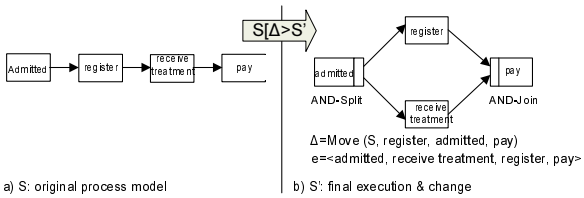


Fig. 1. Original Process Model S and Process Variant S'

In most approaches which allow for the adaptation and configuration of process models, the resulting process variants have to be maintained separately. Then even simple changes (e.g. due to new laws) often require manual re-editing of a large number of process variants. Over time this leads to divergence of the respective process models, which aggravates their maintenance significantly [31].

Considering a given reference process model and analyzing the collection of process variants configured from it, this paper aims at finding the problem makers, i.e., the activities that are involved in process adaptations more often than others. These activities, in turn, cause most deviations from the reference process model and thus lead to highest configuration effort. In particular, we provide algorithms that solely use the reference process model and a collection of variants derived from it as input; i.e., we do not require the presence of a change log [18,

19] The discovered information is particularly useful for monitoring the deviations from the predefined process model or for redesigning it through learning from past executions.

Based on the two assumptions that: (1) process models are block-structured [17] (like for example BPEL [4]) and (2) all activities in a process model have unique labels, this paper deals with the following fundamental research question: *Given a reference process model and a collection of process variants configured from it, how to rank the activities according to their involvement in structural process adaptations (i.e., the adaptations that become necessary when configuring the process variants)?*

The remainder of this paper is organized as follows: Section 2 gives background information needed for understanding this paper. To illustrate our algorithms, we provide an running example in Section 3. We provide a precise, but expensive ranking algorithm in Section 4 and a more efficient approximation ranking algorithm in Section 5. To test the performance of the two algorithms, we conduct comprehensive simulation. Section 6 describes the setup of this simulation whereas Section 7 presents its result. Finally, Section 8 discusses related work and Section 9 concludes with a summary and an outlook.

2 Backgrounds

We first introduce basic notions needed in the following:

Process Model: Let $\mathcal{P}$ denote the set of all sound process models. A particular *process model* $S = (N, E, \ldots) \in \mathcal{P}$ is defined as well-structured Activity Net [17, 29]. N constitutes the set of process activities and E the set of control edges (i.e., precedence relations) linking them. To limit the scope, we assume Activity Nets to be block-structured like in BPEL. An example is provided in Fig. 1.

Process change A process change is accomplished by applying a sequence of change operations to the process model S over time [17]. Such change operations modify the initial process model by altering the set of activities and their order relations. Thus, each application of a change operation results in a new process model. We define *process change* and *process variant* as follows:

Definition 1 (Process Change and Process Variant). *Let $\mathcal{P}$ denote the set of possible process models and $\mathcal{C}$ be the set of possible process changes. Let $S, S' \in \mathcal{P}$ be two process models, let $\Delta \in \mathcal{C}$ be a process change expressed in terms of a high-level change operation, and let $\sigma = \langle \Delta_1, \Delta_2, \ldots \Delta_n \rangle \in \mathcal{C}^*$ be a sequence of process changes performed on initial model S. Then:*

- *$S[\Delta\rangle S'$ iff Δ is applicable to S and S' is the (sound) process model resulting from the application of Δ to S.*
- *$S[\sigma\rangle S'$ 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 denote S' as variant of S.*

[4] see [28] for a technique transforming an unstructured process a model to block (tree) structured model

Examples of high-level change operations include *insert activity*, *delete activity*, and *move activity* as implemented in the ADEPT change framework [17]. While *insert* and *delete* modify the set of activities in the process model, *move* changes activity positions and thus the order relations of the process model. For example, operation $move(S, \texttt{A,B,C})$ moves activity `A` from its current position within process model S to the position after activity `B` and before activity `C`. Operation $delete(S, A)$, in turn, deletes activity `A` from process model S. Issues concerning the correct use of these operations, their generalizations, and formal pre-/post-conditions are described in [17]. Though the depicted change operations are discussed in relation to our ADEPT approach, they are generic in the sense that they can be easily applied in connection with other process meta models as well [29]. For example, a process change as described in the ADEPT framework can be mapped to the concept of life-cycle inheritance known from Petri Nets [25]. We refer to ADEPT since it covers by far most high-level change patterns and change support features when compared to other approaches [29], and it offers a fully implemented adaptive process engine.

Definition 2 (Distance and Bias). *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 process model S into process model S'; i.e., $d_{(S,S')} := min\{|\sigma| \mid \sigma \in \mathcal{C}^* \wedge S[\sigma\rangle S'\}$. Furthermore, a sequence of change operations σ with $S[\sigma\rangle S'$ and $|\sigma| = d_{(S,S')}$ is denoted as a* ***bias*** *between S and S'. All the biases are summarized in a set $\mathcal{B}_{(S,S')} = \{\sigma \in \mathcal{C}^* | \ |\sigma| = d_{S,S'}\}$, which we denote this set as the* ***bias set.***

The *distance* between two process models S and S' is the minimal number of high-level change operations needed for transforming S into S'. Usually, it measures the complexity for model transformation. The corresponding sequence of change operations is denoted as *bias* between S and S'. Generally, it is possible to have more than one minimal sequence of change operations to realize the transformation from S into S', i.e., given two process models S and S' their bias is not necessarily unique [26, 12]. As example consider Fig. 1. Here, the distance between model S and process variant S' is *one*, since we only need to perform one change operation $\Delta_1 = move(S, \texttt{register, addmitted, pay})$ to transform S into S_1. However, it is also possible to transform S into S' with $\Delta_2 = move(S, \texttt{receive treatment, admitted, pay})$. Therefore, we obtain $\mathcal{B}_{(S,S')} = \{\Delta_1, \Delta_2\}$ as bias set. In general, determining the bias and distance between two process models has complexity at $\mathcal{NP}$ level (see [12] for a computation method). Here, we use high-level change operations rather than change primitives (i.e. elementary changes like adding or removing nodes and edges) to measure the distance between process models. This allows to guarantee soundness of process models and also provides a more meaningful measure for distance [12].

Trace: A *trace* t on process model $S = (N, E, \ldots)$ denotes a valid and complete execution sequence $t \equiv < a_1, a_2, \ldots, a_k >$ of activities $a_i \in N$ according to the control flow set out by S. All traces S can produce are summarized in trace set $\mathcal{T}_S$. $t(a \prec b)$ is denoted as precedence relation between activities a

and b in trace $t \equiv < a_1, a_2, \ldots, a_k >$ iff $\exists i < j : a_i = a \wedge a_j = b$. Here, we only consider traces composing 'real' activities, but no events related to silent ones (i.e. activity nodes which contain no action and exist only for control flow purpose [12]). Finally, we consider two process models being the same if they are *trace equivalent*, i.e., $S \equiv S'$ iff $\mathcal{T}_S \equiv \mathcal{T}_{S'}$. The stronger notion of bi-similarity [8] is not required here.

3 Running Example

Fig. 2 gives an example, which is used for illustration purpose through out this paper. Regarding this example, out of a reference model S, five different process variants $S_i \in \mathcal{P}$ $(i = 1, 2, \ldots 5)$ have been configured, which are weighted based on the number of process instances created from them. In our example, 30% of all process instances were executed according to process variant S_1, while 15% of the instances ran on S_2. If we only know process variants, but have no runtime information about related instance executions, we assume the variants to be equally weighted; i.e., every process variant then has weight $1/n$, where n corresponds to the number of given variants.

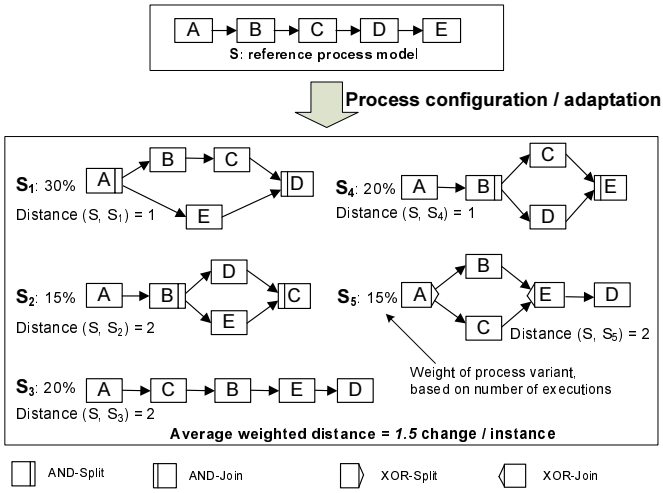


Fig. 2. One illustrative example

We first compute the distances (cf. Def. 2) between process model S and its variants. For example, when comparing S with S_1 we obtain distance *one*, i.e., we only need to perform one change operation (i.e. $move(S, \texttt{E,A,D})$) to transform S into S_1. Or when comparing S with S_2, needed change operations are $move(S, \texttt{D,B,C})$ and $move(S, \texttt{E,B,C})$, and distance between S and S_2 is *two*. Based on the weight of each variant, we can compute the *average weighted distance* between reference model S and its variants; e.g., the distances between

S and S_i, and the weights are depicted in Fig. 2. As average weighted distance we obtain $1 \times 0.3 + 2 \times 0.15 + 2 \times 0.2 + 1 \times 0.2 + 2 \times 0.15 = 1.5$. This means we need to perform on average 1.5 change operations to transform the reference model to a process variant and corresponding instance respectively. Generally, the average weighted distance between a reference model and its process variants expresses how "*close*" they are; i.e., the higher the average weighted distance is, the greater configuration efforts have become

Though we are able to compute the distances between the reference model and each variant, it is not clear which activities are most involved in these configurations. Clearly, we should focus on the activities which are more often involved in these configurations, as they cause the major effort with respect to process configuration. In this context, we need an approach for ranking the activities based on their involvement in process variants configurations. Since we do not presume any run-time data (like change logs [19], or execution logs [27]), it is not possible to know which activities have been involved in process configurations particularly. In the following section, we will provide a method to measure the potential involvement of each activity a_i in process configurations, which we denote as **change impact** i.e., $CI(a_i)$. And based on the change impact of each activity, we are able to rank the activities, lets denote such ranking as **change impact ranking list**.

In this paper, we are interested in detecting changes of order relations in process models. Therefore, we only consider *move* operations but factor out *insert* or *delete* operations. Note that the latter can be easily detected by comparing activity sets of two process models (for details, see [10, 12]).

4 Computing the Precise Change Impact Ranking List

Since we do not presume the presence of a change log or execution log respectively, the major information we can use for our analysis are the bias set $\mathcal{B}_{S,S_i}$ which describe the structural differences between the reference process model and each of its variant S_i. From the bias set, we are able to compute the minimal number of change operations needed to transform the reference model S into a particular variant S_i. The bias set, therefore, can be considered as a purified change log for our analysis. In this section, we present a method to compute the change impact of each activity based on the bias sets. A general description of our approach is as follows:

1. We compute the bias sets $\mathcal{B}_{S,S_i}$ which describe the structural difference between the reference model S and all the variants S_i (cf. Section 4.1).
2. For each bias set $\mathcal{B}_{S,S_i}$, we measure the involvement of each activity a_i in the captured adaptation. (cf. Section 4.2).
3. The change impact $CI(a_i)$ of an activity a_i is then measured by its involvement in all bias sets $\mathcal{B}_{S,S_i}$ (cf. Section 4.2).

