What are the Problem Makers: Ranking Activities According to their Relevance for Process Changes

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

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

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

Abstract

Recently, a new generation of adaptive process management technology has emerged, which enables dynamic changes of composite services and process models respectively. This, in turn, results in a large number of process variants derived from the same process model, but differing in structure due to the applied changes. Since such process variants are expensive to maintain, the process model should be evolved accordingly. In this context, we need to know which activities have been more often involved in process adaptations than others, such that we can focus on them when reconfiguring the process model. This paper provides two approaches for ranking activities according to their involvement in process adaptations. The first one allows to precisely rank the activities, but is expensive to perform since the algorithm is at $\mathcal{NP}$ level. We therefore provide as alternative an approximation ranking algorithm which computes in polynomial time. The performance of the approximation algorithm is evaluated and compared through a simulation of 3600 process models. Statistical significance tests indicate that the performance of the approximation ranking algorithm does not depend on the size of process models, i.e., our algorithm can scale up.

1 Introduction

In today's dynamic business world, success of an enterprise increasingly depends on its ability to react to changes in its environment in a quick, flexible and cost-effective way. Along this trend a variety of process and service support paradigms as well as corresponding specification languages (e.g., WS-BPEL, WS-CDL) have emerged. In addition, there exist different approaches for adaptive processes and services respectively [11, 13]. 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 [18]. 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.

In most approaches supporting structural adaptations of process models, the resulting process variants have to be maintained separately. Then even simple changes often require manual re-editing of a large number of variants. Over time this leads to divergence of the process variant models, which aggravates maintenance significantly. Fig. 1 gives an illustrating example. Out of reference model S, five process variants have been configured, which are weighted based on the number of process instances created from them. In our example, 30% of all instances were executed according to variant S_1, while 15% of the instances did run on S_2. Generally, a large number of process variants may exist at both the process type and process instance level [7].

As deleted or newly inserted activities can be easily identified by comparing the activity sets of the reference model with those of its variants, this paper focuses on analyzing structural process changes through the movement of activities (e.g., swapping the order of activities or arranging two activities in parallel that were ordered sequentially before). We are aiming at finding the *problem makers*, i.e., those activities that are involved in process adaptations more often than others. These activities, in turn, cause most deviations from the given reference model and thus lead to highest adaptation 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 [12]. The discovered information is particularly useful for monitoring the deviations from the predefined composite service (i.e., process model)

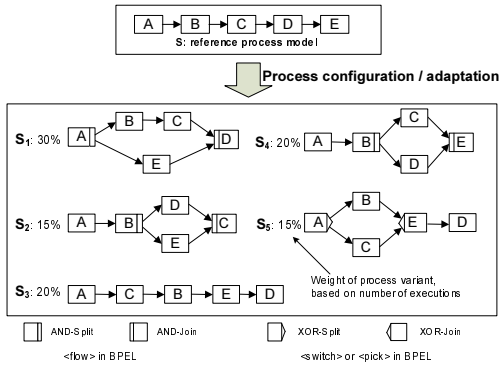

Figure 1. Illustrative example

or for redesigning it through learning from past executions.

Based on the two assumptions that: (1) process models are block-structured (like for example BPEL) 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 S and a collection of process variants S_i configured from it, how to rank process activities according to their involvement in structural adaptations of S (i.e., the adaptations that become necessary when configuring the process variants out of S)?*

Section 2 gives background information needed for understanding this paper. We provide a precise, but expensive ranking algorithm in Section 3 and a more efficient approximation ranking algorithm in Section 4. To test the performance of the two algorithms, Section 5 describes the setup and the results of a simulation. Section 6 discusses related work and Section 7 concludes with a summary and 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* $S = (N, E, \ldots)^1 \in \mathcal{P}$ is defined as Well-structured Activity Net [11]. 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). Examples are 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 [11]. Such change operations modify the initial process model by altering the set of activities and/or their order relations. Thus, each application of a change operation results in a new process model. We define *process change* and *process variants* as follows:

Definition 1 (Process Change and Process Variant) *Let $\mathcal{P}$ denote the set of possible process models and $\mathcal{C}$ be the set of possible process changes. Let $S, S' \in \mathcal{P}$ be two process models, let $\Delta \in \mathcal{C}$ be a process change 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 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.*

