# A Heuristic Approach for Discovering Reference Models by Mining Process Model Variants

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 Systems, 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 Systems (PAISs) has emerged, which enables structural process changes during runtime while preserving PAIS robustness and consistency. Such flexibility, in turn, leads to a large number of process variants derived from the same model, but differing in structure. Generally, such variants are expensive to configure and maintain. This paper provides a heuristic search algorithm which fosters learning from past process changes by mining process variants. The algorithm discovers a reference model based on which the need for future process configuration and adaptation can be reduced. It additionally provides the flexibility to control the process evolution procedure, i.e., we can control to what degree the discovered reference model differs from the original one. As benefit, we can not only control the effort for updating the reference model, but also gain the flexibility to perform only the most important adaptations of the current reference model. Our mining algorithm is implemented and evaluated by a simulation using more than 7000 process models. Simulation results indicate strong performance and scalability of our algorithm even when facing large-sized process models.

## 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 [22]. However, current off-the-shelf enterprise software does not meet these needs [23]. It is deployed in different companies, domains, and countries, and therefore tends to be too generic and rigid. Generally, introduction of enterprise software entails the problem of aligning business processes and IT. This causes huge customization efforts at the site of software buyers that exceed the price for the software licenses by factor five to ten [5]. Software vendors, in turn, make endeavors to close this alignment gap [34], and major progress has

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

been achieved by shifting from function- to process-centered software design. Along this trend a variety of process support paradigms as well as languages have emerged. Using WS-BPEL [1], for example, an executable process can be composed out of existing application services. At runtime, execution of these services is then orchestrated by the PAIS according to the defined process logic. Recently, different approaches for adapting processes have emerged. Generally, structural process adaptations are not only needed for configuration purpose at build time [9, 32], but also become necessary for single process instances during runtime to deal with exceptional situations and changing needs [25, 41].

In response to this need adaptive process management technology has emerged [41, 43]. It allows to configure and adapt process models at different levels. This, in turn, results in large collections of process model variants (*process variants* for short) created from the same process model, but slightly differing 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 respective process instance, we need to move activity *receive treatment* from its current position to the one parallel to activity *register*. This leads to instance-specific process model variant $S'$ as shown in Fig. 1b. Generally, a large number of process variants derived from same original process model exist [22].

### 1.1 Problem Statement

Though considerable efforts have been made to ease process configuration and customization [9, 25], most existing approaches have not yet utilized the information resulting from past process adaptations [40]. Fig. 2 describes the goal of this paper. We aim at learning from past process changes by "merging" process variants into one generic process model, which covers these variants best. By adopting this generic model as new*reference process model* within the PAIS, need for future process adaptations and thus cost for change will decrease.

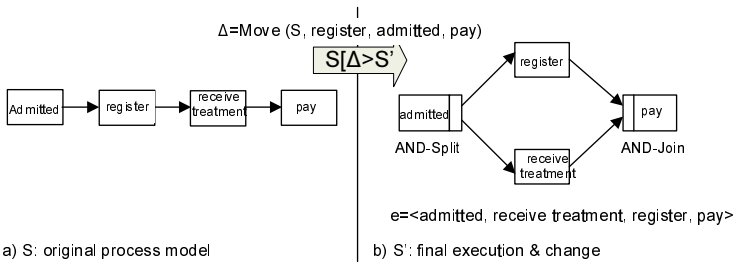



**Fig. 1.** Original Process Model S and Process Variant S'

When deriving a new process reference model, the original one should be also taken into account. In most cases, "dramatic" changes of the current reference model are not preferred due to implementation costs or social reasons. Process designers should therefore have the flexibility to choose to what degree they want to change the original reference model to fit better to the variants.

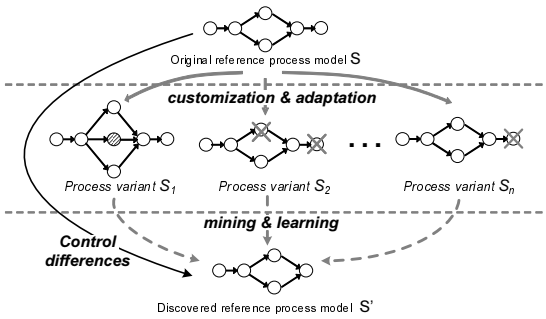



**Fig. 2.** Discovering a new reference model by learning from past process configurations

Based on the two assumptions that (1) process models are well-formed (i.e., block-structured like in WS-BPEL) and (2) all activities in a process model have unique labels, this paper deals with the following fundamental research question: *Given a reference model and a collection of process variants configured from it, how to derive a new reference process model by performing a sequence of change operations on the original one, such that the average distance between the reference model and the process variants becomes minimal?*

The distance between the reference process model and a process variant is measured by the number of high-level change operations (e.g., to insert, delete or move activities [25]) needed to transform the reference model into the respective variant. Clearly, the shorter the distance is, the less efforts needed for process adaptation are. Basically, we discover a new reference model by performing a sequence of change operations on the original one. In this context, we provide users the flexibility to control the distance between old reference model and newly discovered one, i.e., to choose how many change operations shall be applied to the old reference model. Clearly, the most relevant changes (which significantly contribute to reduce the average distance) should be considered first and the less important ones last. Particularly, if users decide to ignore the less relevant changes, the overall performance of our algorithm with respect to the described research goal will not be influenced too much. Such flexibility to control the difference between the original and the discovered model is a significant improvement when compared to our previous work [15]; the approach presented in [15] enables discovery of a reference process model by mining a collection of variants, but is unable to take the original reference process model into account.

The remainder of this paper is organized as follows. Section 2 gives background information needed for understanding this paper. Section 3 introduces our heuristic search algorithm and provides a high-level overview on how it can be used for mining process variants. We describe two important aspects of our heuristics algorithm, (i.e., the fitness function and the search tree) in Sections 4 and 5. To evaluate the performance of our mining algorithm, we conduct a simulation. Section 6 describes its setup, while Section 7 presents the simulation results. Finally, Section 8 discusses related work and Section 9 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* $S = (N, E, \ldots)$[4] $\in \mathcal{P}$ is defined as well-structured Activity Net [25]. $N$ constitutes the set of 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 (similar to WS-BPEL). A simple example is depicted in Fig. 3. For a detailed description and correctness issues, we refer to [25].

***Process change***: A process change is accomplished by applying a sequence of high-level change operations to a given process model $S$ over time [25]. Such operations modify the initial process model by altering its 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, and let $\sigma = \langle \Delta_1, \Delta_2, \ldots \Delta_n \rangle \in \mathcal{C}^*$ be a sequence of changes performed on initial model $S$. Then:*

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

Examples of high-level change operations include *insert activity*, *delete activity*, and *move activity* as implemented in the ADEPT change framework [25]. While *insert* and *delete* modify the set of activities in the process model, *move* changes activity positions and thus the structure of the process model. A formal semantics of these change patterns is given in [31]. For example, operation $move(S, \texttt{A}, \texttt{B}, \texttt{C})$ moves activity `A` from its current position within process model $S$ to the position after activity `B` and before activity `C`. Operation $delete(S, \texttt{A})$, in turn, deletes activity `A` from process model $S$. Issues concerning the correct

[4] A Well-structured Activity Net contains more elements than only node set $N$ and edge set $E$, which can be ignored in the context of this paper.

use of these operations, their generalization and formal pre-/post-conditions are described in [25]. Though the depicted change operations are discussed in relation to our ADEPT change framework, they are generic in the sense that they can be easily applied in connection with other process meta models as well [31, 43]. For example, a process change as realized in the ADEPT framework can be mapped to the concept of life-cycle inheritance known from Petri Nets [37]. We refer to ADEPT since it covers by far most high-level change patterns and change support features when compared to other adaptive PAISs [41, 43].

**Definition 2 (Bias and Distance).** *Let $S$, $S' \in \mathcal{P}$ be two process models. Then:* ***Distance*** *$d_{(S,S')}$ between $S$ and $S'$ corresponds to the minimal number of high-level change operations needed to transform $S$ into $S'$; i.e., we define $d_{(S,S')} := min\{|\sigma| \mid \sigma \in \mathcal{C}^* \wedge S[\sigma\rangle S'\}$. Furthermore, a sequence of change operations $\sigma$ with $S[\sigma\rangle S'$ and $|\sigma| = d_{(S,S')}$ is denoted as* ***bias*** *between $S$ and $S'$.*

The *distance* between $S$ and $S'$ is the minimal number of high-level change operations needed for transforming $S$ into $S'$. The corresponding sequence of change operations is denoted as *bias* $B_{S,S'}$ between $S$ and $S'$.[5]Usually, such distance measures the complexity for model transformation (i.e., configuration). As example take Fig. 1. Here, distance between model $S$ and variant $S_1$ is *one*, since we only need to perform one change operation $move(S, \texttt{register,admitted,pay})$ to transform $S$ into $S'$ [17]. In general, determining bias and distance between two process models has complexity at $\mathcal{NP}$ level [17]. We consider high-level change operations instead of change primitives (i.e., elementary changes like adding or removing nodes / edges) to measure the distance between process models. This allows us to guarantee soundness of process models and provides a more meaningful measure for distance [17, 41].

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

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

We only consider traces composing 'real' activities, but no events related to silent ones, i.e., nodes within a process model having no associated action and only existing for control flow purpose [17]. At this stage, we consider two process models as being the same if they are *trace equivalent*, i.e., $S \equiv S'$ iff $\mathcal{T}_S \equiv \mathcal{T}_{S'}$. The stronger notion of bi-similarity [10] is not needed in our context.

[5] Generally, it is possible to have more than one minimal set of change operations to transform $S$ into $S'$, i.e., given process models $S$ and $S'$ their bias does not need to be unique. A detailed discussion of this issue can be found in [37, 17].

## 3 Overview of Our Heuristic Search Algorithm

Section 3.1 provides a running example which we use throughout the paper. In Section 3.2, we introduce our heuristic search algorithm and give a high-level overview of how it can be applied for mining process variants.

### 3.1 Running Example

An illustrating example is given in Fig. 3. Out of an original reference model $S$, six different process variants $S_i \in \mathcal{P}$ $(i = 1, 2, \ldots 6)$ have been configured. These variants do not only differ in structure, but also with respect to their activity sets. For example, activity `X` appears in 5 of the 6 variants (except $S_2$), while `Z` only appears in $S_5$. The 6 variants are further weighted based on the number of process instances created from them. In our example, 25% of all instances were executed according to variant $S_1$, while 20% ran on $S_2$. If we only know the process variants, but have no runtime information about related instance executions, we assume variants to be equally weighted; i.e., every process variant then has weight $1/n$, where $n$ corresponds to the total number of variants.

We can also compute the distance (cf. Def. 2) between original reference model $S$ and each variant $S_i$. For example, when comparing $S$ with $S_1$ we obtain distance *4* (cf. Fig. 3); i.e., we need to apply four high-level change operations ($move(S, \texttt{H,I,D})$, $move(S, \texttt{I,J}, endFlow)$, $move(S, \texttt{J,B}, endFlow)$ and $insert(S, \texttt{X,E,B})$; cf. Def. 1) to transform $S$ into $S_1$. Based on weight $w_i$ of each variant $S_i$, we can then compute average weighted distance between reference model $S$ and its variants. Regarding our example, as distances between $S$ and $S_i$ we obtain: $4(i = 1, \ldots, 6)$[6](cf. Fig. 3). When considering the variant weights, as average weighted distance, we obtain $4 \times 0.25 + 4 \times 0.2 + 4 \times 0.15 + 4 \times 0.1 + 4 \times 0.2 + 4 \times 0.1 = 4.0$. This means we need to perform on average 4.0 change operations to configure a process variant (and related instance respectively) out of the reference model. Generally, *average weighted distance* between a reference model and its process variants represents how "*close*" they are. The goal of our mining algorithm is to discover a reference model for a collection of (weighted) process variants with minimal average weighted distance to the variants.