4.1 Computing Biases

Let us re-consider the example from Fig. 2. By scanning the reference process model S and a process variant $S_i (i = 1 \ldots 5)$, we are able to compute bias set $\mathcal{B}_{S,S_i}$ [12]. This bias set contains all possible sequences of change operations transforming S into S_i with minimal number of change operations. However, the definition of bias set is too strict in our context, since we are only interested in the activities being involved in adaptation rather than the order in which the different changes were applied. For example, the bias set $\mathcal{B}_{(S,S_2)}$ comprise the two changes σ_1, σ_2 where $\sigma_1 =< \Delta_1, \Delta_2 >$ with $\Delta_1 = move(S, \texttt{D, B, C})$ and $\Delta_2 = move(S, \texttt{E, B, C})$ and $\sigma_2 =< \Delta_2, \Delta_1 >$. Although $\sigma_1 \neq \sigma_2$, this difference is not relevant in our context since we are only interested in the activities being changed rather than the order in which different changes were applied.

Therefore, we keep the granularity of our bias analysis only on the activities that have been involved in changes changed rather than the applied change operations. Regarding our example, we only want to document these activities i.e., {D,E} in the context of σ_1 (see above) rather than the change operation σ_1 itself. When only looking at the changed activities, σ_1 does not differ from σ_2 since the activities concerned by the changes in the two biases are exactly the same.

Definition 3 (Changed Activity Set).
*We define set $\mathcal{A}_\sigma$ representing the activities changed by any change operation of bias σ, i.e., $\mathcal{A}_\sigma = \{a_i | (a_i \text{ is an activity changed by } \Delta_i) \wedge (\Delta_i \in \sigma)\}$. We define $\mathcal{C}_{(S,S')} = \{\mathcal{A}_\sigma | \sigma \in \mathcal{B}_{(S,S')}\}$ as the **Changed Activity Set** of S and S'.*

According to Def. 3, an element $\mathcal{A}_\sigma$ of the Changed Activity Set $\mathcal{C}_{(S,S')}$ corresponds to a set representing the activities changed by bias σ. Regarding our example from Fig. 2, the changed activity sets of the reference model S and its variants S_i $(i = 1 \ldots 5)$ are listed in Table 1.

Models	Changed Activity Set
$\mathcal{C}_{(S,S_1)}$	{{E}}
$\mathcal{C}_{(S,S_2)}$	{{D,E}, {C,D}, {C,E}}
$\mathcal{C}_{(S,S_3)}$	{{B,D}, {B,E}, {C,D}, {C,E}}
$\mathcal{C}_{(S,S_4)}$	{{C}, {D}}
$\mathcal{C}_{(S,S_5)}$	{{B,D}, {B,E}, {C,D}, {C,E}}

Table 1. the Changed activity sets between reference model and all variants

As example, consider $\mathcal{C}_{(S,S_2)}$. We can either move activities D and E, or activities C and D or activities C and E to transform model S into S_2. When further analyzing Table 1, we can see that $\mathcal{C}_{(S,S_3)}$ is exactly the same as $\mathcal{C}_{(S,S_5)}$ though the two process models S_3 and S_5 are quite different (cf. Fig. 2). The reason behind is that, for example, regarding move operations, the Changed Activity

Set only documents which activity has potentially been changed but does not specify to which position the activity have been moved to. Consequently, bias sets $\mathcal{B}_{(S,S_3)}$ and $\mathcal{B}_{(S,S_5)}$, which document the complete information about the changes, are rather different. However, in connection with the research question described in Section 1, we are only interested in activities which have been potentially involved in a change.

4.2 Computing the Change Impact $CI(a_i)$ of Each Activity a_i

We measure the change impact $CI(a_i)$ of each activity a_i by computing its contribution to the average weighted distance between the reference model and its variants. As example, consider distance $d_{(S,S_2)}$ between reference model S and variant S_2, which is 2. Here we want to measure how much each activity has contributed to this distance. We measure it by analyzing changed activity set $\mathcal{C}_{S,S_2}$.

In general, for a given changed activity set, we enumerate all possible solutions to transform the reference process model S into the variants S_i, i.e., $\forall \mathcal{A}_\sigma \in \mathcal{C}_{(S,S_i)}$: $\exists \sigma \in \mathcal{B}_{(S,S_i)}$. Note that we do not have any information about which sequence of change operations was applied to configure S_i out of S. Therefore, to each $\mathcal{A}_\sigma \in \mathcal{C}_{S,S_i}$ we assign same weight $d_{(S,S_i)}/|\mathcal{C}_{S,S_i}|$. Finally, for a particular activity $a_j \in \mathcal{A}_\sigma \in \mathcal{C}_{S,S_i}$, we choose $\frac{d_{(S,S_i)}}{|\mathcal{C}_{(S,S_i)}|} \times \frac{1}{|\mathcal{A}_\sigma|}$.

Consider again the definition for the distance between process models S and S_i (cf. Def. 2). Then $d_{(S,S_i)} = |\mathcal{A}_\sigma|$ holds, since bias σ is a sequence with minimal number of change operations to transform S into S_i. Consequently, the change impact for each activity a_j between S and a particular S_i can be computed as $\frac{|\{\mathcal{A}_\sigma \in \mathcal{C}_{(S,S_i)} | a_j \in \mathcal{A}_\sigma\}|}{|\mathcal{C}_{(S,S_i)}|}$.

Let S be the reference model and let $S_i (i = 1, \ldots, n)$ be weighted process variants S_i (with weight w_i and $\Sigma_1^n w_i = 1$) derived from S. Then the Change Impact $CI(a_j)$ of a particular activity a_j, which measures its potential involvement in process adaptations, can be computed as follows:

$$CI(a_j) = \sum_{i=1}^{n} w_i \times \frac{|\{\mathcal{A}_\sigma \in \mathcal{C}_{(S,S_i)} \mid a_j \in \mathcal{A}_\sigma\}|}{|\mathcal{C}_{(S,S_i)}|} \quad (1)$$

Figure 3 summarizes the Change Impacts of the activities from our example (cf. Figure 2). When reading the depicted table horizontally, it shows the potential involvement of each activity when configuring the reference model S into a particular variant S_i. As example take S_4. The distance between S and S_4 is one, since both activities `C` and `D` could be potentially involved in the corresponding configuration (cf. Table 1). Each of the two activities therefore obtain Change Impact of 0.5 from this particular variant. When reading Figure 3 vertically, it shows the Change Impact of each activity based on the observation of all variants. For example, activity `B` shows Change Impact of 0.5 for configuring variant S_3 and 0.5 for configuring variant S_5. Considering the weight of each variant, the change impact of activity `B` $CI(\texttt{B})$ can be computed by Formula (1). As result

we obtain 0.175, which means that on average we need to move activity B 0.175 times when configuring a variant out of the given reference model.

Variant \ Activity	A	B	C	D	E	total
S_1 (weight: 30%)	0	0	0	0	1	1
S_2 (weight: 15%)	0	0	0.67	0.67	0.67	2
S_3 (weight: 20%)	0	0.5	0.5	0.5	0.5	2
S_4 (weight: 20%)	0	0	0.5	0.5	0	1
S_5 (weight: 15%)	0	0.5	0.5	0.5	0.5	2
Change Impact	0	0.175	0.375	0.375	0.575	1.5
Ranking	5	4	2	2	1	

Fig. 3. The change impact for each activity

Based on the change impact of each activity, we can also rank them accordingly. Clearly, activity E has the highest change impact and therefore should be ranked first. Consequently, if we need to find an activity of the reference process model for re-positioning, activity E will be the first candidate. Reason is that it has been reconfigured (i.e., re-positioned) more often than the other activities.

In fact, we have determined the contribution each activity has on the average weighted distance between the reference model and its variants. If we sum up the change impact of all activities, we obtain $0 + 0.175 + 0.375 + 0.375 + 0.575 = 1.5$. Note that this corresponds to the actual value of the average weighted distance (cf. Fig. 2).

4.3 Discussion

The approach we described in this section is very precise: all possible changes between the reference model and a process variant are enumerated and the change impact of a particular activity is computed by analyzing the reference model and all variants. However, enumerating all possible changes between two models is a $\mathcal{NP}$ problem, this approach can be very expensive. We need to call the $\mathcal{NP}$ algorithm every time we want to find possible changes between the reference model and a particular variant. Therefore, this approach will not scale up. If we have to deal with a large number of variants with complex structures (like models with tens up to hundreds of activities). In the next section, we introduce an approximation algorithm to solve the problem in an efficient way.

5 Compute the Approximation Change Impact Ranking List

To reduce the complexity for computing the change impact of each activity, we now introduce an approximation algorithm which only requires polynomial time to compute the ranking result. We first introduce the notion of order matrix in Section 5.1 and the notion of aggregated order matrix in Section 5.2. Section 5.3 then presents our approximation ranking algorithm.

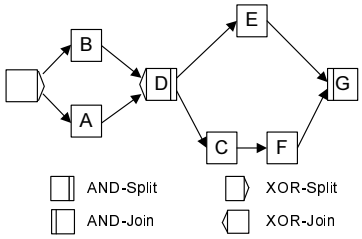
5.1 Representing Process Models as Order Matrices

Theoretical backgrounds of high-level change operations have been extensively discussed in ADEPT [17]. One key feature of our ADEPT change framework is to maintain the structure of the unchanged parts of a process model [17]. For example, when deleting an activity this neither influences the successors nor predecessors of this activity, and therefore also not their order relations. To incorporate this feature in our approach, rather than only looking at direct predecessor-successor relationships between activities (i.e. control edges), we consider the transitive control dependencies for each activity pairs; i.e. for a given process model $S = (N, E, \ldots) \in \mathcal{P}$, we examine for every pair of activities $a_i, a_j \in N$, $a_i \neq a_j$ their transitive order relations. Logically, we determine order relations by considering all traces the process model may produce (cf. Section 2). Results are aggregated in an order matrix $A_{|N|\times|N|}$, which considers four types of control relations (cf. Def. 4):

Definition 4 (Order matrix). *Let $S = (N, E, \ldots) \in \mathcal{P}$ be a process model with $N = \{a_1, a_2, \ldots, a_n\}$. Let further $\mathcal{T}_S$ denote the set of all traces producible on S. Then: Matrix $A_{|N|\times|N|}$ is called **order matrix** of S with A_{ij} representing the order relation between activities a_i,$a_j \in N$, $i \neq j$ iff:*

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

Fig. 4 shows an example. Besides control edges, which express direct predecessor-successor relationships, process model S also contains four kinds of control connectors: AND-Split and AND-Join (corresponding to a flow activity in BPEL),



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

Order matrix of S

'0' : successor
'1' : predecessor
'*' : AND-block
'-' : XOR-block

Fig. 4. Process model and its order matrix

and XOR-Split and XOR-join (corresponding to a switch or pick activity in BPEL). The depicted order matrix represents all four described relations. For example activities `A` and `B` will never appear in the same trace since they are contained in different branches of an XOR block. Therefore, we assign '-' to matrix element A_{AB}. Similarly, we can obtain the relation for each pair of activities. The main diagonal of the matrix is empty since we do not compare an activity with itself.