Examples of high-level change operations include *insert activity*, *delete activity*, and *move activity* as implemented in the ADEPT change framework [11]. While *insert* and *delete* modify the set of activities in the process model, *move* changes activity positions and thus the order relations in a process model. For example, operation $move(S, \texttt{A}, \texttt{B}, \texttt{C})$ shifts 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 [11]. 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 [18]. 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 [17]. We refer to ADEPT since it covers by far most high-level change patterns and change support features [18], 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 S into 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 of model transformations. 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 models S and S' their bias is not necessarily unique [17, 9]. As example take Fig. 1. Here, the distance between model S and variant S_4 is *one*, since we only need to perform one

[1] A formal definition of a Well-structured Activity Net contains more than only node N and edge E, we have ignored others since they are not used in our context

change operation $\Delta_1 = move(S, \texttt{C}, \texttt{B}, \texttt{E})$ to transform S into S_4. However, it is also possible to transform S into S_4 with $\Delta_2 = move(S, \texttt{D}, \texttt{B}, \texttt{E})$. 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 [9] 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 distance between process models. This guarantees soundness of process models and provides a more meaningful measure for distance [9].

Trace: A *trace* t on process model $S = (N, E, \ldots)$ denotes a valid as well as complete execution sequence $t \equiv < a_1, a_2, \ldots, a_k >$ of activity $a_i \in N$ 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 $\exists i < j : a_i = a \wedge a_j = b$.

Order Matrix One key feature of any change framework is to maintain the structure of the unchanged parts of a process model [11]. To incorporate this in our approach, rather than only looking at direct predecessor-successor relation between activities (i.e., control edges), we consider the transitive control dependencies for each activity pair; i.e., for 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 relation. Logically, we determine order relations by considering all traces the process model can produce. Results are aggregated in an order matrix $A_{|N|\times|N|}$, which considers four types of control relations (cf. Def. 3):

Definition 3 (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.*

Regarding our example from Fig. 1, the order matrix for each of the process variant S_i is presented on the top of Fig. 3. Variants S_i contain four kinds of control connectors: AND-Split and AND-Join (corresponding to a flow activity in BPEL), and XOR-Split and XOR-join (corresponding to a switch or pick activity in BPEL). The depicted order matrices represent all four described relationships. As example consider S_5. Activities B and C 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_{BC} for S_5. If certain conditions are met, the order matrix can uniquely represent the process model. Analyzing its order matrix (cf. Def. 3) is then sufficient in order to analyze the process model (see [9] for details).

3 Precise Activity Ranking Algorithm

In this section, we provide an approach to evaluate the potential involvement of each activity a_i in process configurations. We denote such involvement as **change impact** $CI(a_i)$ of this activity. Based on $CI(a_i)$, we are able to rank activities, which we denote as **activity ranking list**.

Since we do not presume the presence of change or execution logs respectively, the major information we can use for our analysis are the bias sets $\mathcal{B}_{S,S_i}$, which can be computed by measuring the structural differences between the reference process model and each of its variant S_i (cf. Def. 2). 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. It, therefore, can be considered as a purified change log for our analysis.

3.1 Computing Changed Activity Set

Let us re-consider the example from Fig. 1. By scanning the reference process model S and a process variant $S_i (i = 1 \ldots 5)$, we are able to compute bias set $\mathcal{B}_{S,S_i}$ [9]. 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 model adaptations rather than the order in which the latter were applied. For example, bias set $\mathcal{B}_{(S,S_2)}$ contains the two changes σ_1, σ_2 where $\sigma_1 =< \Delta_1, \Delta_2 >$ with $\Delta_1 = move(S, \texttt{D}, \texttt{B}, \texttt{C})$ and $\Delta_2 = move(S, \texttt{E}, \texttt{B}, \texttt{C})$ 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 moved rather than the order of the move operations or the position to which activities were moved.

Therefore, we keep the granularity of our bias analysis only on the activities that were potentially re-positioned. Regarding our example, we only want to document these activities (i.e., $\{\texttt{D}, \texttt{E}\}$) in the context of σ_1 (see above) rather

than the change operation σ_1 itself. When only looking at the moved activities, σ_1 does not differ from σ_2.