### 3.2 Heuristic Search for Process Variant Mining

As discussed in Section 2, measuring the distance between two models is an $\mathcal{NP}$ problem, i.e., the time for computing the distance is exponential to the size of the process models. Consequently, the problem set out in our research question (i.e., finding a reference model which has minimal average weighted distance to the variants), is an $\mathcal{NP}$ problem as well. When encountering real-life cases (i.e.,

[6] In our example, all variants have the same distance to the original reference model. We specially designed it in this way in order to better explain our simulation as presented in Section 6. Clearly, our algorithm can also be applied when variants have different distances to the original reference model.

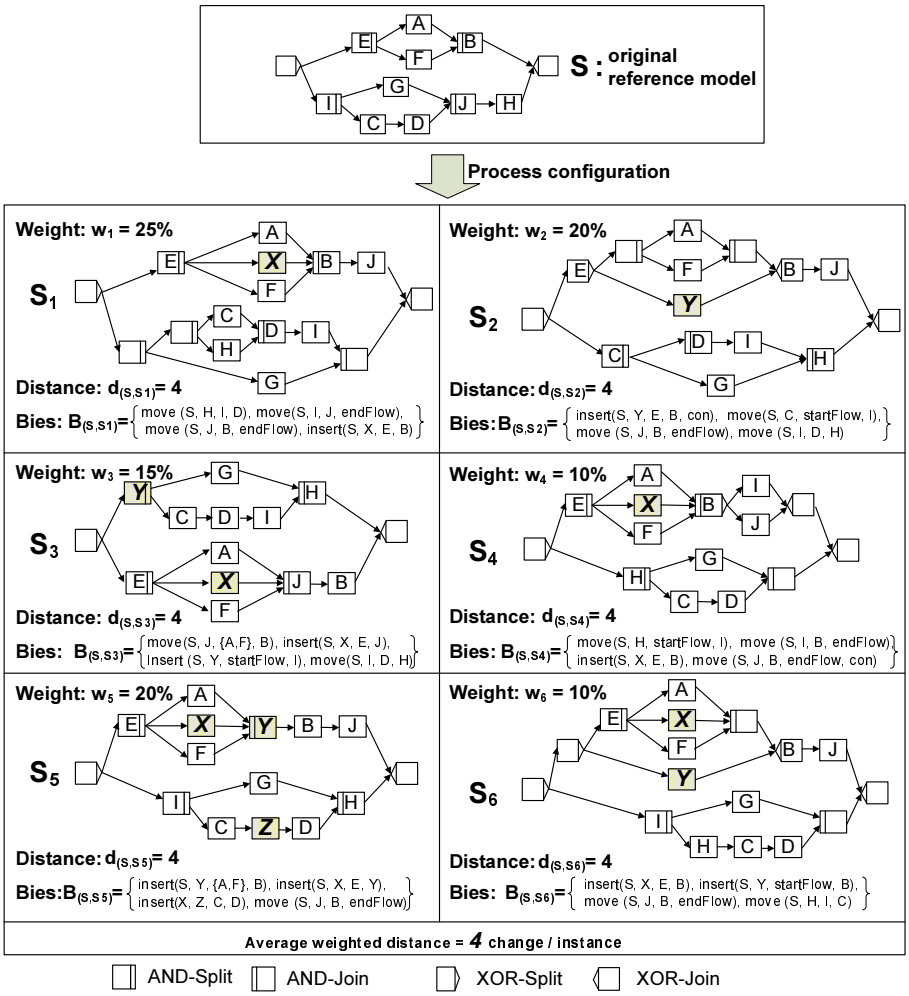



**Fig. 3.** Illustrating example

thousands of variants with complex structure), finding "the optimum" would therefore be either too time-consuming or not feasible. In this paper, we present a **heuristic search algorithm** for variant mining. Our overall goal is to find a solution which is close to "the optimum", but can be computed in a reasonable amount of time.

Heuristic algorithms are widely used in various fields of computer science, e.g., artificial intelligence [20], data mining [36] and machine learning [24]. A problem employs heuristics when "it may have an exact solution, but the computational cost of finding it may be prohibitive" [20]. Although heuristic algorithms do not aim at finding the "real optimum" (i.e., it is neither possible to theoretically prove that the discovered result is the optimum nor can we say how

close it is to the optimum), they are widely used in practice. Usually heuristic algorithms provide a nice balance between the goodness of the discovered solution and the computation time for finding it [20].

Regarding the mining of process variants, Fig. 4 illustrates on how heuristic algorithms can be applied in our context. Here we represent each process variant $S_i$ as a single node in the two dimensional space (white node). The goal for variant mining is then to find the "center" of these nodes (bull's eye $S_{nc}$), which has minimal average distance to them. In addition, as discussed in Section 1, we also want to take the original reference model $S$ (solid node) into account, such that we can control the difference between the newly discovered reference model and the original one. Basically, this requires us to balance two forces: one is to bring the newly discovered reference model closer to the variants (i.e., to the bull's eye $S_{nc}$ at right) than the old one; the other one is to "move" the discovered model not too far away from original model $S$ (the solid node at left) such that it does not differ too much from the original one. Process designer obtain the flexibility to balance these two forces, i.e., they are able to discover a model (e.g., $S_c$), which is closer to the variants than the old one but which is still within a limited distance to the latter. Clearly, the change operations applied first to the (original) reference model should be more important (i.e., reduce the distance between the reference model and the variants more) than the ones positioned at end. Consequently, if we ignore the less relevant changes, we will not influence the overall distance reduction between reference model and variants too much.

Our heuristic algorithm works as follows:

1. We use original reference model $S$ as starting point.
2. We search for all neighboring process models with distance 1 to the currently considered reference process model. If we are able to find a better model $S'$ among these candidate models (i.e., one which has lower average weighted distance to the given collection of variants than $S$), we replace $S$ by $S'$.
3. Repeat Step 2 until we either cannot find a better model or the maximally allowed distance between original and new reference model is reached. Finally, $S'$ corresponds to our discovered reference model $S_c$.

If we do not set any search limitation, our heuristic algorithm is also able to find the "center" of the variants (i.e., $S_{nc}$). This implies that it can be also applied to scenarios where there only exists a collection of variants, but the original reference model is not known. In this case, we can randomly select a variant $S_i$ as starting point and search unlimitedly until we find the "center", i.e., the model with minimal average weighted distance to the collection of variants.

Generally, most important for any heuristic search algorithm are two aspects: the *heuristic measure* and *the algorithm* that uses heuristics to search the state space. Section 4 introduces the **fitness function** which measures the quality of a particular candidate model; i.e., it allows us to approximately evaluate how close such candidate model is to the given variants. Section 5 then introduces a **best-first search** algorithm to search the state space; i.e., this algorithm illustrates how to search for a next candidate process model.

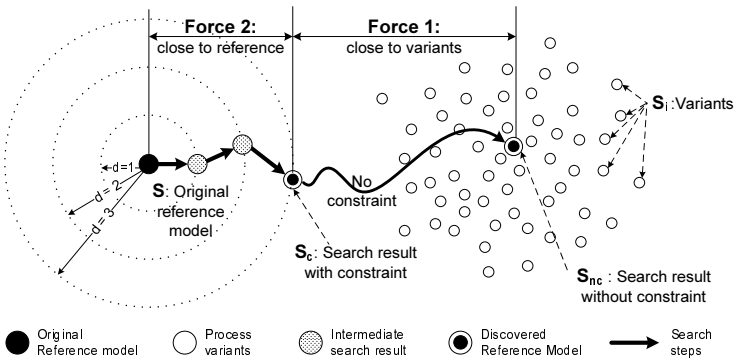



**Fig. 4.** Heuristic search approach

## 4 Fitness Function of Heuristic Search Algorithm

Generally, any fitness function of a heuristic search algorithm should be quickly computable. Since search space may become very large, we must be able to make a quick decision on which path to choose next. Average weighted distance can not be used as fitness function since complexity for computing it is $\mathcal{NP}$. In this section we introduce a fitness function, which can be used to approximately measure the "closeness" between a candidate model and the collection of variants. In particular, it can be computed in polynomial time. Like in most heuristic search algorithms, the chosen fitness function is a "reasonable guessing" rather than a precise measurement. Therefore, in Section 7 we will investigate how the fitness function is correlated with the average weighted distance.

### 4.1 Activity Coverage

For a candidate process model $S_c = (N_c, E_c, \ldots) \in \mathcal{P}$, we first measure to what degree its activity set $N_c$ covers the activities that occur in the considered collection of variants. We denote this measure as *activity coverage* $AC(S_c)$ of $S_c$.

Before we can compute activity coverage, we first need to determine the frequency with which each activity $a_j$ appears within the collection of variants.

**Definition 4 (Activity frequency).**
*Let $S_i = (N_i, E_i, \ldots) \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of variants with weights $w_i$ and activity sets $N_i$. For each $a_j \in \bigcup_{i=1}^{n} N_i$, we define $g(a_j)$ as relative frequency with which $a_j$ appears within the given variant collection. Formally:*

$$g(a_j) = \sum_{S_i : a_j \in N_i} w_i \tag{1}$$

Table 1 shows the frequency of each activities contained in any of the variants in our running example; e.g., `X` is present in 80% of the variants (i.e., in $S_1$, $S_3$, $S_4$, $S_5$, and $S_6$), while `Z` only occurs in 20% of the cases (i.e., in $S_5$).

| Activity | A | B | C | D | E | F | G | H | I | J | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $g(a_j)$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0.8 | 0.65 | 0.2 |

**Table 1.** Frequency of each activity within the given variant collection

**Definition 5 (Activity coverage).** *Let $M = \bigcup_{i=1}^{n} N_i$ be the set of activities which are present in at least one of the variants. Let further $N_c$ be the activity set of a candidate process model $S_c$. Given activity frequency $g(a_j)$ of each $a_j \in M$, we can compute activity coverage $AC(S_c)$ of model $S_c$ as follows:*

$$AC(S_c) = \frac{\sum_{a_j \in N_c} g(a_j)}{\sum_{a_j \in M} g(a_j)} \tag{2}$$

The value range of $AC(S_c)$ is $[0, 1]$. Let us take original reference model $S$ as candidate model. It contains activities `A, B, C, D, E, F, G, H, I`, and `J`, but does not contain `X, Y` and `Z`. Therefore, its activity coverage $AC(S)$, which represents how much it covers the activities in the variant collection, is 0.858.

### 4.2 Structure Fitting

Though $AC(S_c)$ measures how representative the activity set $N_c$ of a candidate model $S_c$ is with respect to a given variant collection, it does not say anything about the structure of the candidate model. We therefore introduce *structure fitting* $SF(S_c)$ as another measurement. It measures to what degree a candidate model $S_c$ structurally fits to the given collection of variants $S_i$.

We first sketch a method which allows to represent a process model $S$ as *order matrix*. Based on this, we introduce *aggregated order matrix* which allows to represent a collection of process variants as matrix. In addition, we introduce the *coexistence matrix* which shows the importance of the order relations. Finally, we describe how to measure structure fitting $SF(S_c)$ of a candidate model $S_c$.