Under certain conditions, an order matrix uniquely represents the process model it was created from. This is stated by Theorem 1. Before giving this theorem, we need to define the notion of *substring of trace*:

Definition 5 (Substring of trace). *Let $S \in \mathcal{P}$ be a process model and let $t, t' \in \mathcal{T}_S$ be two traces on S. We denote t as sub-string of t' iff $[\forall a_i,\ a_j \in t,\ t(a_i \prec a_j) \Rightarrow a_i,\ a_j \in t' \wedge t'(a_i \prec a_j)]$ and $[\exists a_k \in N\colon a_k \notin t \wedge a_k \in t']$.*

Theorem 1. *Let $S, S' \in \mathcal{P}$ be two process models with same activity set $N = \{a_1, a_2, \ldots, a_n\}$. Let further $\mathcal{T}_S$, $\mathcal{T}_{S'}$ be the related trace sets and $A_{n\times n}$, $A'_{n\times n}$ be the order matrices of S and S'. Then $S \neq S' \Leftrightarrow A \neq A'$, if $[\neg\exists t_1, t'_1 \in \mathcal{T}_S\colon t_1$ is a substring of $t'_1]$ and $[\neg\exists t_2, t'_2 \in \mathcal{T}_{S'}\colon t_2$ is a substring of $t'_2]$.*

We give a proof of Theorem 1 in [12]. According to this theorem, there will be a one-to-one mapping between a process model S and its order matrix A, if the substring constraint is met. (Note that the substring constraint can be easily checked and handled [12]); i.e., if the conditions of Theorem 1 are met, the order matrix will uniquely represent the process model. Analyzing its order matrix (cf. Def. 4) will then be sufficient in order to analyze the process model.

5.2 Aggregated Order Matrix

In order to analyze a given collection of process variants, we first compute the order matrix for each of these process variants (cf. Def. 4). Regarding our example from Fig. 2, for instance we obtain five order matrices (cf. Fig. 5). Then, we analyze the order relation for each pair of activities considering all five order matrices derived before. As the order relation between two activities might be not

the same in all order matrices, this analysis does not result in a fixed relationship, but provides a distribution for the four types of order relations (cf. Def. 4). Regarding our example, in 65% of all cases activity `C` is a successor of activity `B` (as for variants S_1, S_2, S_4), in 20% of all cases `C` is a predecessor of `B` (as in S_3), and in 15% of the cases, `B` and `C` are contained in different branches of an XOR block (as in S_5) (cf. Fig. 5). Therefore, we can define the order relation between two activities `a` and `b` as 4-dimensional vector $V_{ab} = (v^0_{ab}, v^1_{ab}, v^*_{ab}, v^-_{ab})$: each field corresponds to the frequency of the corresponding relation type ('0', '1', '*' or '-') as specified in Def. 4. Take our example from Fig. 5: here $v^1_{\mathtt{CB}} = 0.65$ corresponds to the frequency of all cases with activities `B` and `C` having order relation '1', i.e. all cases for which `C` precedes `B`. Regarding our example, we obtain $V_{CB} = (0.65, 0.2, 0, 0.15)$.

We define an *aggregated order matrix* as follows:

Definition 6 (Aggregated Order Matrix).

Let $S_i \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants with same activity set N. Let further A_i be the order matrix of S_i, and let weight w_i represent the relative frequency of process instances executed on basis of S_i. The ***Aggregated Order Matrix*** *of all process variants is defined as 2-dimensional matrix $V_{m \times m}$ with $m = |N|$ and each matrix element $v_{jk} = (v^0_{jk}, v^1_{jk}, v^*_{jk}, v^-_{jk})$ being a 4-dimensional vector. For $\tau \in \{0, 1, *, -\}$, element v^τ_{jk} expresses to what percentage, activities a_j and a_k have order relation τ within the collection of process variants S_i. Formally:*

$\forall a_j, a_k \in N, a_j \neq a_k, \forall \tau \in \{0, 1, *, -\} : v^\tau_{jk} = (\sum^n_{i=1, A_{i_{jk}} = '\tau'} w_i)/(\sum^n_{i=1} w_i)$.

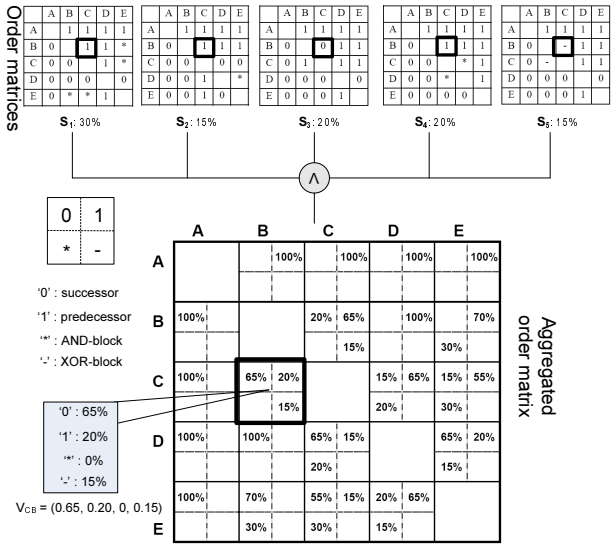


Fig. 5. Aggregated order matrix V

Fig. 5 shows the aggregated order matrix of the process variants from Fig. 2. In an aggregated order matrix, main diagonal is always empty since we do not specify the order relation of an activity with itself. For all other elements, a non-filled value in a certain dimension means it corresponds to zero.

5.3 Approximation Algorithm for Ranking Activities According to Their Change Impact

We have introduced the aggregated order matrix to reflect the fact that the execution orders between two activities may not be the same in different variants. For example, the execution orders between activities `C` and `B` can be represented by $V_{CB} = (0.65, 0.2, 0, 0.15)$. When reconsidering the reference process model from Fig. 2, we can see that the order relation between activities `C` and `B` is "0", i.e., `C` is a successor of `B`. If we built an aggregated order matrix V^{ref} purely based on this reference model, as relationship between `C` and `B` we would obtain $V^{ref}_{CB} = (1, 0, 0, 0)$, i.e., `C` would then always be a successor of `B`. When comparing $V_{CB} = (0.65, 0.2, 0, 0.15)$ (which represents the variants) and V^{ref}_{CB} (which represents the reference model), we can easily see that these two vectors are not the same. This indicates that when configuring reference model into the variants, the position of `B` or `C` might have changed afterwards.

Configuring the reference model into a variant is realized by applying a sequence of change operations (cf. Def. 1). Such an operation either changes the activity set (like *insert* and *delete* operations) or the original relationship between the activities (e.g. *move* operation) of the reference model. Interestingly, the changes of the activity relations are represented by the changes of the order matrix. When configuring the reference model into variant S_1, for example, we need to move activity `E` to the position between `A` and `D`, (i.e., $move(S, \texttt{E, A, D})$). This change influences the order relation of `E` with the other activities. In the reference model S for example, `E` succeeds `A, C`, and `D`. Regarding process variant S_1 configured out of it by applying a move operation, the order relations between `E` on the one side and `B, C` and `D` on the other side is changed, i.e., `E` now precedes `D` and is allocated in parallel to `B` and `C`.

Generally, we can assume that the more an activity is moved, the more its order relation differs from the original one. To quantitatively measure this difference, we compare the order relations set out by the reference model with those of the aggregated order matrix.

Before providing the comparison method, we first introduce function $f(\alpha, \beta)$ which expresses the closeness between two vectors $\alpha = (x_1, x_2, ..., x_n)$ and $\beta = (y_1, y_2, ..., y_n)$:

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

$f(\alpha, \beta) \in [0, 1]$ computes the 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

examples, for instance, we obtain $f(V_{CB}, V_{CB}^{ref}) = 0.933$. This number indicates high similarity between the order relations of the reference model and the ones of the variants (which are represented by the aggregated order matrix).

Based on these considerations, the change impact of a particular activity can be measured using the following formula. To differentiate it from Formula (1), we denote change impact computed by this approximation as $CI_a(a_i)$.

$$CI_a(a_i)^5 = \frac{\sum_{x \in N \setminus \{a_i\}} f^2(V_{a_i x}, V_{a_i x}^{ref})}{|N| - 1} \tag{3}$$

$CI_a(a_i) \in [0, 1]$ corresponds to the average square mean value of the similarity (measured by Formula (2)) between activity a_i and the rest of activities. It therefore approximately reflects how much a_i has been re-configured. If $CI_a(a_i) = 1$ holds, activity a_i will exactly have same order relations with respect to the other activities in both the reference model and all the variants. For this case, We can therefore assume that it has not been moved. $Weight(a_i) = 0$, in turn, means that the order relation of a_i as reflected shown in the reference model is completely different from the order relations in the different variants. We therefore assume that in such case activity a_i has been moved a lot. Note that our ranking is based on descending orders, i.e., the higher the change impact $CI_a(a_i)$ is, the lower the chance will be that it has been potentially moved and the lower such activity should be ranked.

Regarding our example from Fig. 2, the ranking result of the five activities is shown in Table 2.

Activity	E	D	C	B	A
$CI_a(a_i)$	0.6641	0.7384	0.8678	0.9280	1.0000
Rank	1	2	3	4	5

Table 2. Approximate ranking result

From Table 2, we can clearly derive a ranking order of the activities based on their potential involvement in changes. Activity `E` is moved most frequently while activity `A` is the least moved one.

[5] Note that this is not a precise measure since not only the execution orders of the moved activities are affected, but the other activities may be influenced by a change operation as well. For example, when configuring S into S_1, we actually only need to move activity `E`. However, the execution orders of the remaining activities are also changed, e.g., activities `B, C` and `D`. Reason is that move operations can globally influence the execution order while our measure only examines the local information between every pair of activities.

5.4 Comparing the Precision ranking Algorithm and the Approximation Ranking Algorithm

The approximation algorithm presented in Section 5.3 is a polynomial algorithm, i.e., the complexity for computing the change impact $CI_a(a_i)$ of activity a_i value of an activity is at $\mathcal{O}(n^2 \times m)$ where n is the number of activities in each variant and m is the number of variants. [6] Compared to the $\mathcal{NP}$ level complexity of the precise ranking algorithm, efficiency of the approximation ranking algorithm is much better. However, we still have to validate the performance of the approximation algorithm, i.e., we must show how close it is to the real optimum (i.e., the ranking provided by the precise ranking algorithm). Regarding our running example, comparison results are summarized in Table 3.

Precise ranking algorithm	Activity	E	D	C	B	A
	$CI(a_i)$	0.5750	0.3750	0.3750	0.1750	0.0000
	Rank	1	2	2	4	5
Approximation ranking algorithm	Activity	E	D	C	B	A
	$CI_a(a_i)$	0.6641	0.7384	0.8678	0.9280	1.0000
	Rank	1	2	3	4	5

Table 3. Comparing the ranking results of the two algorithms

Table 3 shows the ranking results for our example from Fig. 2 using both precise ranking algorithm and the approximation ranking algorithm. More precisely, tt shows the change impact of each activity, as well as its ranking order. This simple comparison already indicate that that the performance of the approximation ranking algorithm is quite good. Here, it generates the same ranking order as the precise ranking algorithm does.

Of course, such a simple comparison is far from being sufficient. First, as the comparison is only based on one example, it is not allowed to draw general conclusion about the performance of the approximation algorithm. Clearly, it cannot NOT perfectly match the precise ranking algorithm, since the latter one has $\mathcal{NP}$ level complexity. While the approximation ranking algorithm only tries to achieve an approximation using a polynomial algorithm. A more systematic comparison is required to evaluate the real performance of the approximation ranking algorithm. In this context, we are also interested in those factors that influence the performance of the approximation ranking algorithm. In the following, we try to answer the following two questions:

[6] The complexity is computed based on the assumption that we have already had the order matrices for the reference model and the variants. Since the complexity for computing the order matrix from a process model is at $\mathcal{O}(n^2)$, the overall complexity for computing the ranking value of an activity is $\mathcal{O}(m \times n^2 + n)$ where n is the number of activities and m is the number of variants.