Definition 4 (Changed Activity Set) *We define set $\mathcal{A}_\sigma$ representing the activities that are 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. 4, an element $\mathcal{A}_\sigma$ of the Changed Activity Set $\mathcal{C}_{(S,S')}$ corresponds to a set representing the activities being changed by bias σ. For our example from Fig. 1, the changed activity sets of the reference model S and its variants S_i are listed in Table 1. 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.

3.2 Computing the Change Impact $CI(a_j)$

We now measure the change impact $CI(a_j)$ of activity a_j by computing its involvement in each Change Activity Set $\mathcal{C}_{(S,S_i)}$ of model S and its variants $S_i (i = 1, \ldots, n)$.

For an activity $a_j \in \mathcal{A}_\sigma \in \mathcal{C}_{(S,S_i)}$, we can compute its potential involvement in structural adaptations of S when transforming it into variants S_i using $\frac{|\{a_j \in \mathcal{A}_\sigma | \mathcal{A}_\sigma \in \mathcal{C}_{(S,S_i)}\}|}{|\mathcal{C}_{(S,S_i)}|}$. This formula measures for all changes transforming S into S_i (summarized by $\mathcal{C}_{S,S_i}$, i.e., the denominator), to what percentage they involve a_j (i.e., the numerator). If w_i measures the weight of S_i, Change Impact $CI(a_j)$ measures the potential involvement of activity a_j in the configuration of the different variants (i.e., the necessary structural adaptation of the reference model).

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

Fig. 2 summarizes the change impact of each activity a_j as it can be derived from the configured variants S_i in our example (cf. Fig. 1). 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, we obtain $CI(\texttt{B}) = 0.175$, which means that we need to move B on average 0.175 times when configuring a variant out of the given reference model. Fig. 2 also shows the ranking of the activities based on their change impact.

The described approach is precise: all possible changes that may have contributed to the configuration of a variant are enumerated and the change impact of each activity is computed by analyzing the reference model and its variants. However, enumerating all possible changes between two models is a $\mathcal{NP}$ problem [9]. Therefore, this approach will not scale up at the presence of a large number of variants with complex structure (i.e., models with dozens up to hundreds of activities). Section 4 introduces an approximation algorithm to solve the problem in an efficient way.

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. Changed activity sets $\mathcal{C}_{(S,S_i)}$

4 Approximation Activity Ranking Algorithm

To reduce complexity when computing the change impact for each activity, we introduce an approximation algorithm which only requires polynomial time.

4.1 Aggregated Order Matrix

In Def. 3, we have defined the order matrix which can uniquely represent a block-structured process model. In order to analyze a given collection of process variants, we first compute the order matrix for each of these variants (cf. Def. 3). Regarding our example from Fig. 1, we obtain five order matrices (cf. Fig. 3). As the order relation between two activities might be not the same in all order matrices, we represent it as a distribution based on the four types of order relations (cf. Def. 3). Regarding our example, in 65% of all cases activity C succeeds activity B (as for variants S_1, S_2, S_4), in 20% of all cases C precedes 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. 3). Therefore, for a given collection of variants, we can define the order relation between two activities a and b captured by these variants 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. 3. For our example from Fig. 3, for instance, we obtain $V_{CB} = (0.65, 0.2, 0, 0.15)$. Fig. 3 shows the aggregated order matrix of the process variants from Fig. 1. A non-filled value in a certain dimension means it corresponds to zero.

4.2 Approximation Algorithm

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. When reconsidering the reference process model from Fig. 1, we can