**Representing Process Models as Order Matrices** One key feature of our ADEPT change framework is to maintain the structure of the unchanged parts of a process model [25]. For example, when deleting an activity this neither influences 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 pair of activities; i.e. for a given process model $S = (N, E, \ldots) \in \mathcal{P}$, we examine for 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 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. 6):

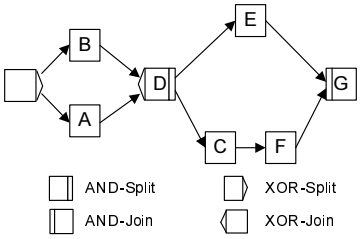
**Definition 6 (Order matrix).** *Let $S = (N, E, \ldots) \in \mathcal{P}$ be a process model with $N = \{a_1, a_2, \ldots, a_n\}$. Let further $\mathcal{T}_S$ denote the set of all traces producible*

*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. 5 gives an example. Besides control edges, which express direct predecessor-successor relationships, the depicted process model $S$ contains different control connectors: AND-Split, AND-Join, XOR-Split and XOR-join. The depicted order matrix represents all these 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 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 [17]. Analyzing its order matrix (cf. Def. 6) is then sufficient for analyzing the corresponding process model.




a) Process model S

| | 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 | |

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

b) Order matrix of S

**Fig. 5.** a) Process model and b) related order matrix

**Aggregated Order Matrix** For a given collection of process variants, first, we compute the order matrix of each process variant (cf. Def. 6). Regarding our

running example from Fig. 3, we need to compute six order matrices (cf. Fig. 6). Note that we only show a partial view of the order matrices here, (activities `H, I, J, X, Y` and `Z`) due to space limitations. Afterwards, we analyze the order relation for each pair of activities considering all order matrices derived before. As the order relation between two activities might be not the same in all order matrices, this analysis does not result in a fixed relation, but provides a distribution for the four types of order relations (cf. Def. 6). Regarding our example, in 65% of all cases, `H` is a successor of `I` (as in $S_2$, $S_3$, $S_4$ and $S_6$), in 25% of all cases `H` is a predecessor of `I` (as in $S_1$), and in 10% of all cases `H` and `I` are contained in different branches of an XOR block (as in $S_4$) (cf. Fig. 6). Generally, to a collection of process variants we can define the order relation between activities `a` and `b` as 4-dimensional vector $V_{ab} = (v^0_{ab}, v^1_{ab}, v^*_{ab}, v^-_{ab})$. Each field then corresponds to the frequency of the respective relation type ('0', '1', '*' or '-') as specified in Def. 6. Take our running example and consider Fig. 6. Here, $v^1_{\texttt{HI}}$ corresponds to the frequency of all cases with activities `H` and `I` having order relationship '1', i.e., all cases for which `H` precedes `I`; we obtain $V_{\texttt{HI}} = (0.65, 0.25, 0, 0.1)$.

Formally, we define an *aggregated order matrix* as follows:

**Definition 7 (Aggregated Order Matrix).**
*Let $S_i = (N_i, E_i, \ldots) \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants with activity sets $N_i$. Let further $A_i$ be the order matrix of $S_i$, and let $w_i$ represent the number of process instances being executed based on $S_i$. The **Aggregated Order Matrix** of all process variants is defined as 2-dimensional matrix $V_{m \times m}$ with $m = |\bigcup N_i|$ 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 $v^\tau_{jk}$ expresses to what percentage, activities $a_j$ and $a_k$ have order relation $\tau$ within the collection of process variants $S_i$. Formally: $\forall a_j, a_k \in \bigcup N_i, a_j \neq a_k : v^\tau_{jk} = \frac{\sum_{A_{i_{jk}} =' \tau'} w_i}{\sum_{a_j, a_k \in N_i} w_i}$.*

Fig. 6 shows the aggregated order matrix $V$ for the process variants from Fig. 3. Again, due to space limitations, we only consider order relations for activities `H, I, J, X, Y`, and `Z`. Generally, in an aggregated order matrix, the 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.

**Importance of the Order Relations** Generally, the order relations computed by an aggregated order matrix may be not equally important. For example, relationship $V_{\texttt{HI}}$ between `H` and `I` (cf. Fig. 6) would be more important than relation $V_{\texttt{HZ}}$, since activities `H` and `I` appear together in all six process variants while activities `H` and `Z` only show up together in variant $S_5$ (cf. Fig. 3). We therefore define co-existence matrix $CE$ in order to represent the importance of the different order relations occurring within an aggregated order matrix $V$.

**Definition 8 (Coexistence Matrix).**

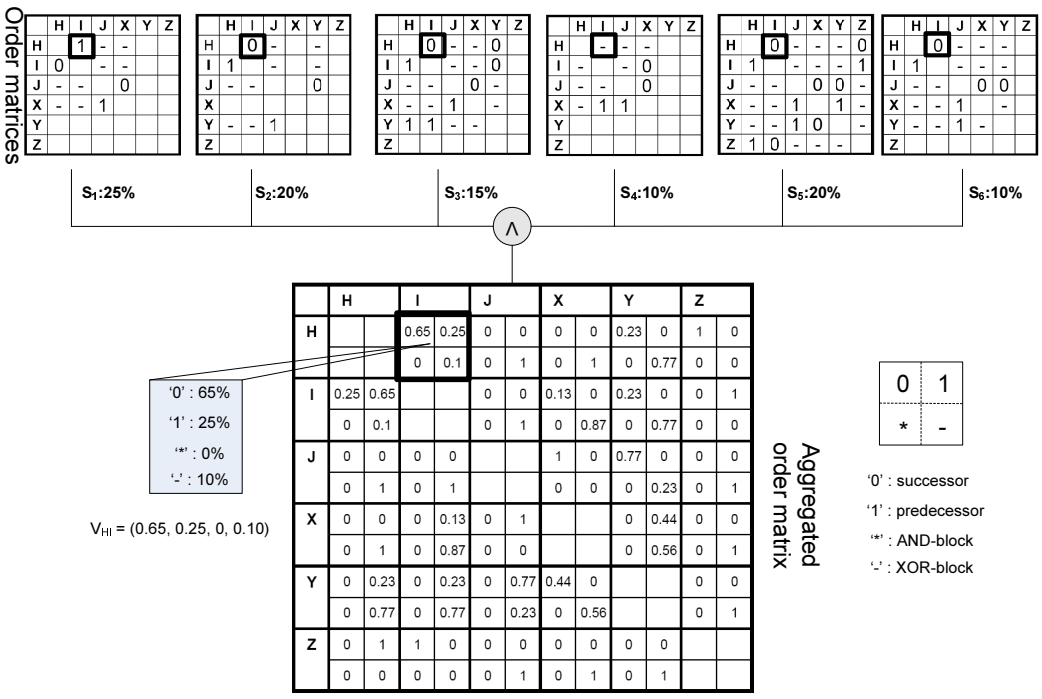



**Fig. 6.** Aggregated order matrix based on process variants

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

Regarding our running example, Table 7 shows the corresponding coexistence matrix. Again, due to space limitations, we only depict the coexistence matrix for activities `H, I, J, X, Y`, and `Z`. For instance, $CE_{\texttt{HI}} = 1$ and $CE_{\texttt{HZ}} = 0.2$. This indicates that order relation between `H` and `I` is more important than the one between `H` and `Z`.

**Structure Fitness of a Candidate Process Models** Since we can represent a candidate process model $S_c$ by its corresponding order matrix $A_c$ (cf. Def. 6), we determine the structure fitting $SF(S_c)$ between $S_c$ and the variants (cf. subsection 4.2) by measuring how similar order matrix $A_c$ and aggregated order matrix $V$ (representing the variants) are. We can compute $S_c$ by measuring the order relations between every pair of activities in $A_c$ and in $V$.

| | H | I | J | X | Y | Z |
|---|---|---|---|---|---|---|
| H | | 1 | 1 | 0.8 | 0.65 | 0.2 |
| I | 1 | | 1 | 0.8 | 0.65 | 0.2 |
| J | 1 | 1 | | 0.8 | 0.65 | 0.2 |
| X | 0.8 | 0.8 | 0.8 | | 0.45 | 0.2 |
| Y | 0.65 | 0.65 | 0.65 | 0.45 | | 0.2 |
| Z | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | |

**Fig. 7.** Pairwise co-existance of activities

For example, consider reference model $S$ as candidate process model $S_c$ (i.e., $S_c = S$). A partial view of the corresponding order matrix $A$ is depicted in Fig. 8. Obviously, $A_{\texttt{HI}}$ ="0" holds, i.e., `H` is successor of `I` in model $S$ (cf. Fig. 8). Consider now aggregated order matrix $V$. Here the order relation between activities `H` and `I` is represented by the 4-dimensional vector $V_{\texttt{HI}} = (0.65, 0.25, 0, 0.1)$. If we now want to compare how close $A_{\texttt{HI}}$ and $V_{\texttt{HI}}$ are, we first need to build an aggregated order matrix $V^c$ purely based on our candidate process model $S_c$ ($S$ in our case). Fig. 8 shows both the order matrix $A_c$ and the "calculated" aggregated order matrix $V^c$ of process model $S_c$ ($S_c = S$). As order relation between `H` and `I` in $V^c$, we obtain $V^c_{\texttt{HI}} = (1, 0, 0, 0)$, i.e., `H` is always a successor of `I`. We now can compare $V_{\texttt{HI}}$ (which represents the variants) with $V^c_{\texttt{HI}}$ (which represents the reference model).

We use Euclidean metrics $f(\alpha, \beta)$ to measure the closeness between two vectors $\alpha = (x_1, x_2, ..., x_n)$ and $\beta = (y_1, y_2, ..., y_n)$:

$$f(\alpha, \beta) = \frac{\alpha \cdot \beta}{|\alpha| \times |\beta|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{3}$$

$f(\alpha, \beta) \in [0, 1]$ computes the cosine value of the angle $\theta$ between vectors $\alpha$ and $\beta$ in Euclidean space. If $f(\alpha, \beta) = 1$ holds, $\alpha$ and $\beta$ exactly match in their directions; $f(\alpha, \beta) = 0$ means, they do not match at all. Regarding our running example, we obtain $f(V_{\texttt{HI}}, V^c_{\texttt{HI}}) = 0.848$. This number indicates high similarity between the order relations of the candidate process model with respect to `H` and `I` and the ones captured by the variants.