1. *How good does the approximation algorithm perform, i.e., how close are its ranking results in comparison to the precise ranking results?*
2. *What factors have influence on the performance of the approximation ranking algorithm, i.e., based on what conditions does it perform better or worse?*

6 Simulation

We use simulation to answer the above questions, i.e., by generating hundreds or thousands of examples, we are able to conclude statistically how good the performance of the algorithm is and to test which factors significantly influence it.

In order to analyze the influential factors, we require the dataset for simulation to be well structured and well understood, i.e., we need to know the features of the dataset which we are analyzing in order to determine which parameters are more important than others. So far, there are no such real-life data available for simulation. And even if this had been the case, such data would certainly not cover all the scenarios we want to examine. Therefore, we use automatically created datasets to run our simulations.

This section will describe how the dataset are generated. Since both the ranking algorithms require a reference process model and a collections of process variants derived from it, this section has been divided into three subsections:

1. We first describe an algorithm to randomly generate a reference model in Section 6.1.
2. Besides purely examining the performance of our algorithms, we are also interested in whether the performance of our algorithms can be influenced by some external parameters (like the size of the models or the similarity between the models, etc). Therefore, Section 6.2 describes which parameters will be considered when configuring the reference model into process variants.
3. Section 6.3 then describes how we adjust the different parameters in configuring the process variants.

In the following, we generate 36 groups of datasets using different values for the parameters we consider. For each of these groups, we generate 1 reference model and 100 process variants configured out of the reference model (i.e., we consider 3636 process models). For creating the data sets, we assume different scenarios. We compare the two ranking algorithms based on the ranking results we obtain from the 36 groups. In the following, we describe the scenarios we used to generate the datasets and in the next section (cf. Section 7), we will evaluated the ranking result of the different algorithms.

6.1 Generating the Reference Process Model

Our general idea of randomly generating (block structured) reference model is to cluster blocks, i.e., we randomly cluster activities (blocks) into a bigger block and

input : Set of activities a_i the process model to be generated should contain, $i = (i, \ldots, n)$
output: Valid process model S

1 Define each activity a_i as a basic block B_i, $i = (1, \ldots, n)$;
2 Define set $\mathcal{B} := \{B_1, \ldots, B_n\}$ `/* initial state */` ;
3 **while** $|\mathcal{B}| > 1$ **do**
4 randomly selected two blocks $B_i, B_j \in \mathcal{B}$;
5 randomly select an order relation $\tau \in \{0, 1, *, -\}$;
6 build block B_k which contains sub-blocks B_i and B_j having order relation τ ;
7 $\mathcal{B} := \mathcal{B} \setminus \{B_i, B_j\}$;
8 $\mathcal{B} := \mathcal{B} \bigcup \{B_k\}$;
9 **end**
10 $S := B_0$ with $B_0 \in \mathcal{B}$

Algorithm 1: Randomly generating a reference model

this clustering continues iteratively until all the activities (blocks) are clustered together. The detail of our approach is depicted in algorithm 1.

To illustrate how Algorithm 1 works, an example is given in Fig. 6. As input a set of activities { `A,B,C,E` and `E` } are given, and the goal is to construct a valid, block-structured process model S out of them. The algorithm starts by considering each activity a_i as basic block B_i, and adding these blocks to set $\mathcal{B}$ (lines 1 and 2). Regarding our example, $\mathcal{B} = \{\{\texttt{A}\}, \{\texttt{B}\}, \{\texttt{C}\}, \{\texttt{D}\}, \{\texttt{E}\}\}$. The algorithm first randomly select two blocks B_i, B_j (lines 4) and link them with a randomly chosen order relation τ (lines 5 and 6). Regarding our example, blocks {`B`} and {`C`} are selected to construct a new block {`B, C`} with a randomly chosen order relation 1 (which means `B` precedes `C`). The newly created block {`B, C`} will then replace blocks {`B`} and {`C`} in the block set $\mathcal{B}$, i.e., $\mathcal{B} = \{\{\texttt{A}\}, \{\texttt{B}, \texttt{C}\}, \{\texttt{D}\}, \{\texttt{E}\}\}$ (lines 7 and 8). This procedure (lines 4-8) is repeated until block set $\mathcal{B}$ contains one single block B_0 ($B_0 = \{\texttt{A,B,C,D,E}\}$ regarding our example). This block then represents our randomly generated process model S. (line 10). Fig. 6 shows the process model we randomly generated as well as the block constructed in each iteration.

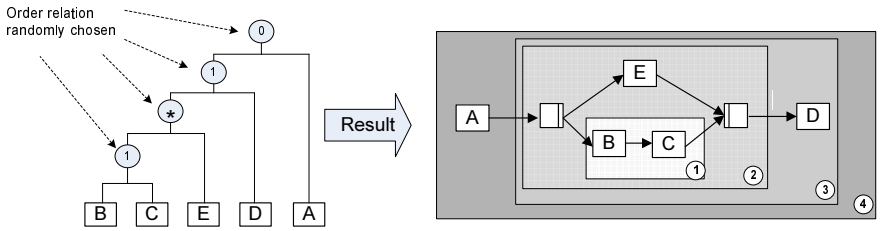


Fig. 6. Example of generating random process model

In practise, certain order relations are used more often than others. For example, the predecessor-successor relation is used more frequently than AND/XOR-splits [33]. When randomly generating a process model, we therefore take this into account as well. Rather than randomly setting the order relation for two blocks, we set the probability for choosing AND-split ($\tau =' *'$) and XOR-split ($\tau =' -'$) to 10% respectively, while predecessor-successor relationships ($\tau = \{0, 1\}$) are chosen with probability of 80%.

We therefore randomly generate 3 reference models, containing 10, 20 and 50 activity respectively [7]. According to [15], process models containing more than 50 activities have high risk of errors. therefore, it is not recommended to design such large model. Following this guideline, we also set the largest size of a process model for 50 activities in our simulation.

6.2 Parameters Considered for Generating Process Variants

Taking a generated reference process model, we control how variants are configured by adjusting specific parameters. For example, these parameters determine how many change operations needed to perform to configure a particular variant and where activities should be moved to and so forth. Basically, we have considered the following parameters when generating the process variants.

1. **Parameter 1 (Size of Process Models)** The size of a variant (i.e., the number of its activities) can potentially influence results. Therefore, we need to check the behavior of our algorithm when applying them to variants of different sizes. This is also important to test the scalability of the approximation algorithm, whose performance should not depend on the size of the sample data.
2. **(Parameter 2 (Similarity of Process Variants)** This parameter measures how "close" these variants are, e.g., whether or not the variants are similar to each other. In this context, similarity measures how difficult it is to configure one variant into another [12].
3. **Parameter 3 (The Activities Been Changed)** Often we can observe that the probability with which activities are changed is not uniformly distributed, i.e., some activities might be involved in changes more often than others. We therefore want to analyze whether the probability distribution of how frequently an activity is changed would thus influence the performance of our algorithms.
4. **Parameter 4 (Position Where Activities Are Moved To)** Change can be local or global. A local change will influence the order relations of only very few activities (e.g., when swapping the order of two directly succeeding activities) while global ones can influence quite a lot of order relations in a process model (for example when moving one activity from the beginning to the end). Therefore, we need to check whether or not this influences the results.

[7] Each of these model will be used in 12 groups of datasets, since we want to avoid the influence of the reference model. A detailed discussion can be found later in Section 6.3

6.3 Method for generating Data Sets

The dataset for our simulation analysis is generated by a randomly generating a reference model (cf. Section 6.1) and a collection of variants configured based on this reference model. When configuring these variants, by applying a sequences of change operations to the reference model, we vary the parameters described in Section 6.2 and consequently generate variants based on different scenarios. The following choices are available for the different parameters:

Parameter 1 (Size of Process Models) This parameter controls how many activities shall be contained in a process model. There can be three options:

1. **Small-sized** models : 10 activities per variant
2. **Medium-sized** models: 20 activities per variant
3. **Large-sized** models: 50 activities per variant

(Parameter 2 (Similarity of Process Variants) The closeness between the variants is measured by determining the total number of change operations we have to apply when generating variants (cf. Def. 2). Three possible choices exist:

1. **Small change**: 10% of activities are moved
2. **Medium change**: 20% of activities are moved
3. **Large change**: 30% of activities are moved

For example, for the datasets comprising large-size process variants (i.e., variants with 50 activities), medium-change would mean to randomly change 10 activities when generating generate a variant of the given a reference model. This way, we can control the distance between the reference model and its variant. And indirectly, we can control the similarity between variants since they are all controlled in a certain distances with the reference model.

Parameter 3 (The Activities Been Changed) This parameter controls which activities are moved when generating the variant models. We consider two scenarios:

1. In this scenario, we randomly pick the activities be moved, i.e., each activity is assumed to have the same probability to be involved in a move operation. As example assume that we need to move two activities in order to configure one particular variant out of the reference model. In this scenario, we assume each activity in the reference model would have the same probability to be chosen, i.e., we can randomly pick two different activities. Since every time activities are selected randomly, there will be no activity which has been moved significantly more often than others in the collection of variants. Please note that it does not mean every activity will be changed exactly for the same number of times when configuring the collection of process variants. Random differences will occur but these differences are not significant enough in a statistical sense. Table 4 shows one example for random picking activities with small model (10 activities) and small change (10% of them are changing, i.e., one change).

2. **Activities are selected based on Gaussian distribution**. Very often we can observe that some activities are involved in changes than others. We therefore have to simulate the situation in which activities are not been involved in change with same frequency. We set the probabilities for changing (i.e., moving) the activities by using a Gaussian distribution, i.e., some activities are assumed to be moved more often than others when generating the process variants. If n corresponds to the number of activities in the reference model and we randomly give a permutation of the activity set: the probability for selection the number n activity follows Gaussian distribution $\mathcal{X}\sim(n/2,(n/10)^2)$. This means the expected mean of the distribution is $n/2$ and its expected standard deviation is $n/10$. The Table 4 gives a general ideal of such distribution for a dataset with small size model (comprising 10 activities) and only small change (10% of them are changing) are performed to configure each process variant:

Activity	A	B	C	D	E	F	G	H	I	J
Number of times activity being involved in changes operations	Selected activity randomly									
	15	10	9	10	9	12	7	6	11	11
	Selected activity based on a Gaussian distribution									
	0	0	2	15	34	35	14	0	0	0

Table 4. When configuring 100 variants from the reference model, this table shows the number of times one particular activity being involved in change operations based on either random selection or Gaussian distribution

Parameter 4 (Position Where Activities Are Moved To). While parameter 3 determines how frequent an activity is moved, Parameter 4 controls the position to which corresponding activities are moved to. Clearly, a local change (e.g., to swap the order of two directly succeeding activities) has less effects on order relations than a global change (e.g., move an activity from the beginning to the end). This section analyzes whether this will influence the performance of our ranking algorithms.

When performing a move operation, one important issue has to be considered. Since we only consider block-structured process models, move operations must not "destroy" this block structure. Given the reference model S an the candidate activity a_i for moving, we perform the following three steps to guarantee block-structure of the resulting model:

1. We first remove activity a_i from the process model S.
2. We enumerate all possible blocks the modified process model contains. A block can be one single activity or a self-contained part of the process model or even the model itself. (See Appendix A for an algorithm enumerating all possible blocks in a process model). Also note that the number of possible candidate blocks is normally very large, e.g., several hundred potential blocks can be identified for a large process model (with 50 activities).