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	

Figure 2. Change impact of each activity

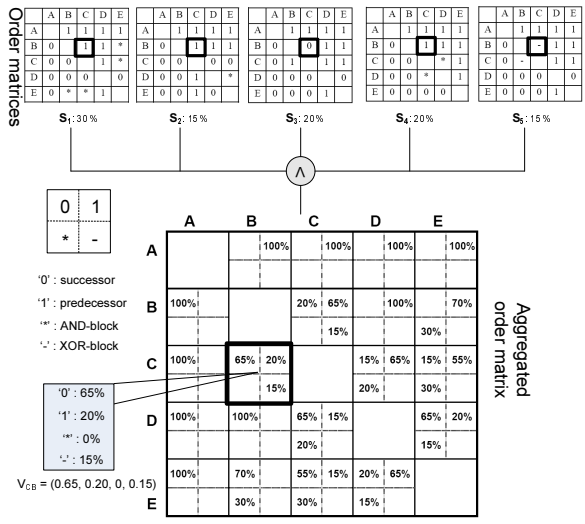

Figure 3. Aggregated order matrix V

see that the order relation between activities C and B is "0", i.e., C succeeds B. If we built an aggregated order matrix V^{ref} purely based on this reference model, we would obtain $V_{CB}^{ref} = (1, 0, 0, 0)$, 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_{CB}^{ref} (which represents the reference model), their difference indicates that, the position of B or C might have changed when configuring reference model into the variants. Generally, we can assume that the more an activity is involved in configuration of the process variants, the more its order relation in the variants differs when compared to the reference model. To quantitatively measure this difference, 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 example, for instance, we obtain $f(V_{CB}, V_{CB}^{ref}) = 0.933$. This number indicates high similarity of the order relations between B and C in the reference model and the ones in the variants.

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

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

$CI_a(a_j) \in [0, 1]$ corresponds to the average square mean of the similarity (measured by Formula (2)) between activity a_j and the rest of activities. It therefore approximately reflects how much a_j has been re-configured.[2] If $CI_a(a_j) = 1$ holds, activity a_j will have exactly same order relations with respect to the other activities in both the reference model and all the variants. For this case, we can assume that the activity has not been moved. If not, we can assume a_j has been involved in process configurations for a certain degree. Note that our ranking is based on descending orders, i.e., the higher the change impact $CI_a(a_j)$ is, the lower the chance will be that the activity has been potentially moved. Regarding our example from Fig. 1, the ranking result of the five activities and their change impact $CI_a(a_j)$ are shown in Table 2. Clearly, activity E is moved most frequently while activity A is the least moved one.

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

Table 2. Approximate ranking result

4.3 Algorithm Comparison

The approximation algorithm is a polynomial one, i.e., complexity for computing change impact $CI_a(a_j)$ of activity a_j is at $\mathcal{O}(n^3 \times m)$ where n is the number of activities per variant and m is the number of variants. 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 its performance, i.e., we must show how close it is to the real optimum (i.e., the ranking provided by the precise algorithm).

If we simply compare the result of the precise ranking algorithm (cf. Fig. 2) and the approximation ranking algorithm, we can easily claim that the performance of the approximation ranking algorithm is quite good, since it generates the same ranking order as the precise ranking algorithm does. Clearly, such a simple comparison is far from being sufficient. In the following, we will use simulation to answer the following two questions:

1. *How good does the approximation algorithm perform, i.e., how close are its ranking results in comparison to the precise ones?*
2. *Can the approximation ranking algorithm scale up, i.e., does its performance depend on the size of the process models?*

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

5 Simulation

In our simulations, we have identified several parameters for which we want to investigate whether or not they influence the performance of our approximation ranking algorithm. Due to space limitation, this paper only discusses one parameter, the *size of a process model*. We provide a detailed analysis of the other parameters in [10].

To analyze the influence of the size of process models, we generate process models of three different sizes:

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

Based on different scenarios, we generate 36 groups of datasets. Each of them contains:

1. **The reference model**, i.e., a randomly generated model from which we configure the process variants (see [10] for details).
2. **The process variants**. We generate each variant by configuring the reference model according to a particular scenario. For each group we generate 100 process variants. [3]

Using the reference process model and the 100 process variants, we can rank the activities with the precise and the approximation ranking algorithm. Table 3 shows the ranking result of a scenario with 10 activities per variant. We do the same for the 36 groups of datasets (i.e., 3636 process models) as generated according to different scenarios. [4]

5.1 Evaluation Approach

From Table 3 it becomes clear that the precise ranking algorithm and the approximation ranking algorithm do not always provide same results, e.g., activity `I` is ranked third regarding the precise ranking result, but ranked fourth regarding the result of the approximation ranking. In this subsection, we evaluate the performance of the approximation ranking algorithm, i.e., we measure how close 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 like data mining or information retrieval [16, 1]. In our context, we know that 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 subsets of the top n ranked activities are NOT necessarily same, e.g., let us compare the top three activities as provided by the two ranking algorithms (cf. Table 3). While the precise ranking list contains `A, F` and `I`, the approximation ranking list comprises `A,F` and `J`. The difference between the top n activities in the two ranking list can be measured using $Precision(n)$:

Definition 5 (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:*

$$Precision(n) = \frac{|P(n) \bigcap A(n)|}{|P(n)|} \tag{4}$$

$Precision(n)$ reflects how much "useful information" about the actual top n activities (measured by the precise ranking algorithm) we can get when applying the approximation algorithm. As example consider Table 3. 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`. Therefore, $precision(3) = 2/3 = 0.6667$ holds. Table 3 shows all precision values concerning the top n ranked activities. Fig. 4 additionally plots the precision values.