Based on (3) which measures *similarity* between the order relations, and Coexistence matrix $CE$ (cf. Def. 8) which measures *importance* of the order relations, we can define structure fitness $SF(S_c)$ of candidate model $S_c$ as follows:

**Definition 9 (Structure Fitness).** *Let $S_i = (N_i, E_i, \ldots) \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants with activity sets $N_i$. Let further $CE$ be the*

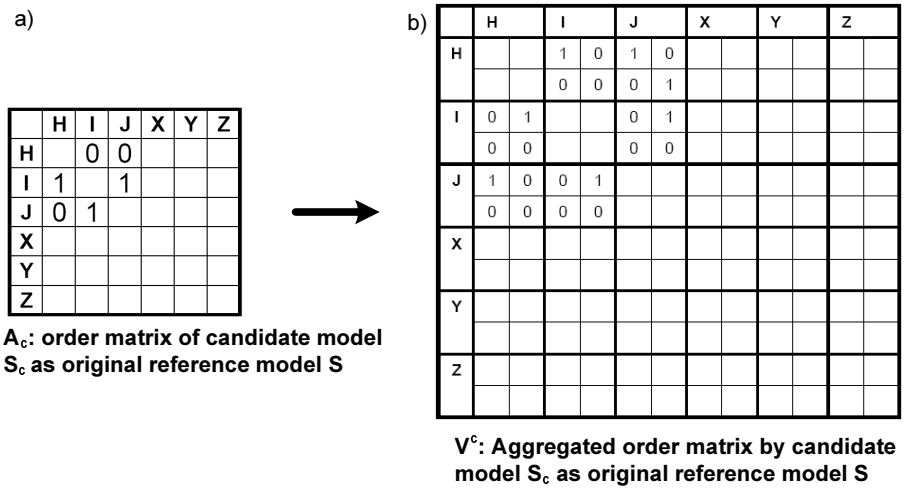



**Fig. 8.** Order matrix $A_c$ and aggregated order matrix $V^c$ constructed by candidate model $S_c = S$

*coexistence matrix, and $V$ be the aggregated order matrix of the collection of variants. For candidate model $S_c$, let $m = |N_c|$ corresponds to the number of activities in $S_c$; let further $V^c$ be aggregated order matrix of $S_c$. we can compute* structure fitness $SF(S_c)$ *as follows:*

$$SF(S_c) = \frac{\sum_{j=1}^{m} \sum_{k=1,k\neq j}^{m} (f(V_{a_j a_k}, V^c_{a_j a_k}) \times CE_{a_j a_k})}{m \times (m-1)} \tag{4}$$

For every pair of activities $a_j, a_k \in N_c, j \neq k$, we first compute similarity of corresponding order relations (as captured by $V$ and $V_c$) by means of $f(V_{a_j a_k}, V^c_{a_j a_k})$, and second the importance of these order relations by $CE_{a_j a_k}$. Structure fitness $SF(S_c) \in [0, 1]$ of candidate model $S_c$ then equals the average of the similarity multiplied by the importance of every order relation. Regarding our example from Fig. 3, structure fitting $SF(S)$ of the original reference model $S$ corresponds to 0.749.

### 4.3 Fitness Function

So far, we have described the two measurements activity coverage $AC(S_c)$ and structure fitting $SF(S_c)$ to evaluate fitness of a candidate model $S_c$. While $AC(S_c)$ measures to what degree activities occurring in the variants are covered by the candidate model $S_c$, $SF(S_c)$ measures to what degree $S_c$ fits structurally to the variants.

**Definition 10 (Fitness).** *For candidate model $S_c$, let $AC(S_c)$ be the activity cover of $S_c$ and let further $SF(S_c)$ be the structure fitness of $S_c$. We compute*

*the* fitness $Fit(S_c)$ *of a candidate model* $S_c$ *as follows:*

$$Fit(S_c) = AC(S_c) \times SF(S_c) \tag{5}$$

As $AC(S_c) \in [0,1]$ and $SF(S_c) \in [0,1]$, value range of $Fit(S_c)$ is [0,1] as well. Fitness value $Fit(S_c)$ indicates how "close" a candidate model $S_c$ is to the given collection of variants. If $Fit(S_c) = 1$ holds, candidate model $S_c$ will fit perfectly to the variants; i.e., no additionally adaptation will be needed. Otherwise, further adaptations might be required. The higher $Fit(S_c)$ is, the closer $S_c$ will be to the variants and the less configuration efforts will be needed. Regarding our example from Fig. 3, fitness value $Fit(S)$ of the original reference process model $S$ is $Fit(S) = AC(S) \times SF(S) = 0.858 \times 0.749 = 0.643$.

As fitness of a candidate model $S_c$ is evaluated by activity coverage $AC(S_c)$ multiplied by structure fitting $SF(S_c)$, we can automatically balance the number of activities to be considered in candidate model $S_c$. If too many activities of low relevance (i.e., activities which only appear in a limited number of instances; e.g., activity `Z` in our example) are considered in the candidate model, we will obtain a high $AC(S_c)$ value. However, $SF(S_c)$ then possibly will decrease since coexistence values (cf. Def. 8) of such less relevant activities are often very low (cf. Fig. 7). On the contrary, if $S_c$ contains too few activities, $SF(S_c)$ can potentially be very high, while $AC(S_c)$ will be too low in order to qualify $S_c$ as a good candidate model. Therefore, a high value for $Fit(S_c)$ does not only mean that $S_c$ structurally fits well to the variants, but also that a reasonable number of activities is considered in the candidate model.

The complexity of computing $Fit(S_c)$ is polynomial. To be more precise, let $n$ be the number of variants and let $m = |\bigcup_{i=1}^{n} S_i|$ be the total number of activities in the variants. The complexity to compute activity frequency (cf. Def. 4) is $\mathcal{O}(mn)$ and the complexity to compute aggregated order matrix $V$ (cf. Def. 7) is $\mathcal{O}(2m^2n)$. Based on this, the complexity to compute the fitness function is $\mathcal{O}(m + 2m^2)$. Note that it is significantly lower than $\mathcal{NP}$ level complexity as needed for computing the average weighted distance.

As already discussed, the fitness function $Fit(S_c)$ is only a "reasonable guess" rather than an exact measurement (like average weighted distance). Therefore, we analyze the performance of our fitness function later in Section 7.

## 5 Constructing the Search Tree

We have sketched the basic steps of our heuristic mining algorithm in Section 3.2. In Section 4, we have then discussed how to evaluate a candidate process model $S_c$ based on fitness function $Fit(S_c)$. In this section, we show how we can find adequate candidate process models. For that purpose we present a *best-first* algorithm which allows us to construct a search tree in such a way that we can find the best candidate model in the search space. Section 5.1 first provides an overview on how we construct the search tree by comparing the result models from changing each activity $a_j \in N_c$ in the candidate model $S_c$. In Section 5.2, we further describe in what ways we can adapt a particular activity $a_j$ in order

to find all *kids* of a given candidate process model $S_c$, i.e., all models which have distance one to $S_c$. In Section 5.3, we provide search results and an evaluation of our example models from Fig. 3. Finally, Section 5.4 provides a prototype implementation of the described algorithm.

### 5.1 The Search Tree

Let us revisit Fig. 4 which gives a general overview of our heuristic search approach. Starting with the current candidate model $S_c$, in each iteration, we search for its "neighbors" (i.e., process models which have distance 1 to $S_c$) to see whether we still can find a better candidate model $S'_c$ with higher fitness value. Generally, we can construct a neighbor model for a given process model $S_c = (N_c, E_c, \ldots)$ by applying one insert, delete, or move operation to $S_c$. All activities $a_j \in \bigcup N_i$ ($N_i$ corresponds to the activity set of variant $S_i$), which have appeared in the variant collection, are candidate activities for change. Obviously, an insert operation adds an activity $a_j \notin N_c$ to $S_c$, while the other two operations delete or move an activity $a_j$ already present in $S_c$ (i.e., $a_j \in N_c$). Generally, numerous process models may result by changing one particular activity $a_j$ on $S_c$. Note that the positions where we can insert ($a_j \notin N_c$) or move ($a_j \in N_c$) activity $a_j$ can be numerous.

Section 5.2 provides a details on how to find all process models resulting from the change of one particular activity $a_j$ on $S_c$. In this section, first of all, we assume that we have already found the best process model (i.e., with highest fitness value) from all the models resulting from changing a particular activity $a_j$ on $S_c$. We denote this model as the **best kid** $S^j_{kid}$ of $S_c$ when changing $a_j$.

Our basic idea is to create all neighbor models, to evaluate each of them with the fitness function, and to finally choose the one with highest fitness value. We present a best-first algorithm to perform our heuristic variant mining (cf. Algorithm 1). To illustrate this algorithm, we use the search tree depicted in Fig. 9.

Our search algorithm starts with setting the original reference model $S$ as the initial state, i.e., $S_c = S$ (see the node at the top of Fig. 9). We further define $AS$ as active activity set, which contains all activities available for change. At the beginning, $AS = \{a_j | a_j \in \bigcup_{i=1}^{n} N_i\}$ contains all activities that appear in at least one process variant $S_i$. For each activity $a_j \in AS$, we determine the corresponding best kid $S^j_{kid}$ of $S_c$ when changing $a_j$ on $S_c$ (i.e., when deleting, moving or inserting $a_j$). If the best kid $S^j_{kid}$ has higher fitness value than $S_c$, we mark it with lines; otherwise, we mark it white and remove $a_j$ from $AS$ (cf. Fig. 9).[7] Afterwards, we find the best one among all the best kids $S^j_{kid}$, i.e., the

[7] For all nodes marked as white, we remove them from active activity set $AS$. Consequently, we stop searching the best kid when changing them. In principle, it is still possible that when changing them later (i.e., based on another candidate model $S'_c$), we can find a better resulting model. However, such chance is very low due to the fact that we have already enumerated all possible solutions by changing such activity on $S_c$. We therefore remove them from $AS$ in order to reduce the search space.

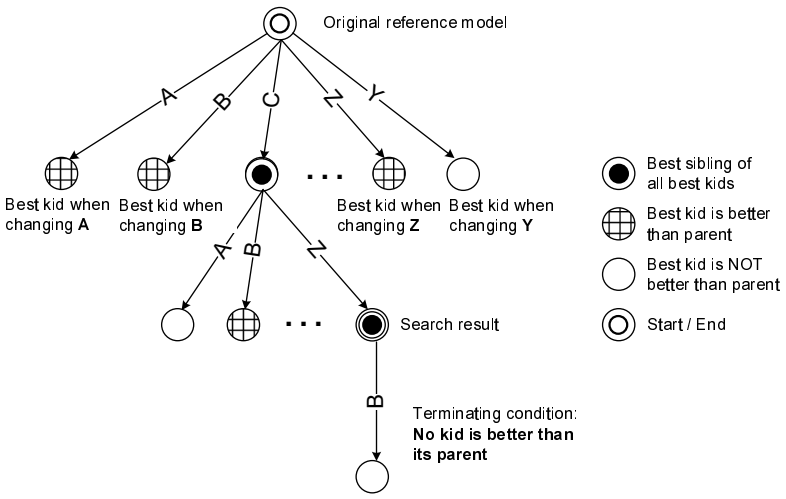



**Fig. 9.** Construct the search tree

one with highest fitness value. We denote this model as best sibling $S_{sib}$ and we mark the corresponding activity $a_s$ accordingly. Since this model $S_{sib}$ is the best model we are able to obtain by one change operation on candidate model $S_c$, we set $S_{sib}$ as the first intermediate search result and replace $S_c$ by $S_{sib}$ for further search (cf. Fig. 9, $S_{sib}$ are marked as bull's eyes). Note that we also remove $a_s$ from $AS$ since this activity has now been already considered for change.

The described search method goes on iteratively, until termination condition is met, i.e., we either can not find a better model, or the allowed search distance is reached. The final search result $S_{sib}$ corresponds to our discovered reference model $S'$ (the node marked by a bull's eye and circle in Fig. 9).

### 5.2 Options for Changing One Particular Activity

Section 5.1 has shown how to construct a search tree by comparing the best kids $S^j_{kid}$. This section discusses how to find such best kid $S^j_{kid}$ when changing a particular activity $a_j$, i.e., we discuss how to find the "neighbors" of a candidate model $S_c$ by performing one high-level change operation (cf. Def. 1) on $a_j$. The best kid $S^j_{kid}$ is consequently the one with highest fitness value among all considered models.