3. We cluster the candidate activity a_i randomly with one of the potential blocks we enumerated in step 2. The clustering technique will be similar to how we generate a random process model in Section 6.1. i.e., we randomly select an order relation $\tau \in \{0, 1, *, -\}$ as the order relation between activity a_i and the selected block. Therefore, they are clustered together to form a larger block.

Following these three steps, we can guarantee that the resulting process model is sound and block-structured. Every time we cluster an activity with a block, we actually move this activity to the position where it can form a bigger block together with the selected one. On example is given in Fig. 7:

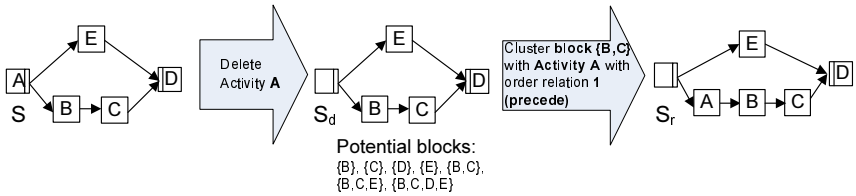


Fig. 7. Example of how to guarantee block structure when perform a move operation

Given a process model S, we would like to know where we can move saying activity `A` to, so that the resulting model maintains block-structured. The first step is to delete this particular activity (`A` in this case) from the process model and we obtain an intermediate process model S_d. Then, we enumerate all possible blocks such model contains. Regarding our example, S_d contains the following seven blocks $\{\{\texttt{B}\}, \{\texttt{C}\}, \{\texttt{D}\}, \{\texttt{E}\}, \{\texttt{B,C}\}, \{\texttt{B,C,E}\}, \{\texttt{B,C,D,E}\}\}$. We can then cluster activity `A` with any one of the blocks with order relation τ between the activity and the selected block, $\tau \in \{0, 1, *, -\}$. Regarding our example, block $\{\texttt{B,C}\}$ are chosen and it is clustered with `A` with order relation 1, i.e., `A` precedes block $\{\texttt{B,C}\}$.

If we know where one particular activity a_i can be moved to, we then need to determine whether it is a local change or a global change. Clearly, if a_i is clustered with a block close to its original position, it should be a local change. Otherwise, if a_i is clustered with a block far away from where it used to be, such move should be considered as a globe one. Therefore, whether a change is a local change or a global change is determined by the distance the moved activity and the block which it will be clustered with.

In general, the distance between an activity and a block is measured by the differences between them and the remaining activities. A detailed discussion can be found in Appendix B. Regarding our example in Fig. 7, the distances between activity `A` and all possible blocks it can be clustered with are summarized in Table 5.

Parameter 4 therefore can control whether we perform more local changes or more globe changes. We consider two options for determining such target block.

1. **Random block selection** In this scenario, we randomly select a block from the list of all possible blocks, i.e., each potential block has the same probability to be chosen. This way, block selection is not controlled and both local and global changes are possible. Regarding our example in Fig. 7, the probability for selecting all 7 potential blocks equals $1/7 = 0.143$ (cf. Table 5).
2. **Selection of block that is close to the activity to be moved** Blocks with shorter distance to the activity to moved have higher probability to be chosen. In order to realize this, we first rank all blocks based on their distance to the activity to be moved, and the probability to choose the i_{th} block is $2 \cdot \frac{1}{\sqrt{2\pi}} e^{-(\frac{5i/m}{2})^2}$, where m equals the number of blocks (this curve has similar shape as Gaussian distribution $\mathcal{X} \sim (0, 1^2)$ within the interval of [0,5]). Regarding our example in Fig. 7, the probability for choosing each block is also summarized in Table 5. Clearly, the probability for choosing a closer block is a lot higher than the probability for choosing a block which has higher distance. Therefore, following this scenario we are expected to perform more local change compared to the other scenario.

Blocks	{B}	{B,C}	{B,C,E}	{C}	{E}	{B,C,D,E}	{D}
Distance to Activity A	2.45	3.10	3.31	3.74	3.74	3.90	5.65
Rank	2.45	3.10	3.31	3.74	3.74	3.90	5.65
The probability of this block being selected	Select block randomly						
	0.143	0.143	0.143	0.143	0.143	0.143	0.143
	Select block based on a Gaussian distribution						
	0.618	0.288	0.08	0.013	0.001	8.19E-5	2.97E-6

Table 5. When selecting a block to be clustered with A, this table shows the probability of selecting each candidate block based on either random selection or Gaussian distribution

6.4 Simulation Setup

Section 6.3 has discussed four parameters we considered. Regarding parameter 1 and 2, we can choose between 3 values, while parameter 3 and parameter 4 respectively allow for setting two values, i.e., in total we have $3 \times 3 \times 2 \times 2 = 36$ possible configurations of these four parameters and consequently should consider 36 groups of datasets. These 4 parameters, and consequently should consist For example, one particular group may prescribe *10 activities* for a variant, where each variant is generated by moving *10% of activities* from the reference model. When performing the move operations, the activities are selected based on the *Gaussian distribution* whereas the block determining the position to which we move these activities to is *randomly* selected.

We document the following information when generating the datasets:

1. **Original reference model**, i.e., the model based on which we perform the changes. We have sketched a method to randomly generate a reference model in Section 6.1. In addition, to make results comparable, we use same reference model for all groups with same number of activities. For example, all groups with parameter 1 = "small size" use same reference model containing 10 activities.
2. **100 process variants**. Based on a given reference model, we generate each variant by configuring the reference model according to the different scenarios as described in Section 6.3. For each group, we generate 100 process variants. Note that although the 100 variants are generated by following a same scenario, they are not necessarily same. The reason is that the scenario we described In Section 6.3 only depicted the feature of the collection of variants but not a particular variant. For example take Table 4. When following the scenario to select activity randomly, i.e., each activity has the *same probability* to be chosen when configuring a process variant, it is actually not the case that each activity has been chosen for exactly a same amount of times.
3. **Ranking Results for Each Group** Based on the reference process model and the 100 process variants, we can rank the activities based on the precise ranking algorithm (cf. Section 4) as well as the approximation ranking algorithm (cf. Section 5). Table 6 shows an example. In this group, each variant contains *10 activities*, and it is generated by moving *10% of the activities* in the reference model. When performing the changes, activities and blocks are *randomly* selected. [8]

Precise ranking result

Activity	A	F	I	B	D	J	E	C	G	H
$CI(a_i)$	0.1450	0.1250	0.1100	0.1000	0.0999	0.0999	0.0900	0.0800	0.0700	0.0699
Rank	1	2	3	4	5	5	7	8	9	10

Approximation ranking result

Activity	A	F	J	I	D	G	E	B	C	H
$CI_a(a_i)$	0.9787	.9792	0.9903	0.9904	0.9908	0.9911	0.991726	0.991728	0.9921	0.9923
Rank	1	2	3	4	5	6	7	8	9	10

Table 6. Precise and approximation ranking result

From Table 6, it becomes clear that the precise ranking algorithm and the approximation ranking algorithm does not always provide same ranking results. For example, Activity `I` is ranked third regarding the precise ranking result, but ranked fourth regarding the approximation ranking result. In the next section, we provide a method to evaluate the differences between the two ranking algorithm.

[8] All datasets and ranking results are available at:
http : *//wwwhome.cs.utwente.nl/ lic/Resources.html*

7 Evaluation

In the following, we analyze the ranking results provided by our two algorithms by means of simulation. This section consists of four parts:

1. Pre-processing the ranking results, i.e., tie-breaking (cf. Section 7.1).
2. Defining the approach for evaluating the ranking results (cf. Section 7.2)
3. Analyzing the performance of the approximation ranking algorithm (cf. Section 7.3)
4. Analyzing the parameters that have significant influence on the ranking results (cf. Section 7.4)

7.1 Pre-process of the ranking result

When examining the precision ranking list in Table 6, we can see activities `D` and `J` have the same change impact and consequently have the same rank. This triggers a question about how to handle the case in which several activities have the same rank, i.e., how to break the tie.

Most algorithms handle tie-breaking randomly, i.e., they enforce a random order of the activities with same rank. Unfortunately, this method does not work in our context. Consider a situation in which the precise ranking algorithm assigns to all activities the same change impact, i.e., all activities have same rank. If we enforce a random order in connection with the precise ranking algorithm, the matching between the precise ranking and the approximation ranking algorithms would be fully dependent on such random order. If the random order is "luckily" the same as in the approximation algorithm, the match will be perfect. However, if the random order is "unfortunately" generated differently, the opposite will be the case.

To handle this tie-breaking problem, we enforce an order for activities with same rank by additionally considering the order of the other ranking algorithm. As example, consider the case in Table 6. As activity `D` and `J` have the same rank in the precise ranking algorithm, we sort `D` and `J` by considering their rank in the approximation ranking algorithm. As in the approximation ranking algorithm `J` precedes `D` in the rank, we also put `J` before `D` when breaking the tie. In this case, the precise ranking result will change, see Table 7.

Clearly, enforcing an order this way also bears the risk that we artificially make the two ranks provided by the two algorithms more similar with each other. To have a clear understanding of this problem, we also document the number of all possible solutions to break the tie whenever we perform a tie-breaking. If n activities have same change impact value, the number of possible solutions to break such tie equals the total number of sequences these n activities can make (their permutation). And we can compute it by $P_n^n = n!$. Regarding our example from Table 7, two activities `J` and `D` have the same impact factor, therefore we have $P_2^2 = 2! = 2$ solutions, i.e., either `JD` or `DJ`. This total number of possible solutions p therefore can represent the risk we bear when break the tie using the above mentioned way. The reason is that our tie-breaking approach only take 1

Precise ranking result										
Activity	A	F	I	B	**J**	**D**	E	C	G	H
Change impact	0.1450	0.1250	0.1100	0.1000	0.0999	0.0999	0.0900	0.0800	0.0700	0.0699
Rank	1	2	3	4	5	5	7	8	9	10
Approximation ranking result										
Activity	A	F	**J**	I	**D**	G	E	B	C	H
Change impact	0.9787	.9792	0.9903	0.9904	0.9908	0.9911	0.991726	0.991728	0.9921	0.9923
Rank	1	2	3	4	5	6	7	8	9	10

Table 7. Precise ranking algorithm after resort according to the approximation ranking algorithm

particular sequence from the p possible solutions. Therefore, the higher p is, the higher the chance we will artificially make the two ranking results more similar. The p values for all groups are shown in Appendix C.

In Appendix C, the p value for approximation ranking algorithms all equals 1 in all groups of our dataset. This means that no activities have equal change impact values provided by the approximation raking algorithm and therefore no tie-breaking is performed to pre-process the approximation rankings. However, when checking the precise ranking result, we can see some high p values. It means that we have performed tie-breaking for some activities in the ranking. However, these p values are not large enough to threaten the validity of our tie-breaking strategy. Reason is that they are still considerably low compared with the total number of possible sequences these activities can make, i.e., the ranking algorithm is not based on a random sequence of activities.

7.2 Evaluation Approach

In this section, we evaluate the performance of the approximation ranking algorithm, i.e., we measure how close our approximation is to the "real" optimum (i.e., the precise ranking).