We can derive the curve depicted in Fig. 4 by plotting and interpolating all precision values. Besides, we plot a optimum line $precision(n) = 1$, $n = 1 \ldots 10$ and further mark the surface area between the two curves. The size of this surface area then can be used to evaluate the performance of the approximation ranking algorithm for the given dataset. If the precise and approximation ranking algorithms provide different ranking results, there will be a number n' such that $precision(n')$ does not equal to 1. In this case, the precision curve deviates from the optimum curve and creates space. Regarding our example, the surface area occupies 11.2% of the value space (the rectangle between (0,0) and (10,1) in our case). This number can be used as indicator showing how close the approximation line is to the real optimum line, i.e., showing how good our approximation algorithm works. The larger this area is, the bigger the difference of the two ranking results is, and the worse the approximation algorithm actually performs.[5] Altogether, the proposed method is used to evaluate the performance of our approximation algorithm in the conducted simulation comprising 36 groups of datasets.

5.2 Evaluation Result

5.2.1 Surface Area Distributions

We first analyze the distributions of the surface area values for different groups. A standard method is to use his-

[3] Note that the scenario only describes the statistical feature of the collection of variants configured from the reference model, it does not control how one particular variant is generated, i.e., the 100 variants are not the same but only share a certain feature (e.g., same size).

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

[5] This evaluation method is inspired by the precision-recall curve used in information retrieval [1] and statistics [15]. We omit "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 correlation analysis [15] since we are interested in the ranking order rather than the exact change impact of activities.

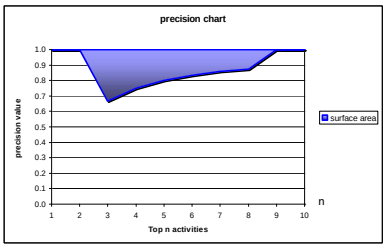

Figure 4. Surface area for the precision chart

tograms [15]. A histogram shows the distribution of surface areas for different intervals. The result is given in Fig. 5. The value range of the surface area is [0, 0.4] in all 36 groups, and the surface area of most groups (i.e.,10 out of 36 groups) falls into interval [0.15,0.2). When computing the mean and standard deviation of the surface area, we obtain as mean 0.1933 and as standard deviation 0.0871. Using $Kolmogorov$-$Smirnov$ test [15], we can see that the probability of the surface area following a Gaussian distribution is 91.8% (see Fig. 5 for the fitting lines). We also test the confidence interval of this surface area since it is an important factor to measure performance of the algorithm. The 95% confidence interval is [0.1637,0.2225], which indicates that the mean of the surface area has 95% probability falling into the interval [0.1637,0.2225].

5.2.2 Scalability of the Ranking Algorithm

We now analyze whether the approximation ranking algorithm scales up, i.e., whether its performance depends on the size of process models. For this purpose we divide our 36 groups of datasets into 3 sub-groups: one with small-sized models, one with medium-sized models and one with large-sized models. We then analyze whether the surface areas from the three groups are significantly different from each other. If so, the size of models has significant influence on the performance of the approximation algorithm. Consequently, we want to test the following null hypothesis: *"H_0: The size of process model has no influence on the surface area."*

If the hypothesis is tested to be statistically significant (i.e., probability larger than 5%), we accept it, i.e., the size is assumed to have no influence on the performance of the approximation algorithm. Comparable to most hypothesis tests, we assume that errors are independent and follow normal distribution [15, 6]. The standard approach to this type of problem is to examine the data using a two-way Analysis of Variance (ANOVA) [15, 6]. Let us divide the dataset into sub-sets $Y_j, j = 1 \ldots m$ based on the size of the model. Let $y_{ij}, i = 1 \ldots n$ be the surface area of a group in set Y_j. Since we consider three model sizes ("small-sized", "medium-sized" and "large-sized") in our example, m is 3 and $n = |Y_j| = 36/3 = 12$. Let $\overline{y}$ be the average surface area of all groups, let $\overline{y}_j$ represent the average surface area of the groups in Y_j, and let $\overline{y}_i$ be the average surface area of three corresponding groups in each sub-set Y_j. Two-way