Regarding a particular activity $a_j$, we consider three type of basic change operations: *delete, move* and *insert* activity (cf. Section 5.1). The neighbor model resulting through deletion of an activity $a_j \in N_c$ can be easily determined by removing $a_j$ from the process model and the corresponding order matrix; furthermore, movement of an activity can be simulated by its deletion and subsequent re-insertion at the desired position. Thus, the basic challenge in finding neighbors of a candidate model is to apply one activity *insertion* such that *block*

```
   input  : A process model S; a collection of process variants
            S_i = (N_i, E_i, ...), i = 1, ..., n; allowed search distance d ;
   output: Resulting process model S'
 1 AS = ⋃_{i=1}^{n} N_i                    /* Define AS as active activity set */;
 2 S_c = S                                 /* Define initial candidate model */;
 3 t = 1                                   /* Define initial search step */ ;
 4 while |AS| > 0 and t ≤ d                /* Search condition */;
 5 do
 6     S_sib = S_c                         /* Set S_c as initial S_sib */ ;
 7     Define a_s as the selected activity ;
 8     foreach a_j ∈ AS do
 9         S_kid = FindBestKid(S_c) ;
10         if Fitness(S_kid) > Fitness(S_c) then
11             if Fitness(S_kid) > Fitness(S_sib) then
12                 S_sib = S_kid ;
13                 a_s = a_j ;
14             end
15         else
16             AS = AS \ {a_j} ;
                                  /* Best kid not better than its parent */
17         end
18     end
19     if Fitness(S_sib) > Fitness(S_c) then
20         S_c = S_sib ;                   /* Initiate next iteration */ ;
21         AS = AS \ {a_s} ;
22     else
23         break ;
24     end
25     t = t+1 ;
26 end
```

**Algorithm 1**: Heuristic search algorithm for variant mining

*structuring* and *soundness* of the resulting model can be guaranteed. Obviously, for a particular activity $a_j$, the positions where we can (correctly) insert it into candidate model $S_c$ are the subjects of our interest. Inserting $a_j$ at a (correct) position within $S_c$ results in one neighbor model. Therefore, finding all neighbors first requires finding all valid positions where we can correctly insert $a_j$ in $S_c$.

Fig. 10 provides one example. Given a process model $S$, we would like to find all process models that may result when inserting activity `X` into $S$. We apply the following two steps to "simulate" the insertion of an activity:

1. First, we enumerate all possible blocks the candidate model $S$ contains. A block can be an atomic activity, a self-contained part of the process model, or the process model itself (cf Algorithm 2 for an algorithm enumerating all possible blocks of a process model). Note that the number of possible

candidate blocks can become very large; e.g., hundreds of potential blocks may exist for a process model containing 50 activities.

2. After having determined all blocks of the current model we now can simulate all possible insertions of activity `X`. For this purpose, we can cluster `X` with each block and position it in relation to this block, i.e., we can set order relation $\tau \in \{0, 1, *, -\}$ between `X` and the selected block $B$ (cf. Def. 6). This way, we insert `X` to the respective position such that it forms another block together with $B$; or in another word, we replace block $B$ by another block $B'$ which contains $B$ and `X`. Consequently, we obtain a neighbor model $S'$ by inserting `X` into $S$.

Following these two 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 add this activity to the position where it can form a bigger block together with the selected one, i.e., we replace a self-contained block of a process model by a bigger one. Consider our example from Fig. 10a. Among the determined blocks, we can find the sequential block defined by activities `C` and `D` (step 1). Then we can cluster activity `X` with this block using order relation $\tau =$ "0" for example (step 2). Consequently, we obtain $S'$ as one neighbor of $S$ (cf. Fig. 10). In the following, we describe these two steps in detail.

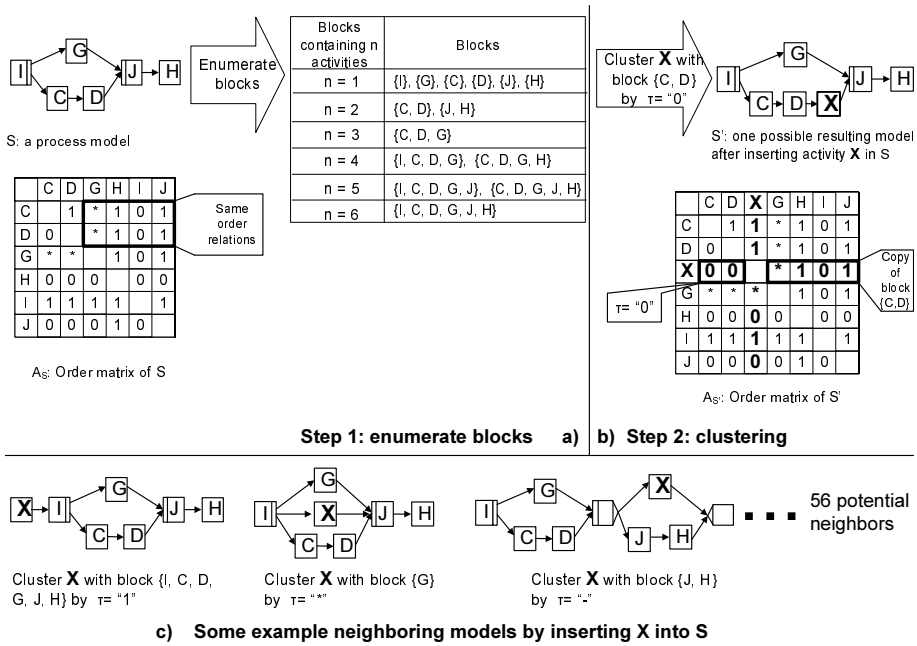



**Fig. 10.** Finding the neighboring models by inserting `X` into process model $S$

**Step 1: Block-enumerating Algorithm** We now present an algorithm to enumerate all possible blocks of a process model $S$.

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 $[\forall a_k \in N \setminus \{a_i, a_j\} : A_{ik} = A_{jk}]$ holds; i.e., two activities can form a block if and only if they have exactly same order relations to the remaining activities. Consider our example from Fig. 10a. Here activities `C` and `D` can form a block since they have same order relations to the remaining activities `G, H, I` and `J`.

**input** : A process model $S = (N, E, \ldots)$ and its order matrix $A$
**output**: A set $BS$ with all possible blocks

Define $BS_x$ be a set of blocks containing blocks with $x$ activities. $x = (1, \ldots, n)$;
Define each activity $a_i$ as a block $B$, $i = (1, \ldots, n)$ ;
$BS_1 = \{B_1, \ldots, B_n\}$. /* initial state */ ;
**for** $i = 2$ **to** $n$ /* Compute $BS_i$ */;
**do**
  let j = 1; let k = i;
  **while** $j \leq k$ **do**
    k = i - j /* A block containing $k$ activities can only be obtained by merging blocks containing $i$ and $j$ activities */;
    **foreach** $(B_j, B_k) \in BS_j \times BS_k$;
    merge = TRUE /* judge whether $B_j$ and $B_k$ can form a block */;
    **do**
      **if** $B_j \bigcap B_k = \emptyset$ /* Disjoint? */ **then**
        **foreach** $(a_\alpha, a_\beta, a_\gamma) \in B_j \times B_k \times (N \setminus B_j \bigcup B_k)$ **do**
          **if** $A_{\alpha\gamma} \neq A_{\beta\gamma}$ **then**
            merge = FALSE /* two blocks con merge only if they show same order relations to the activities out side the two blocks */;
            break ;
          **end**
        **end**
      **else**
        merge = FALSE;
      **end**
      **if** merge = TRUE **then**
        $B_p = B_j \bigcup B_k$;
        $BS_i = BS_i \bigcup B_p$;
      **end**
    **end**
    j = j + 1 ;
  **end**
**end**
$BS = \bigcup_{x=1}^{n} BS_x$

**Algorithm 2**: Block enumerating algorithm

The block-enumerating algorithm is depicted in Algorithm 2. Let us first define $BS_x$ as the set containing all blocks comprising exactly $x$ activities. In its initial state, each activity forms a single block by its own (line 2) and consequently we obtain $BS_1$ (line 3). The algorithm starts by computing $BS_2$ (blocks containing 2 activities) and continues iteratively to compute $BS_i$ until it reaches its upper boundary $i = n$. In each iteration, we can determine a block containing $x$ activities by merging two disjoint blocks containing $j$ and $k$ activities respectively ($i = j+k$) (line 8). For example, a block containing 2 activities can only be obtained by merging two blocks of which each contains 1 activity. Or we can only obtain a block containing 5 activities by merging two disjoint blocks containing either 1 and 4 activities respectively or 2 and 3 activities respectively (line 4 - 29). Lines 9 to 26 check whether or not two blocks $B_j$ and $B_k$ can be merged together. This is possible iff any activities $a_\alpha \in B_j$ and $a_\beta \in B_k$ show same order relations to the remaining activities outside the two blocks. Otherwise (lines 14 -17), $B_j$ and $B_k$ cannot form a block (i.e., merge = $FALSE$). Until we obtain all sets of blocks $BS_x$ with $x = 1, \ldots, n$ activities per block, we can define set $BS$ as $BS = \bigcup_{x=1}^{n} BS_x$. $BS$ consequently corresponds to all blocks, process model $S$ contains (line 28). In our context, we consider each block as a set rather than a process model, since structure of these blocks is already clear in $S$.[8]Consider the example from Fig. 10a. For the given a process model $S$, all possible blocks are enumerated. For example, as activities `C` and `D` show same order relations in respect to the remaining activities in order matrix $A_s$, they may form a block. Or, block {`C, D`} and block {`G`} show same order relations in respect to remaining activities `H, I` and `J`; therefore they can form a bigger block {`C, D, J`}. As $S$ contains 6 activities, its blocks are organized in 6 groups containing blocks of different sizes.

**Step 2: Cluster Inserted Activity with One Block** In Step 1, we have shown how to enumerate all possible blocks for a given candidate model $S_c$. Based on this, we describe where we can insert a particular activity $a_j$ in $S_c$ such that we obtain a sound and block-structured model again.

Assume that we want to insert activity `X` in $S$ (cf. Fig. 10). To ensure the block structure of the resulting model, we "cluster" `X` with an enumerated block, i.e., we replace one of the previously determined blocks $B$ by a bigger block $B'$ containing $B$ as well as `X`. In the context of this clustering, we set the order relation between block $B$ and inserted activity `X` as $\tau \in \{0, 1, *, -\}$ (see Def. 6), i.e., the order relations between `X` and all activities of $B$ are defined by $\tau$. One example is given in Fig. 10b, where the inserted activity `X` is clustered with block {`C, D`} by order relation $\tau$ = "0", i.e., we set `X` as successor of the

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

sequence block containing C and D. To realize this clustering, we have to set the order relations between X on the one hand and activities C and D from the selected block on the other hand to "0". Furthermore, order relations between X and the remaining activities are the same as for C and D respectively. Afterwards these three activities form a new block {C, D, X} replacing the old one (i.e., {C, D}). This way, after inserting X into $S$, we obtain a new sound and block-structured process model $S'$.

Fig. 10b shows only one resulting model $S'$ which we obtain when inserting X in $S$. Obviously, $S'$ is not the only neighboring models in this context since we can insert X at different positions in $S$; i.e., for each block $S$ enumerated in Step 1, we can cluster it with X by any one of the four order relations $\tau \in \{0, 1, *, -\}$. Regarding our example from Fig. 10, $S$ contains 14 blocks. Consequently, the number of models that may result when inserting X in $S$ equals $14 \times 4 = 56$; i.e., we obtain 56 potential models by inserting X into $S$. Fig. 10c shows some neighboring models of $S$. Note that the 56 resulting models are not necessarily unique, i.e., it is possible that some of them are same. However, this is not an important issue in our context since our fitness function $Fit(S_c)$ can be quickly computed. Therefore, some redundant information will not significantly decrease the performance of our heuristic search algorithm.