Precision is a widely used notion for measuring the performance of ranking algorithms in different domains [23, 1]. In our context, we can consider the precise ranking algorithm provides the ranking order we want to have, while the approximation ranking result is the one we actually get. As the activities are ranked differently, the sub-sets of the top n ranked activities are necessarily same. For example, let us compare the top three activities as ranked in the two ranking algorithms (cf. Table6). While the precise ranking list contains activities `A, F` and `I`, the approximation ranking list comprises activities `A,F` and `J`. Consequently, if we need to pick the top 3 activities, the two ranking algorithms will provide different activity sets. The difference between the top n activities in the two ranking list can be measured using $Precision(n)$:

Definition 7 (Precision(n)). *Let $P(n)$ be the set containing the top n ranked activities provided by the precise ranking algorithm. Let further $A(n)$ be the set*

containing the top n ranked activities as provided by the approximation ranking algorithm. We define ***Precision(n)*** *as follows:*

$$Precision(n) = \frac{|P(n) \bigcap A(n)|}{|P(n)|}$$

$precision(n)$ reflects the ratio of the top n ranked activities shared in the result lists of both the precise and approximation ranking algorithm. Def. 7 uses the precise ranking result as reference list, i.e., the optimum to which we want to be close. The approximation ranking result in turn, corresponds to the information we actually get. Consequently, $precision(n)$ measures how much "useful information" about the actual top n activities we can get from the application of approximation algorithm. As example, consider Table 6, when comparing the top 3 activities of the precision ranking result with those of the approximation ranking result, we can see that activities `A` and `F` have been correctively selected, whereas this does not apply to activity `I` (i.e., `I` is not contained in the top 3 activities suggested by the precise ranking algorithm). Therefore, $precision(3) = 2/3 = 0.6667$. Table 8 shows all precision values concerning the top n ranked activities.

Precise ranking result										
Activity	A	F	I	B	J	D	E	C	G	H
Weight	0.1450	0.1250	0.1100	0.1000	0.0999	0.0999	0.0900	0.0800	0.0700	0.0699
Rank	1	2	3	4	5	6	7	8	9	10
Approximation ranking result										
Activity	A	F	J	I	D	G	E	B	C	H
Weight	0.9787	.9792	0.9903	0.9904	0.9908	0.9911	0.991726	0.991728	0.9921	0.9923
Rank	1	2	3	4	5	6	7	8	9	10
precision(n) for top n activities										
top n activity	1	2	3	4	5	6	7	8	9	10
$precision(n)$	1.0000	1.0000	0.6667	0.7500	0.8000	0.8333	0.8571	0.8750	1.000	1.000

Table 8. precision table

Here $precision(1)$ and $precision(2)$ equal 1. This means that the top ranked n activities ($n = (1, 2)$) are the same for the precise ranking list and the approximation ranking list. However, as the approximation algorithm ranks activity `J` at the third place, we see a decrease on $precision(3)$ since `I` should be ranked third as done in the case of the precise ranking list. Then, $Precision(n)$ keeps increasing as n increases until it reaches 1 again at the end of the chart ($n = 9, 10$). Trivially, $precision(n)$ will always be 1 if n equals the number of activities. We have additionally plotted the precision values in Fig. 8.

We can derive the curve depicted by plotting and interpolating all precision values in Fig. 8. Besides this precision curve, we plot a line $precision(n) = 1$, $n = (1, 2, \ldots, 10)$. Further more we have marked the surface area between the two

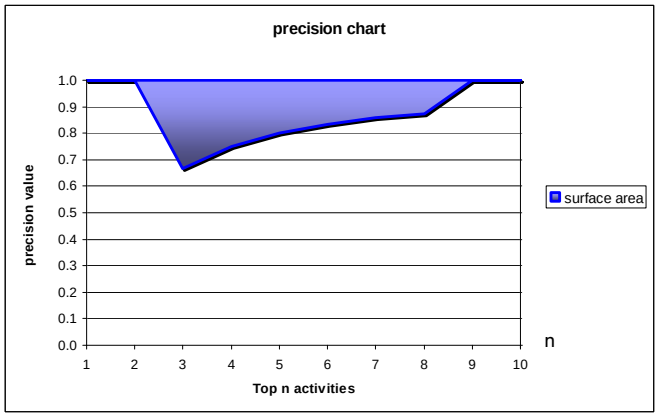


Fig. 8. Surface area for the precision chart

curves. The size of this surface area then can be used to evaluate the performance of the approximation ranking algorithm for the given dataset. Trivially, if the approximation ranking algorithm delivers the same ranking result as the precise ranking algorithm, the precision value for any top ranked n activities will equal 1, i.e., the precision curve matches the optimum curve $precision(n) = 1$ with $n = 1, 2, \ldots, 10$. If the precise and approximation ranking algorithms provide different ranking results, there will be a number n' the precision value $precision(n')$ does not equal to 1. In this case, the precision curve deviates from the optimum curve, i.e., there is space between the two curves. The larger this area is, the bigger the difference of the two ranking results will be, and the worse the approximation algorithm actually performs. Regarding our example, the surface area between the two curves equals 1.12. To make the surface are independent from the number of activities and to normalize the value within interval [0,1], we measure the ratio between the surface area and the area created by the optimal line (the rectangle between (0,0) and (1,10) in our case). This value is 0.112 in our example. This number can be used as indicator showing how close the approximation line is to the real optimum line, i.e., showing how good the approximation algorithm works. Obviously, the lower the value is, the better the approximation algorithm works. [9] Altogether, the proposed method (i.e., measuring the ratio between the precision line and the optimum line), is used to evaluate the performance of our approximation algorithm in the conducted simulation which consists of 36 groups of datasets.

[9] This evaluation method is inspired by the precision-recall curve used in information retrieval [1] or statistics [22]. We omitted "recall" since in our context, it always equals n/m for the top n ranked activities in a rank list of size m. We do not apply correlation analysis in statistics [22] since we are interested in the ranking order rather than the exact change impact of activities.

7.3 Evaluation Result

In Appendix C, we give a short summary of the evaluation results for all groups. For each group, we indicate the influence of tie-breaking policy as described in Section 7.1. We also show the surface area (precision) introduced in Section 7.2. A complete list, including the weights the ranking orders, etc, is available at: http://wwwhome.cs.utwente.nl/~lic/Resources.html.

In the next two sub-sections we visualize the results in two different perspectives.

Surface Area Values Distributions We first analyze the distributions of the surface area values at different value ranges. A standard method is to depict the histogram [22]. A histogram shows the distribution of surface areas into different intervals. The result is shown in Fig. 9. The value range of the surface area is [0, 0.4] in all 36 groups. The histogram shows eight sub-intervals with an increment of 0.05 in each interval. From Fig. 9, it becomes clear that in most groups, i.e.,10 out of 36 group, the surface area falls into interval [0.15,0.2). If we enlarge this interval to [0.1, 0.25), more than 60% of the groups are covered. When computing the mean and standard deviation of the surface areas, we obtain as mean 0.1933 and as standard deviation 0.0871.

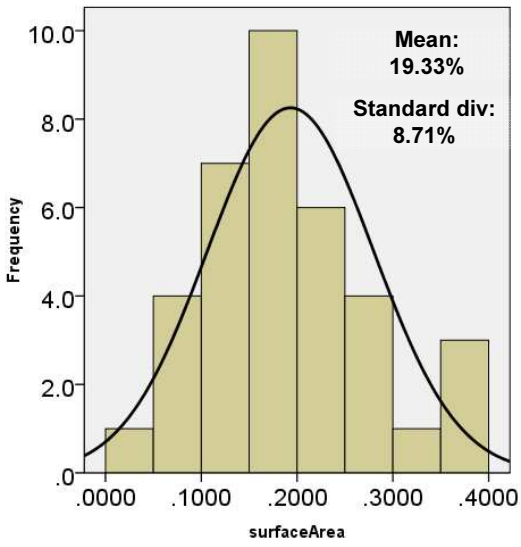


Fig. 9. Histogram of surface area value

To have a better understanding of the distribution of the surface areas, we also tested whether the surface area follows a Gaussian distribution. Here, we apply One-Sample $Kolmogorov - Smirnov$ Test [22] to test our assumption.

The $Kolmogorov - Smirnov\ Z$ value is 0.555, which indicates that the data have 91.8% confidence following a Gaussian distribution. Clearly, the expected mean is 0.1933 while the standard deviation is 0.0871. The distribution curve is depicted in Fig. 9.

In addition, We also tested the confidence interval [22] of this surface area since it is an important factor to measure the performance of the algorithm. The 95% confidence interval is [0.1637,0.2225]. This indicates that the mean of surface area has 95% probability falling into the interval [0.1637,0.2225].

Average Precision Values at Top n Ranked Activities In the former subsection, we have analyzed the distributions of surface area values in different groups. This analysis shows the general performance of the approximation ranking algorithm, and indicates that the average imprecision, measured by the surface area, is around 20%. However, in most use cases we are mainly interested in the activities at the top of the ranking list. A typical question is for example, how the approximation algorithm will perform if we only compare the precision of the top 10% ranked activities? In the following, we evaluate the precision of our approximation algorithm at different positions of the ranking order.

Based on the 36 dataset groups, we evaluate the precision value for top $n\%(n = 10, 20, \ldots, 100)$ ranked activities of the ranking list. The mean as well as the standard deviation of the precision values are depicted in Fig. 10.

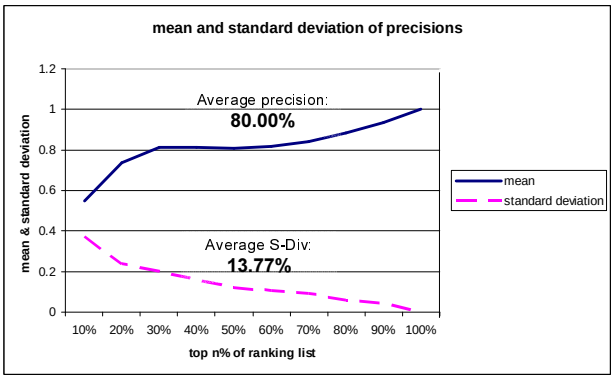


Fig. 10. mean and standard deviation of precision values at top n% of ranking list

The average precision value starts at 55% with standard deviation of 37.53% when only evaluating the top 10% of the ranking list. The average precision values keep increasing as we enlarge the ranking list we compare. This value becomes stable after evaluating the top 30% of the ranking list. Without any surprise, the value ends with 100% at the end of the curve.

Our simulation results show that the approximation algorithm performs relatively poor when only considering the first activities of the ranking list, but

quickly becomes stable when we enlarge the ranking list we analyze. Note that even 55% of precision at the beginning of the ranking list is considered being acceptable in the field of domain-specific information retrieval, where the precision values are quite low and vary largely [1].

In conclusion, we can now answer the first research question formulated in Section 5.4 about the performance of the approximation ranking algorithm. The average surface area, which measures the imprecision of the approximation ranking algorithm is 19.33% and with a probability of 95% falls into the interval [0.1637,0.2225]. The precision of the approximation ranking algorithm is 55% when only measuring the top 10% activities. This value increases as the rank list increases, and will becomes stable after considering more than 30% of the top ranked activities.

7.4 Significance test

In this section, we analyze which of the four parameters presented in Section 6.2 have significant influence on the surface area, i.e., which parameters do significantly influence the performance of the approximation ranking algorithm.

We assume that all parameters are independent from each other. In the following, we analyze the situation in which one parameter is varying while the other parameters remain constant. For example, assume we want to measure the influence of Parameter 2 (similarity between variants). Then we need to divide our 36 groups of datasets into 3 sub-groups: one group with parameter 2 = "small", another one with Parameter 2 = "medium", a third one with Parameter 2 = "large". We then analyze whether or not the results from the three groups are significantly different from each other. If so, Parameter 2 has significant influence on the results of the approximation ranking algorithm.

Therefore, for each of the four parameters $n(n = 1, 2, 3, 4)$, we want to test the following null hypothesis:

H_0^n: Parameter n has no influence on the surface area.