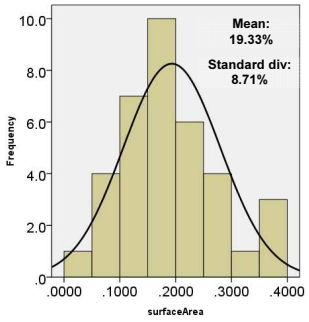

Figure 5. Histogram of surface area value

ANOVA can be computed as follows:

$$F_A = \frac{\frac{n\sum_j(\overline{y_j}-\overline{y})^2}{m-1}}{\frac{\sum_{i,j}(y_{ij}-\overline{y_i}-\overline{y_j}+\overline{y})^2}{(n-1)(m-1)}} \quad (5)$$

Probability of accepting our hypothesis H_0 follows F distribution with $n-1$ and $(n-1)(m-1)$ degrees of freedom [15]. In our example, F_A is 1.3665, which indicates that the probability of accepting H_0 is 0.2758, i.e., it is significant. This means we can **accept** H_0 and thus the size of process models **does NOT influence** the size of the surface area. This, in turn, proves that the performance of our approximation algorithm is stable; i.e., it can scale up.

6 Related work

Ranking techniques have been widely used in fields like information retrieval [1] or data mining [16]. 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 the workflow field, conformance checking techniques are widely used to measure the match between the designed process model and its actual executions [14]. Such technique has also been applied in certain process mining approaches like genetic mining [3]; [5] additionally represents a process mining technique by discovering a collection of process variants. However, a prerequisite of this approach is a valid change log which is not always available in practice. Similar techniques for conformance checking have been applied in process monitoring where people focus on handling exceptional situations and measuring fulfillment of business rules [4]. In the web services field, service monitoring techniques are also used to monitor the behavior of the agreed service compositions. Violations of these agreement can be identified and be punished [2]. However, most of the mentioned approaches analyze behavior inconsistencies 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 [8] for a detailed comparison). Also, few of the above mentioned approaches are able to provide a detailed analysis of every individual process activity based on the observed process variants.

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	6	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
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 3. Precision table

7 Summary and Outlook

One key contribution of this paper is to provide both a precise algorithm and an approximation algorithm to rank the activities according to their potential involvement in process reconfigurations. Using these techniques, we are able to identify which activities have been configured more often than others. Such information is valuable for identifying optimization of the currently used (reference) process model or when re-designing process models. It can also be used in process monitoring to identify which parts of a composite service have been adapted more often than others during run time.

The precise ranking algorithm is precise but also time-consuming. Therefore, we introduce the approximation ranking algorithm, which can be computed in polynomial time. Its performance has also been evaluated by a simulation. After analyzing about 3600 process models, we demonstrated that the precision of the approximation ranking algorithm is around 80% and the performance of the approximation ranking algorithm can scale up. Our next step is to make use of the suggested technique for process variant mining [7]. Based on the ranking result, we can focus on highly ranked activities (i.e., more relevant adaptations), and the trivial configurations will not be considered when discovering a new reference model by learning from the variants.

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

[4] 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, 2001.

[5] 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. *International Journal of BPIM*, 3(1):61–78, 2008.

[6] David Hull. Using statistical testing in the evaluation of retrieval experiments. In *SIGIR'93*, pages 329–338, 1993.

[7] 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.

[8] C. Li, M. Reichert, and A. Wombacher. Mining process variants: Goals and issues. In *IEEE SCC (2)*, pages 573–576. IEEE Computer Society, 2008.

[9] 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.

[10] C. Li, M. Reichert, and A. Wombacher. What are the problem makers: Discovering the most frequently changed activities in adaptive processes. Technical Report TR-CTIT-09-05, Enschede, 2009.

[11] M. Reichert and P. Dadam. ADEPTflex -supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[12] 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.

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

[14] 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.

[15] D.J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, 2004.

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

[17] 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.

[18] 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.