### 5.3 Search Result for Our Running Example

Regarding our example from Fig. 3, we now present the search result we obtain when applying our heuristics search algorithm. Fig. 11 does not only show the final resulting model, but all the intermediate process models discovered during the search. Note that in this scenario, we do not set any limitation on the number of search steps, i.e., we allow the algorithm to go as far as possible to find the best reference model.

Fig. 11 shows the evolution of the original reference model $S$. The first operation $\delta_1 = move(S, \texttt{J}, \texttt{B}, endFlow)$ changes $S$ into intermediate result model $R_1$. According to Algorithm 1, $R_1$ constitutes that neighbor model of $S$ which can be derived by applying one valid change operation to $S$ and which shows highest fitness value in comparison to all other neighbor models of $S$. Using $R_1$ as next input for our algorithm, we discover process model $R_2$. In this context, change operation $\delta_2 = insert(R_1, \texttt{X, E, B})$ is applied. Finally, we obtain $R_3$ by performing change $\delta_3 = move(R_2, \texttt{I,D,H})$ on model $R_2$. Since we cannot find a "better" process model by changing $R_3$ anymore, we obtain $R_3$ as final result. Note that if we set constraints on allowed search steps (i.e., we only allow to change original reference model by maximal $d$ change operations), the final search result would be as follows: $R_d$ *if* $d \leq 3$ *or* $R_3$ *if* $d > 3$. We further compare the original reference model $S$ and all (intermediate) search results in Table 2.

We first show the fitness value of all the models in Fig. 11. As our heuristic search algorithm is based on finding process models with better fitness values, we can observe improvements of the fitness values with each search step. The fitness value $Fit(S)$ increases from 0.643 (model $S$) to 0.841 (model $R_1$), and

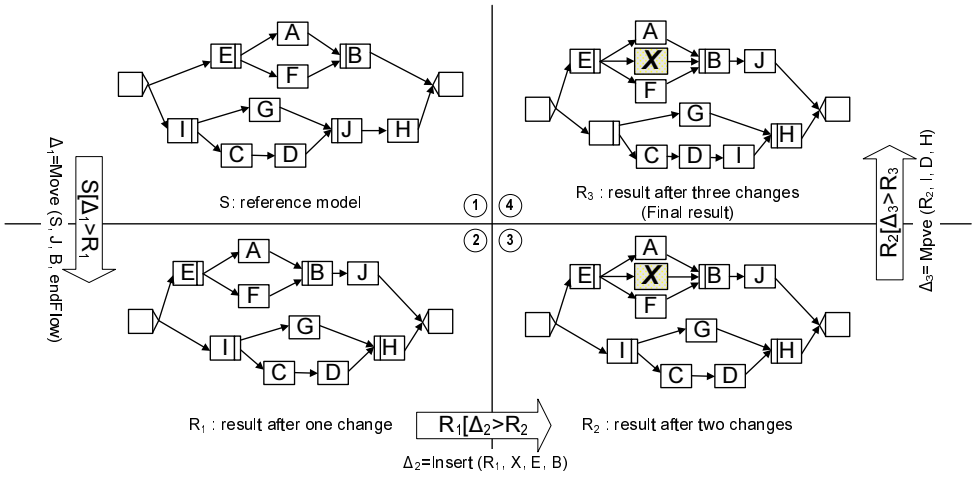



**Fig. 11.** Search result by every change operations

then to 0.854 (model $R_2$). Finally, it reaches 0.872 (model $R_3$). Though such fitness value is only a "reasonable guessing" of how good the result model is, the improvement of the fitness value at least indicates that discovered models is assumed to get better in each iteration.

Still, we need to examine whether or not the discovered process models are indeed getting better. We therefore compute the average weighted distance between the discovered model and the variants, which is a precise measurement in our context. From Table 2, the improvement of average weighted distances after applying the above changes becomes clear, i.e., the average weighted distance drops monotonically from 4 (when considering model $S$) to 2.25 (when considering model $R_3$). Measuring the average weighted distance shows that for the given example, the algorithm performs as expected.

One important reason to design a heuristic search algorithm in our context was to be able to only consider the most relevant change operations, i.e., the important changes (reducing average weighted distance between reference model and variants most) should be discovered at beginning while the trivial ones should be either ignored or be put at the end (cf. Section 1). We therefore additionally evaluate *delta-fitness* and *delta-distance*, which indicate the relative

| | $S$ | $R_1$ | $R_2$ | $R_3$ |
|---|---|---|---|---|
| Fitness | 0.643 | 0.814 | 0.854 | 0.872 |
| Average weighted distance | 4 | 3.2 | 2.6 | 2.25 |
| Delta fitness | | 0.171 | 0.04 | 0.017 |
| Delta Distance | | 0.8 | 0.6 | 0.25 |

**Table 2.** Search result by every change

improvement of fitness values and the reduction of average weighted distance for every iteration of the algorithm. For example, the first change operation $\delta_1$ changes $S$ into $R_1$, and consequently improves fitness value (delta-fitness) by 0.0171 and reduces average weighted distance (delta-distance) by 0.8. Similarly, $\delta_2$ reduces average weighted distance by 0.6 and $\delta_3$ by 0.25. It is obvious that the delta-distance is monotonically decreasing as the number of change operations increases. This indicates that the important changes are performed at beginning of the search, while the less important ones are performed at the end.

Another important feature of our heuristic search is its ability to automatically decide on which activities shall be included in the reference model. A predefined threshold or filtering of the less relevant activities in the activity set are not needed. In our example, `X` is automatically inserted, when `Y` and `Z` are not. The only concern in our heuristic variant mining is to reduce the average weighted distance, i.e., the three change operations (insert, move, delete) are automatically balanced based on their influence on the reduction of average weighted distance. This is also a significant improvement when compared to many other process mining techniques in which preprocessing of trivial activities should be conducted before performing the mining [15, 38].

### 5.4 Proof-of-Concept Prototype

The described approach has been implemented and tested using Java. Figure 12 depicts a screenshot of our prototype. We have used our ADEPT2 Process Template Editor [27] as tool for creating process variants. For each process model, the editor can generate an XML representation with all relevant information (like nodes, edges, blocks) being marked up. We store created variants in a variants repository (cf. Fig. 12) which can be accessed by our mining procedure.

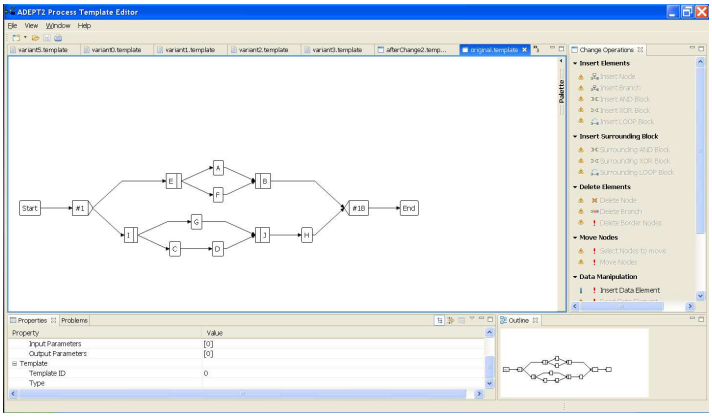



**Fig. 12.** Screenshot of the prototype

The mining algorithm has been developed as stand-alone Java program, independent from the process editor. It can read the original reference model and all process variants, and it can generate the result models according to the XML schema of the process editor. All intermediate search results are also stored and can be visualized using the ADEPT2 editor.

ADEPT2 is a next generation adaptive process management tool, which allows for the flexible execution of process instances. Particularly, the ADEPT2 framework enables ad-hoc changes of single process instances during runtime as well as changes at the process type level and their propagation to running instances if desired and possible [26]. Based on the presented mining algorithm, ADEPT is able to provide full process lifecycle support.

## 6 Simulation Setup

Clearly, using one example to measure the performance of our heuristic mining algorithm is far from being enough. Since computing the average weighted distance is at $\mathcal{NP}$ level, the fitness function, whose calculation only needs polynomial time, is only an approximation of average weighted distance. Therefore, these two measures can NOT be correlated perfectly, i.e., it is not for sure that the improvement of the fitness value will always result in a deduction of average weighted distance. Therefore, the first question is *how the fitness improvement (delta-fitness) correlates with the reduction of average weighted distance (delta-distance)?*

Moreover, we want to analyze whether the algorithm can scale up. Clearly, it will take longer time to find the result if we have to cope with a large collection of variants with dozens or even hundreds of activities, simply because the search space would be significantly larger. What is more important to know is whether the performance will also change when facing large models, i.e., *does the correlation between delta-fitness and delta-distance depend on the size of the model?*

In addition, we are interested in whether it is really true that the most important change operations (i.e., the change operations which largely reduce average weighted distance) are performed at beginning of the search. If this is the case, we do not need to fear too much when setting search limitations or filtering out the change operations performed at the end. Therefore, the third research question is as follows: *To what degree are the important change operations positioned at the beginning of the search steps? I.e., to what degree are delta-fitness and delta-distance monotonically decreasing as the number of change operations increases?*

Finally, we investigate whether we can further improve the performance of our heuristic variant mining algorithm using some other data mining or artificial intelligence techniques, i.e., we try to adopt the concept of "pruning" as commonly used in data mining [36] and artificial intelligence [20]. In our context of variant mining, we can "prune" out the situation when delta-fitness is not nicely correlated with delta-distance, and consequently improve the perfor-

mance of our algorithm by adapting our algorithm to such situation. Therefore, the last research question is as follows: : *How can we improve the performance of our heuristic mining algorithm by adopting the concept of "pruning"?*

We try to answer all these questions using *simulation*, i.e., by generating thousands of data samples, we can provide a statistical answer for these questions.

To summarize, we use simulation to answer the following research questions:

1. *How much is delta-fitness correlated to delta-distance in our heuristic algorithm?*
2. *Can the performance of the algorithm scale up? That is, does the correlation between fitness value and average weighted distance depend on the size of models?*
3. *Is it really true that the important change operations are performed at the beginning? That is, does improvement of the average weighted distance monotonically decreases as the number of change operations increases?*
4. *Would it be possible to further "prune" the data such that we can improve the performance of our heuristic mining algorithm?*

In this section, we describe how we setup the simulation, simulation results themselves are presented in Section 7. In total, we created 72 groups of datasets based on different scenarios. Each of these groups consists of 1 reference model and 100 variants configured out of it. In total, we create 7272 process models for analysis. This section is organized as follows. Section 6.1 describes an algorithm to generate a random reference process model. Section 6.2 describes based on which parameters we can configure a collection of variants out of such randomly created reference model. Section 6.3 summarizes the considered scenarios and describe how we adapt the parameters when creating the respective variants. Finally, Section 6.4 shows how simulations are setup to measure the performance of our heuristic variant mining algorithm.

### 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 this clustering continues iteratively until all the activities (blocks) are clustered together. The detail of our approach is depicted in algorithm 3.

To illustrate how Algorithm 3 works, an example is given in Fig. 13. As input a set of activities {`A,B,C,D,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 cluster them together with a randomly chosen order relation $\tau$ (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

**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 $\tau$ ;
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 3**: Randomly generating a reference model

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. 13 shows the process model we randomly generated as well as the block constructed in each iteration.