If the hypothesis is tested to be statistically significant (i.e., probability is larger than 5%), we will accept the hypothesis, i.e. Parameter n is then assumed to have no influence on the performance of the approximation algorithm. If this is not the case, we need to reject this hypothesis, i.e., the parameter then significantly influences the performance of the approximation ranking algorithm. Comparable to most of the hypothesis tests, we assume that errors are independent and follow normal distribution [22, 9].

Regarding our example, Parameter 1 and 2 have three options to choose while parameter 3 and 4 have two (cf. Section 6.2). We will first explain how to measure parameter 3 and 4 and then parameter 1 and 2.

Parameter 3 and 4 We apply t test to evaluate the statistical significance of hypothesis H_0 [22] for Parameter 3 and 4. The significance test attempts to disprove this hypothesis by determining a probability value (p-value), i.e., which measures the probability that the observed difference could have occurred by

chance. If the p-value is less than a given threshold value α (which is often set to 0.05), we can reject H_0 and conclude that the tested parameter has significant influence on the surface area.

We first examine influence of Parameter 3. For this purpose, we divide our dataset into 2 sets X and Y: X contains all the groups with Parameter 3 = "Random", while Y comprises groups with Parameter 4 = "Gaussian". Since we have totally 36 groups of datasets, we have $|X| = |Y| = 36/2 = 18$. Let $X_i \in X$ and $Y_i \in Y$ be the surface area values of two groups with parameter 3 varying while the remaining three parameters have the same values, $i = 1, \ldots, n$ and $n = 18$. We define $D_i = Y_i - X_i$ for $i = 1, \ldots, n$, and $\overline{D}$ is the average value of $D_i (i = 1, \ldots, n)$. Assume that the model is additive, i.e., the observed difference D_i can be measured by the group different θ plus an independent error ϵ_i ($D_i = \theta + \epsilon_i$). Our null hypothesis H_0 is then $\theta = 0$, which means the two groups are the same. The mathematical details for the t-test are given below.

t-test

$$t = \frac{\overline{D}}{s(D_i)/\sqrt{n}} \qquad \text{with}$$

$$\overline{D} = \frac{1}{n}\sum_{i=1}^{n} D_i \qquad \text{and} \qquad s(D_i) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(D_i - \overline{D})^2}$$

Distribution under H_0 should follow Student's t with $n-1$ degrees of freedom [22].

Using the same approach we can test Parameter 4; results are shown in Table 9.

Parameter	Mean diff.	Standard dev.	t value	Probability	Significant?	has influence?
Parameter 3	-0.1043	0.0802	-5.5206	3.70E-5	NO	YES
Parameter 4	0.0776	0.0799	4.1225	7.15E-4	NO	YES

Table 9. Significance test result for Parameter 3 and 4

For Parameter 3, the result shows that groups in Y (groups with Parameter 3 = "Gaussian") perform better than groups in X (groups with Parameter 3 = "Random"), since the mean difference is a negative value. The significance test indicates very low probability that such difference is generated by random errors (probability = 3.70E-5). Because of this, we need to reject H_0, i.e., Parameter 3 can significantly influence the performance of the approximation ranking algorithm. And the more changes are performed based on few activities, the better the approximation ranking algorithm performs.

Similar analysis can be made for Parameter 4. Regarding Table 9, we can see that groups in X are better than groups in Y, since the mean difference is a positive value. The two groups are also significantly different to each other since the probability value is very low (7.15E-4). We can conclude that Parameter 4

is also can significantly influence the performance of the approximation ranking algorithm: the more global change we apply, the better the approximation algorithm performs.

Since parameter 3 and 4 have significant influence on the performance of the approximation algorithm, we further analyze the groups of datasets partitioned by the values of parameter 3 and 4. We first divided the 36 groups of dataset into two subsets, one with Parameter 3 being set to "Random" and another with Parameter 3 set to "Gaussian". Similar to the approach described in Fig. 10, we analyze the mean and standard deviations of the precision values at top n% ranked activities in the two sub-groups respectively. Similar analysis can be applied to parameter 4, i.e., we divide the datasets into two subsets based on the value of Parameter 4, and analyze the mean and standard deviation of the top n% precision values. Results are depicted in Fig. 11.

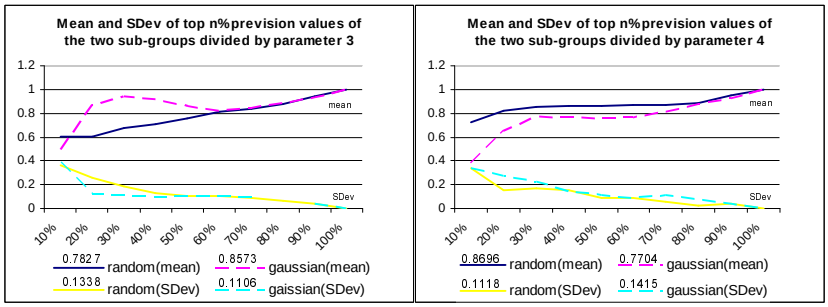


Fig. 11. Comparing the mean and standard deviation of the top n% precision values in different groups

We plot the mean and standard deviation of the subgroups in the same graph. It then becomes clear that for Parameter 3, the group with value "Gaussian" (dashed line) performs better than the one with value "Random". Reason is that average precisions are higher while standard deviations are lower. For Parameter 4, the group with value "Random" then performs better than the group with value "Gaussian" for the same reasons as described above. By plotting the mean and standard deviation of the precision values for the top n% activities, we can derive the same result as we obtained through the significance test.

Parameter 1 and 2 Since Parameter 1 and 2 both have three values to choose ("small", "medium", and "large"), we cannot use the t-test approach described above. The standard approach to this type of problem is to examine the data using a two-way Analysis of Variance (ANOVA) [22, 9].

We first analyze Parameter 1. Let us divide the dataset into sub-sets $Y_j, j = (1, \ldots, m)$ based on the value of Parameter 1. Let $y_{ij}, i = (1, \ldots, n)$ be the surface area of a group in set Y_j. Such a group in Y_j should have same index i

as a group in Y'_j in case parameter 2, 3 and 4 all have the same value for these two groups. Regarding our example, since Parameter 1 can have three values ("Small", "Medium" and "Large"), m is then 3 and since $n = |Y_j|$ then n is $36/3 = 12$. Let $\overline{y}$ be the average surface area of all groups while $\overline{y}_j$ represent the average surface area of the groups in Y_j. The standard model is as follows:

$$y_{ij} = \mu + \alpha_i + \beta_j + \epsilon_{ij}$$

This means that each observation y_{ij} can be broken down into the true mean performance μ, the group effect α_i, the influence of parameter β_j, and the error ϵ_{ij}. We assume α and β to be independent and additive. Two-way ANOVA can be computed as follows:

$$F_A = \frac{MSR}{MSE} = \frac{\frac{n\sum_j(\overline{y_j}-\overline{y})^2}{m-1}}{\frac{\sum_{i,j}(y_{ij}-\overline{y_i}-\overline{y_j}+\overline{y})^2}{(n-1)(m-1)}}$$

MSR corresponds to the mean-squared difference between groups Y_j and MSE is the mean-squared error. We can test our hypothesis for $\beta_j = 0$, $j = 1, \ldots, m$. The assumption is that errors are independent and normally distributed. The probability of accepting our hypothesis H_0 follows F distribution with $n-1$ and $(n-1)(m-1)$ degrees of freedom. The test result is given in Table 10.

Parameter	F value	Probability	Significant?	has influence?
Para 1	1.3665	0.2758	YES	NO
Para 2	0.6545	0.5295	YES	NO

Table 10. Significance test result for parameter 2 and 3

For Parameter 1, the probability of H_0^1 being true is 0.2758 and therefore significant. This means that we can accept the hypothesis that Parameter 1 does not influence the size of the surface area. To be more precise, the number of activities in each variant **does NOT influence** the performance of the approximation algorithm. Similarly, Parameter 2 also has no influence on the size of the surface area, i.e., whether the variants are similar or not also **does NOT influence** the performance of the approximation algorithm.

The statistical results of Parameter 1 also indicate that the performance of our approximation algorithm is able to scale up, since the goodness of our approximation algorithm performs does not depend on the size of the process models.

8 Related work

Ranking techniques have been wildly used in fields like information retrieval [1] or data mining [23]. In information retrieval, for example, a query results in a list

of web sites or documentations, which are ranked according to the relevance of the searched object. In [13], the authors also provide a technique to retrieve similar process models using a query process model. However, this research focuses on retrieving relevant information rather than analyzing the structure of process models. In the workflow field, conformance checking techniques are widely used to measure the match between the designed process model and its actual execution behavior [21]. Such technique has also been applied in certain process mining approaches like genetic mining [4]. [6] also represented a process mining technique by discovering a collection of process variants. However, a prerequisite of this approach is a valid change log which documenting all the changes when configuring the reference process model to each variants. Clearly, such change log is not always available in our context. Similar techniques for conformance checking have been applied in process monitoring where people focus on handling exceptional situations and measuring fulfillment of business rules. [5, 32]. In the field of web services, service monitoring techniques are also used to monitor the behavior of the agreed service compositions. Violations of these agreement can be identified and also be punished [2]. Similar techniques also apply in business IT alignment measures [24] or security checks [30], where un-predefined business rules or security protocols are automatically identified and measured. However, most of the above mentioned approaches analyze behavior in-consistencies to measure the matching between the designed model and real executions. This behavior is different than the structural change on which we focused in this paper (see [11] for a detailed comparison). Also, few of the above mentioned approach are able to provide a detailed analysis of every individual process activity based on the observed in-consistencies.

9 Summary and Outlook

The key contribution of this paper is to provide both a precise ranking algorithm and an approximation ranking algorithm to rank the activities according to the degrees with which they are potentially involved in reconfiguration a given reference process model. Using these techniques, we are able to identify which activities have been configured more often than others. Such information is valuable for identifying optimization for the currently used (reference) process model when understanding the reference model or when re-engineering process models. It can also be used in process monitoring to identify which parts of a process (i.e., composite service) has been adapted more often than others during run time execution.

The precise ranking algorithm is precise but also time-consuming. Therefore, we introduced the approximation ranking algorithm, which can be computed in polynomial time, and evaluated it by performing an simulation. After having analyzed about 3600 process models, we demonstrated the performance of the approximation ranking algorithm and identified several parameters which can significantly influence the performance of the approximation ranking algorithm. The overall preciseness of the approximation ranking algorithm is around 80%

and the performance is not dependent on the size or the similarities between process variants. This result indicates that the approximation ranking algorithm is able to scale up since the performance is not dependent on the size of process model we analyze.

The proposed approach allows to identify the activities which are the problem makers, i.e., which are potentially responsible for change changes when configuring the process variants. Our next step is to also make use of the suggested technique for process variant mining [10]. In process variant mining, we can discover a reference process model by mining a collection of process variants. However, such technique does not consider the original reference model and therefore could generate a spaghetti-like structure which is too different from the original reference model. This is surely not preferred since re-engineering the reference model from the old version to the newly discovered one would be too complex and costly. The ranking algorithm discussed in this paper provides an opportunity to take the original reference model into consideration. This way, we can only focus on the highly ranked activities and the trivial configurations will not be considered in the end result.

References

1. H.M. Blanken, A.P. de Vries, H.E. Blok, and L. Feng. *Multimedia Retrieval*. Springer, 2007.
2. L. Bodenstaff, A. Wombacher, M. Reichert, and M. C. Jaeger. Monitoring dependencies for SLAs: The mode4sla approach. In *IEEE SCC (1)*, pages 21–29, 2008.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.
4. A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, NL, 2006.
5. D. Grigori, F. Casati, U. Dayal, and M. Shan. Improving business process quality through exception understanding, prediction, and prevention. In *VLDB '01*, pages 159–168, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
6. C.W. Günther, S. Rinderle-Ma, M. Reichert, W.M.P. van der Aalst, and J. Recker. Using process mining to learn from process changes in evolutionary systems. *Int'l Journal of Business Process Integration and Management*, 3(1):61–78, 2008.
7. T. H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Trans. Knowl. Data Eng.*, 15(4):784–796, 2003.
8. J. Hidders, M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede, and J. Verelst. When are two workflows the same? In *CATS '05:*, pages 3–11, Darlinghurst, Australia, 2005.
9. David Hull. Using statistical testing in the evaluation of retrieval experiments. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 329–338, New York, NY, USA, 1993. ACM.
10. C. Li, M. Reichert, and A. Wombacher. Discovering reference process models by mining process variants. In *ICWS'08*, pages 45–53. IEEE Computer Society, 2008.
11. C. Li, M. Reichert, and A. Wombacher. Mining process variants: Goals and issues. In *IEEE SCC (2)*, pages 573–576. IEEE Computer Society, 2008.

12. C. Li, M. Reichert, and A. Wombacher. On measuring process model similarity based on high-level change operations. In *ER '08*, pages 248–262. Springer LNCS 5231, 2008.
13. R. Lu and S. W. Sadiq. On the discovery of preferred work practice through business process variants. In *ER*, pages 165–180. Springer, 2007.
14. R. Lu and S.W. Sadiq. Managing process variants as an information resource. In *BPM'06*, pages 426–431, 2006.
15. Jan Mendling, Gustaf Neumann, and Wil M. P. van der Aalst. Understanding the occurrence of errors in process models based on metrics. In *CoopIS'07, LNCS 4803*, pages 113–130, 2007.
16. B. Mutschler, M. Reichert, and J. Bumiller. Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors and implications. *IEEE Transactions on Systems, Man, and Cybernetics (Part C)*, 38(3):280–291, 2008.
17. M. Reichert and P. Dadam. ADEPTflex -supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
18. S. Rinderle, M. Jurisch, and M. Reichert. On deriving net change information from change logs - the deltalayer-algorithm. In *BTW'07*, pages 364–381, 2007.
19. S. Rinderle, M. Reichert, M. Jurisch, and U. Kreher. On representing, purging, and utilizing change logs in process management systems. In *BPM'06*, pages 241–256. LNCS 4102, Springer, 2006.
20. M. Rosemann and W.M.P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, 2007.
21. A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008.
22. D.J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, 2004.
23. P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.
24. W. M. P. van der Aalst. Business alignment: using process mining as a tool for delta analysis and conformance testing. *Requir. Eng.*, 10(3):198–211, 2005.
25. W.M.P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203, 2002.
26. W.M.P. van der Aalst and T.Basten. Identifying commonalities and differences in object life cycles using behavioral inheritance. In *ICATPN '01*, pages 32–52, London, UK, 2001. Springer-Verlag.
27. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003.
28. J. Vanhatalo, H. Volzer, and J. Koehler. The refined process structure tree. In *BPM'08*, pages 100–115. Springer, 2008.
29. B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, 66(3):438–466, 2008.
30. B. Weber, M. Reichert, W. Wild, and S. Rinderle. Balancing flexibility and security in adaptive process management systems. In *CooplS '05*, pages 59–76, 2005.
31. M. Weske. *Business Process Management*. Springer, 2007.
32. M. zur Muehlen and M. Rosemann. Workflow-based process monitoring and controlling - technical and organizational issues. In *HICSS '00*, page 6032, Washington, DC, USA, 2000. IEEE Computer Society.

33. Michael zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *CAiSE'08*, pages 465–479. LNCS 5074, Springer, 2008.

A Block-enumerating algorithm

Let $S = (N, E, \ldots) \in \mathcal{P}$ be a process model with $N = a_1, \ldots, a_n$. Let further $A_{|N|\times|N|}$ be the order matrix of S. Two activities a_i and a_j can form a block if and only if $\forall a_k \in N \setminus a_i, a_j : A_{ik} = A_{jk}$, i.e., two activities can form a block if and only if then shows exactly same order relations to the rest of activities. We can describe the block detecting algorithm as follows:

input : A process model S and its order matrix A
output: A set BS with all possible blocks

1 Define BS_x be a set of blocks containing blocks with x activities. $x = (1, \ldots, n)$;
2 Define each activity a_i as a block B, $i = (1, \ldots, n)$;
3 Put all these blocks B in BS_1 `/* initial state */` ;
4 **for** $i = 2$ **to** n `/* Compute` BS_n `*/;`
5 **do**
6 | let j = 1; let k = i;
7 | **while** $j < k$ **do**
8 | | k = i - j;
9 | | **forall** *(*$\forall B_j \in BS_j$*) and (*$\forall B_k \in BS_k$*);* `/* judge whether` B_j `and` B_k `can form a block */;`
10 | | **do**
11 | | | **if** $B_j \bigcap B_k = \emptyset$ **then**
12 | | | | **if** *((*$\forall a_\alpha \in B_j$*) and (*$\forall a_\beta \in B_k$*) and (*$\forall a_\gamma \in N \setminus B_j \bigcup B_k$*)) ;*
13 | | | | *satisfies* $A_{\alpha\gamma} = A_{\beta\gamma}$ **then**
14 | | | | | Define $B_i = B_j \bigcup B_k$;
15 | | | | | Put B_i in BS_i;
16 | | | | **end**
17 | | | **end**
18 | | **end**
19 | | j ++ ;
20 | **end**
21 **end**
22 $BS = \bigcup_{x\in(1,\ldots,n)} BS_x$

Algorithm 2: Block enumerating algorithm

In the block enumerating algorithm, the initial state is that each activity forms a single block for itself, i.e., blocks only with one activity in it (line 3). Then we try to compute each block set BS_x which contains all blocks with x activities in each block (line 4). Line 8 means that a block containing i activities can be created by merging two disjoint blocks containing j and k activities

$(i = j + k)$. Lines 9 to 18 shows the algorithm to check whether the two block B_j and B_k are able to merge into a bigger block: two blocks can merge iff any two activities a_α, $a\beta$ from the two blocks B_j and B_k shows unique order relations to the rest of activities outside the two blocks B_j and B_k (line 11 to 18). The block set BS containing all possible blocks is the union of block sets BS_x with $x = (0, \ldots, n)$.

The complexity of this algorithm in worst-case scenario is 2^n while n equals the number of activities. However, this complexity only happens when any combination of activities is able to form a block (like a process model with all activities in parallel with each other). During our simulation, for most of time we can enumerate the block from all process models within just a few second. This indicates the complexity in practise is not so high.

B Distance measure

A process model S can be represented by an order matrix A showing the order relations between activities in the model. We therefore measure the distance between two activities a_i, a_j by their relations towards other activities. For activity a_i, we can compute the distributions of activities in the four types of order relation (cf. 5.1), i.e., we compute how many predecessor a_i has and how many successor a_i has, etc. Such distribution can be represented in a 4-dimensional vector $p_i = (n_{i_0}, n_{i_1}, n_{i_*}, n_{i\_})$ where n_{i_0} counts the number of predecessor a_i has, n_{i_1} counts how many successors a_i has, etc. Similarly, we can compute vector p_j for activity a_j. Therefore, the distance between two activities a_i, a_j in process model S can be measured by the geometry distance between p_i and p_j.

For example, take Fig. 4. We can represent activity `C` with $p_{\texttt{C}} = (3, 2, 1, 0)$ and `F` with $p_{\texttt{F}} = (4, 1, 1, 0)$. The distance between `C` and `F` is $(3-4)^2 + (2-1)^2 + (1-1)^2 + (0-0)^2 = 1.41$. Similarly, we can compute the distance between activity `C` and `G` is 3.74. Based on the distances, we can see that activity `F` is closer to `C` than activity `G` is. It is also clear in the process model that activity `F` is a direct successor of `C` while `G` is not. The distance between an activity and a block is measured by the average distances between such activity and all the activities in the block.

We did not choose some common distance measures like shortest path in graph theory [3] or common number of predecessor / successor in tree structure [7]. The reason is that these measures can not show the global influence of a change operation. For example take Fig. 4. If we switch the order between `C` and `F`, it will have lower influence than if we switch the order of `F` and `G`. Although these two pairs have both direct predecessor-successor relationship, changing `F` and `G` will also influence activity `E`. Such difference can only be detected if we use the above mentioned distance measure: distance between `C` and `F` is 1.41, while distance between `F` and `G` is 2.45.

C Precision values in different groups

Group	Para 2	Para 3	Para 4	Para 5	No. PreList *	No. AppList	Precision
1	Small	Small	Random	Random	2	1	0.1218
2	Small	Small	Random	Gaussian	4	1	0.2797
3	Small	Small	Gaussian	Random	6	1	0.1379
4	Small	Small	Gaussian	Gaussian	6	1	0.2323
5	Small	Medium	Random	Random	1	1	0.1801
6	Small	Medium	Random	Gaussian	2	1	0.3400
7	Small	Medium	Gaussian	Random	2	1	0.0912
8	Small	Medium	Gaussian	Gaussian	2	1	0.1746
9	Small	Large	Random	Random	1	1	0.1329
10	Small	Large	Random	Gaussian	1	1	0.2571
11	Small	Large	Gaussian	Random	1	1	0.1911
12	Small	Large	Gaussian	Gaussian	1	1	0.0569
13	Medium	Small	Random	Random	32	1	0.1973
14	Medium	Small	Random	Gaussian	1.008E+4	1	0.2704
15	Medium	Small	Gaussian	Random	432	1	0.0517
16	Medium	Small	Gaussian	Gaussian	54	1	0.1752
17	Medium	Medium	Random	Random	12	1	0.1805
18	Medium	Medium	Random	Gaussian	4	1	0.3800
19	Medium	Medium	Gaussian	Random	720	1	0.0497
20	Medium	Medium	Gaussian	Gaussian	96	1	0.1868
21	Medium	Large	Random	Random	4	1	0.1141
22	Medium	Large	Random	Gaussian	16	1	0.3685
23	Medium	Large	Gaussian	Random	12	1	0.1078
24	Medium	Large	Gaussian	Gaussian	24	1	0.1852
25	Large	Small	Random	Random	1.271E+16	1	0.1099
26	Large	Small	Random	Gaussian	3.583E+8	1	0.2201
27	Large	Small	Gaussian	Random	1.229E+20	1	0.0973
28	Large	Small	Gaussian	Gaussian	2.152E+14	1	0.1797
29	Large	Medium	Random	Random	2.877E+10	1	0.1868
30	Large	Medium	Random	Gaussian	8.847E+6	1	0.2412
31	Large	Medium	Gaussian	Random	1.672E+11	1	0.1402
32	Large	Medium	Gaussian	Gaussian	4.977E+7	1	0.2347
33	Large	Large	Random	Random	1.084E+12	1	0.2212
34	Large	Large	Random	Gaussian	1024	1	0.3752
35	Large	Large	Gaussian	Random	1.161E+8	1	0.2281
36	Large	Large	Gaussian	Gaussian	13824	1	0.2596

Table 11. Precision values for different groups

*Note: the maximal possible number of permutations:

1. Para 2 = "Small" : 10! = 3.629E+6
2. Para 2 = "Medium" : 20! = 2.433E+18
3. Para 2 = "Large" : 50! = 3.041E+64