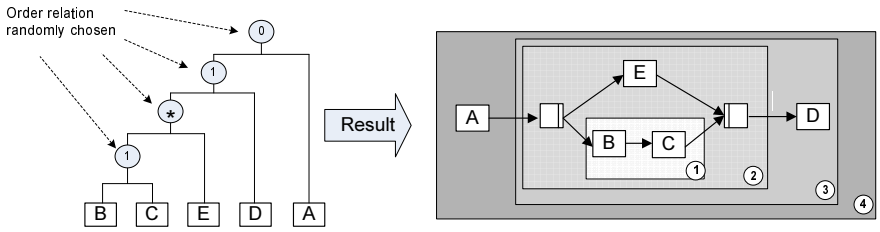



**Fig. 13.** Example of generating random process model

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 [46]. 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%.

### 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 or inserted 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 variants (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 our algorithm, i.e., whether or not the correlation of delta-fitness and delta-distance depends on th size of the models.
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 [17].
3. **Parameter 3 (Activity Occurrence)** One important feature of our heuristic mining algorithm is its ability to automatically decide which activities should or should not be considered in the discovered model. Clearly, activities with low activity frequency (cf. Def. 4) may not be considered. We use this parameter to perform a detailed analysis of this situation.
4. **Parameter 4 (Activity Consistence)** As activity frequency can be quickly computed by scanning the activity sets of the variants, it is also important to know whether the position for one particular activity is consistent. e.g., if an activity is often inserted, we are also interested in whether such activity is always inserted into a particular position. This parameter therefore helps us to examine whether or not the position where activities are inserted or moved to can influence our search algorithm.

As parameter 1 and 2 are easy to understand, we use Fig. 14 to provide a further analyze of parameter 3 and 4.

Consider the situation when an activity $a_j$ is very often inserted when configuring variants (high occurrence), and its inserted position is also very consistent (high consistency). The heuristic algorithm therefore should have a high chance to also insert $a_j$ in the reference model, since such insertion is very often used in configuring each variant (up-right part of Fig 14). On the contrary, if an activity appears in only a few variants (low occurrence) and its positions in those variants are also changing all the time (low consistency), we therefore can ignore such activity (down-left part of Fig. 14) since such change does not repeat often. The difficult part is the rest two parts in the value space (marked with a question mark). It would be difficult to know whether we should insert one activity with high activity frequency but its positions are not very stable. Reason is that even if we insert this activity in the reference model, we also need to move it frequently since its position in the variants are not stable, therefore inserting such activity is not necessarily lead to the reduce of the average weighted distance. On the

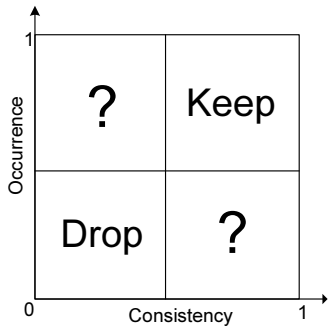



**Fig. 14.** Consistency and occurrence

contrary, if an activity does not appear often in the variants, but its positions in these variants are very consistent, it is also difficult to determine whether or not we should insert this activity in the reference model. If we insert it, we need to often delete it since it does not shows up often in the variants. Therefore, in the following subsections, we will explain how to simulate these situations to cover the value space in our simulation.

### 6.3 Methods for Generating Data Sets

When configuring the variants based on a given reference model, we vary the values of the parameters described in subsection 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 the original reference model and consequently control, in general, the size of process variants. There can be three options:

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

In our simulation, we use the same reference model for the groups containing process models of same size. Reason is that we want to avoid the influence of the randomly generated reference model. According to [21], 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. Note that the variants may have different activity sets than the reference model since we also employ insert and delete operations when configuring a variant.

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

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

For example, for the datasets comprising large-size process variants (i.e., variants with 50 activities), medium-change would mean we need to perform 10 change operations on the reference model to configure a particular process variant. 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 (Activity Occurrence)** We use Fig. 15 to illustrate how we configure parameter 3. The X-axis represents a list of activities and the Y-axis shows their corresponding probability being involved in process configurations.

Given a reference model $S$, let $a_j \in N$, $j = 1 \ldots, m$ be the activities in $S$. Based on parameter 1 and 2, we can decide on by performing how many change operations we are able to configure $S$ into a particular variant $S_i$. For example, if parameter 1 is "large-sized" model ($S$ contains 50 activities) and parameter 2 is "large-change" (change 30% of activities in $S$), we know that we need to change 15 activities to configure one variant out of $S$. Let $x$ be the number of change operations, we can create $x$ new activities $a_x$, $x = n + 1 \ldots n + x$. Then the $m + x$ activities constitute the X-axis in Fig. 15.

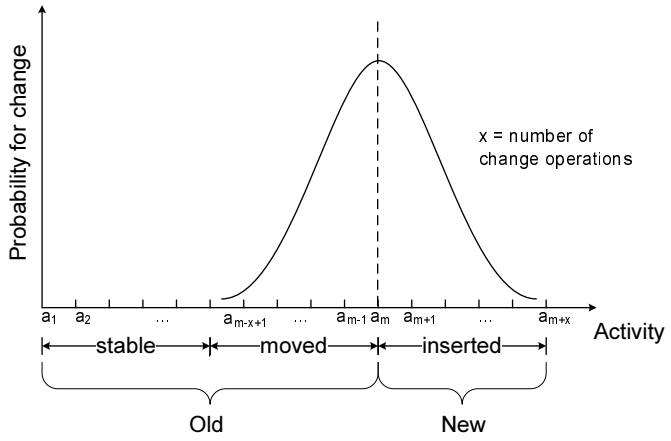



**Fig. 15.** Change occurrence for activities in the reference model

The Y-axis therefore shows the probability each activity is involved in change operations. In order to compare activities with different probability being involved in change operations, we assign each activity with different probability. We use Gaussian distribution to realize the difference. For activity $a_j, j = (m - x + 1) \ldots (m + x)$, the probability it involved in change operations is $\int_{j-1}^{j} \frac{1}{\sqrt{2\pi}} e^{\frac{-(j-m)^2}{2(\frac{x}{3})^2}}$ (i.e.,$\mathcal{X} \sim (m, (\frac{x}{3})^2)$, which means the expected value of the dis-

tribution is $m$ with standard deviation $\frac{x}{3}$. The probability of changing $a_j$ is the integral in the interval [j-1,j) ). Table 3 shows the probability for the groups with parameter 1 being small-sized and parameter 2 being large-change. Note that activity `K, L` and `M`, are not in the original reference model $S$. The purpose of using Gaussian distribution here is only to simulate the situation that activities have different probability being involved in changes. We do *NOT assume* that the probability for activities being involved in change must follow Gaussian distribution or any other distributions.

| Activity | A | B | C | D | E | F | G | H | I | J | K* | L* | M* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Probability | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.022 | 0.136 | 0.341 | 0.341 | 0.136 | 0.022 |

**Table 3.** The probability for each activity being involved in change operations when configuring process variants

After we described how to simulate the probability for each activity being involved in changes, now we describe what kind of change operations we consider on generating each variants. Since the original reference model $S$ only contain activities $a_j, j = 1 \ldots m$, if an activity $a_j, j = m+1 \ldots m+x$ are involved in process configurations, we can only *insert* such activity in $S$ in order to configure a particular variant $S_i$. For the activities $a_j, j = m-x+1 \ldots m$ which already exist in the reference model $S$, we can *move* such activity to another position when configuring a particular variant $S_i$. We ignored delete operation when generating the dataset since it can be easily handled. [9]

While how often an activity $a_j$ is inserted when generating variants can be easily obtained by activity frequency $g(a_j)$, it is difficult to know how often activities are involved in move operations because move operation does not lead to the change of the activity set and structure changes are difficult to identify. We design our simulation by considering both insert and move operation also with the purpose of checking whether these two types of change operations are considered equally important by our heuristic mining algorithm. Since the move and insert operations have the same probability to be employed when generating dataset, we would also expect the heuristic mining algorithm can also apply relatively same amount of change or insert operations to discover a reference model. If it is not the case, it is an indication that the fitness function is not properly designed.

**Parameter 4 (Activity Consistence)** While parameter 3 describe how often a particular activity is changed, parameter 4 controls where these activities are changed (moved or inserted) to.

As we discussed in subsection 5.2, there are numerous options to insert an activity $a_j$ into a particular process model $S_c$. $a_j$ can be clustered with any block

[9] Delete operation is only not considered in generating dataset but is still considered in performing mining. Deleting activity $a_j$ leads to the change of activity frequency $g(a_j)$, which can also be realized by insert operation.

in $S_c$ by one of the order relation $\tau = \{0, 1, *, -\}$. Since the number of blocks contained in a process model is often large, the number of possible resulting models by inserting $a_j$ in $S_c$ is also large.

Therefore, for a particular group, we can define several change operations as *consistent change operations*, i.e., when an activity $a_j$ is inserted into $S_c$, it is always clustered with a particular block by a particular order relation $\tau'$. Therefore, we can control how frequent we perform the corresponding consistent change operations to configure a particular variant $S_i$.

The first step is then to determine these consistent change operations for a particular group. For example, consider Fig. 15. For a given reference model $S$, we only allow performing $x$ changes from activity set $a_j, j = (m-x+1) \dots (m+x)$ to configure a particular process variants $S_i$. The remaining activities, i.e., $a_j, j \in [1, (m-x)]$, are stable, i.e., will not be changed. These activities are the suitable candidate blocks to be cluster with. For each activity $a_j, j \in [(m-x+1), (m+x)]$, we can find a corresponding activity $a_{j'}, j' \in [1, (m-x)]$, so that if we need to perform a consistent change on $a_j$, it is always clustered with $a_{j'}$ by a particular order relation $\tau'$. For example, consider the case we described in Table 3. Activities `A, B, C, D, E, F` and `G` will not be involved in change since their probability for change all equal to 0. Therefore, if we need to move `H, I` or `J` or to insert `K, L` or `M`, we always cluster them with one of the stable activities, e.g., if we need to move `J` to configure a particular variant by a consistent change operation, it will always be clustered with `B` by order relation saying $\tau' =$ "0".

We therefore can control *activity consistence* of activity $a_j$ by setting how often a consistent change operation is performed. If $a_j$ is to be change, we set the probability to perform a consistent change operation to $p$, and the probability to perform a random change operation (i.e., randomly select a block and cluster it with a random order relation) is consequently $1 - p$. In order to analyze the relation between the activity consistency and activity occurrence, we have designed four different scenarios to cover the space as illustrated in Fig. 14. The scenarios are depicted in Fig. 16.

These four scenarios are constructed by keeping either occurrence or consistency stable while changing the others:

1. **LowOcc** scenario keeps occurrence of activities all at 30%, while makes the consistency of these activities varying from 0 to 80%.
2. **HighOcc** scenario keeps the occurrence of activities all at 70% while making the consistency of them also varying from 0 to 80%.
3. **LowCon** scenario keeps the consistency of activities all at 30% while making the occurrence change from 0 to 80%.
4. **HighCon** scenario keeps the consistency of activities all at 70% while making the occurrence changes from 0 to 80%.

Note that the scenario is not describing a particular activities but a set of activities. For the group with parameter 1 being "large-sized" and parameter 2 being "large-change", 30 activities (15 for move and 15 for insert) will be involved in configuring the 100 process variants with different occurrence or consistency values. Therefore, one particular scenario can cover a large space in the value

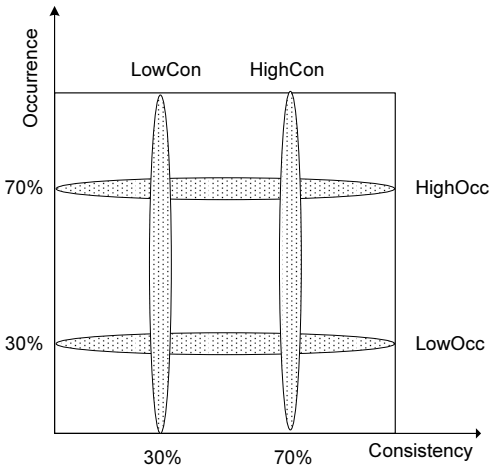



**Fig. 16.** Space coverage using simple scenarios

space (cf. Fig. 16). This can significantly enhance our research since for each scenarios, we have a collection of activities for analysis.

In order to better cover the value space, we have designed four additional scenarios. While the occurrence always follows the curve as depicted in Fig. 15, the consistency of the corresponding activities are depicted in Fig. 17.

1. **Positively correlated** scenario makes the consistency of a particular activity positively correlated to its occurrence, i.e., when activity $a_j$ has high occurrence, it also has high consistency. This applies to both moved and inserted activities.
2. **Negatively correlated** scenario, on the contrary, makes the consistency negatively correlated to the occurrence. When $a_j$ has high occurrence value, it should have a low consistency. This applies also to both moved and inserted activities.
3. **Focus on "move"** scenario assigns high consistency to the moved activities while set low consistency to the inserted activities.
4. **Focus on "insert"** scenario on the contrary assigns high consistency to the inserted activities and low consistency to the moved activities.

Fig. 17 also depicts the value space as covered by each scenario. We additional design these scenario not only to better cover the value spaces, but also to analyze whether the insert and move operations are considered equally important by the algorithm, i.e., since insert and move operations are equally used when generating the dataset, our heuristic mining algorithm also should not show significant difference on move and insertions when discovering the reference model.

To sum up, this subsection describes how each group of dataset are generated by adjusting the values of different parameters. Since we have 3 options for parameter 1 (small-size, medium-size and large-sized), 3 options for parameter

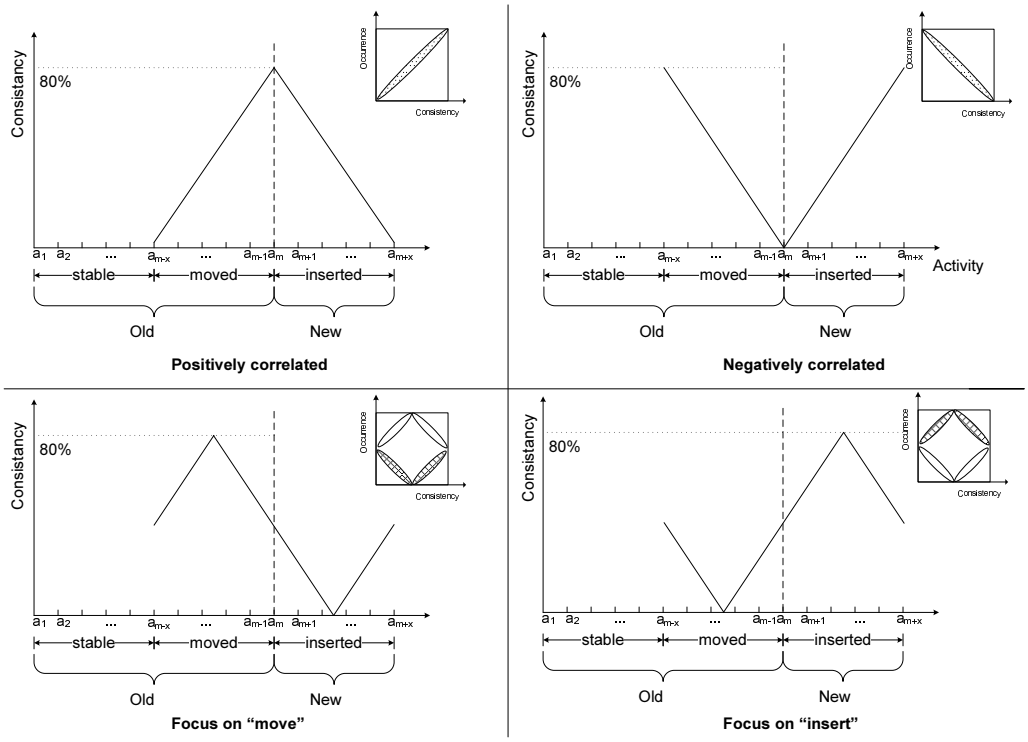



**Fig. 17.** Space cover using complex scenarios

2 (small-change, medium-change and large-change) and 8 scenarios to cover the value space of occurrence and consistency, we have in turn generated $3\times3\times8 = 72$ groups of dataset containing 1 reference model and 100 variants configured from it. In the next subsection, we will describe what information we can obtain from each group of dataset.

### 6.4 Simulation Setup

For each one of the 72 groups of dataset constructed based on different scenarios, we perform our heuristic mining to discover a new reference model by mining the collection of variants. We do not set any constraints on search steps, i.e., the algorithm will only terminate when no better model can be discovered. We used a Dell Latitude Laptop (2.4 GHZ CPU and 3.5 GB RAM) to run our simulation under Windows environment. The following information are documented in each group:

1. **Original reference model**, i.e., the model based on which we perform the changes (cf. subsection 6.1).

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 subsection 6.3. For each group, we generate 100 process variants. Note that although the 100 variants are generated by following a same scenario, these models are *NOT* same. The reason is that the scenario only depicted the feature of the collection of variants but not a particular variant.
3. **Search results**. We document all the intermediate process models as well as the end search result as obtained from the heuristic search. The corresponding change operations are also documented. As example consider Fig. 11 as the heuristic search result we can obtain by mining the reference model $S$ and variants $S_i$ from Fig. 3.
4. **Fitness and average weighted distance**. Similar to the evaluation results we presented in Table 2 of search results (cf Fig. 11), we also compute the fitness and average weighted distance value of every intermediate process models as obtained from our heuristic mining. We additional documents delta-fitness and delta-distance in order to examine the influence of every change operation.
5. **Execution time**. At last, we also document The running time to perform our heuristic search algorithm.

## 7 Simulation Result

In Section 6, we have described how the simulation is setup and what research questions we want o answer through the simulation, in this section we give the answer to these research questions. In particular:

1. Subsection 7.1 will provide the basic analysis of the heuristic algorithm, e.g., average distance reduction, running time etc.
2. Subsection 7.2 will show the correlation analysis of delta-fitness and delta-distance.
3. Subsection 7.3 will compare the correlation values of process models with different size to answer whether the performance of our algorithm can scale up.
4. Subsection 7.4 will examine whether nor not the important changes are performed at the beginning.
5. Subsection 7.5 will provide a method to train a threshold value to improve the performance of our heuristic mining algorithm.

### 7.1 Basic Performance Analysis

**Improvement on average weighted distances** In 60 (our of 72) groups, we are able to discover a new reference model different than the original one. The average weighted distance of the discovered model is *0.765* shorter than that of the original reference model, i.e., setting the discovered model as the

new reference model can reduce on average 0.765 change operations to configure the variants. When compared to the average weighted distance of the original reference process model, it is on average *17.92%* shorter.

**Number of Change and Move operations** For the 60 (out of 72) groups which we are able to discover a different reference model, we perform in total *284* change operations, i.e., on average *4.73* change operations per group. Within the 284 change operations, there are *132* insert operations and *152* move operations. Clearly, we see *no significant difference* between the number of insert or move operations. which indicates that the two operations are well-balanced by our algorithm. Reason is that we performed relatively same amount of insert or move operations when generating the data set (cf. subsection 6.3) and such trend is also shown during the discovery of the reference model.

**Running Time** Clearly, the number of activities contained in the variants can significantly influence the execution time of our algorithm. The search space is simply larger for large models since the number of candidate activities for change is higher and the number of blocks contained in a large model is also higher. We therefore analyze the execution time of our algorithm by considering the size of process models. The average running time is summarized in Table 4.

| | small-sized | medium-sized | large-sized |
|---|---|---|---|
| Average search time (s) | 0.184 | 4.568 | 805.539 |
| Average # of changes performed | 1.83 | 3.52 | 8.43 |
| Model sizes | $10 \sim 13$ | $20 \sim 26$ | $50 \sim 65$ |

**Table 4.** Average search time for process models of different sizes

While it takes only little time to discover the result model for the small-sized and medium-sized models, it takes considerable longer time (on average 805.539 seconds) to find the result for the large-sized model. It takes long not only because the process models are larger, but also because the search steps are longer as well. We need to perform on average 8.43 change operations on the original reference model to discover the end result. Reason is that the variants in these groups are more different from each other than those in the small-sized and medium-sized model (the number of change operations is determined by the model size, i.e., we change respectively 10%, 20% and 30% of activities to configure a variant (cf. subsection 6.3)). The consequence is that the discovered model could also be more different from the original model. However, we believe the running time is acceptable considering the complexity of the problem, especially when compared with other data mining problems which might take hours or even days to compute [36].

### 7.2 Correlation of Delta Fitness and Delta Distance

One important issue we want to investigate is how fitness value is correlated with average weighted distance. As we discussed in Section 6, fitness value is only a "quick guess" of how close a candidate model $S_c$ is to the collection of variants, it is not as precise as the average weighted distance and can also not perfectly correlated with the average weighted distance since computing it is an $\mathcal{NP}$ problem. In this subsection, we will analyze how much they are correlated with each other.

Our heuristic search algorithm is a best-first approach, i.e., we search whether we can find a process model with a higher fitness value, therefore, it is more useful to measure how much the delta-fitness (the difference between the fitness values before and after change) is correlated with the delta-distance (the difference between the average weighted distances before and after change), because it is improvement of the fitness value (delta-fitness) that guide the search steps (cf. Section 5). Another reason not to directly analyze the correlation of the fitness value and the distance value is their value ranges. While, the fitness value of a model has value range $[0, 1]$, the average weighted distance has value range $[0, +\infty]$. On the contrary, the delta-fitness and the delta-distance both have value range $[-1, 1]$ since we only change one activity at a time. Therefore, the correlation between the delta-fitness and delta-distance is more reasonable to consider. Similar techniques for evaluating fitness function is also widely used in evaluating other algorithms [14].

Since every change operation will lead to particular change on the process model and consequently creates a delta-fitness value and a delta-distance value, we obtain 284 data samples since we performed in total 284 change operations. We plot these data sample in Fig. 18 as (delta-fitness, delta-distance). For example, consider the search result in Table 2. We can obtain three data samples of delta-fitness and delta-distance from it, i.e., (0.171,0.8), (0.04,0.6) and (0.017,0.25).

In Fig. 18, the x-axis is the delta-fitness value and the y-axis is the delta-distance value. All delta-fitness are larger than 0. It is easy to understand since the algorithm would only perform a change if and only if it can find a model with higher fitness value. However, the delta-distance is not always larger than 0, which indicates that sometimes the a change operation can make the result even worse. It is not surprising to see this since the fitness value is only a quick guessing of the distance. We additional plot a line with delta-distance being 0 to separate "good" samples (positive delta-distance) and "bad" samples (negative delta-distance).

In Fig. 18, we also marked the data sample from groups of different model sizes separately. It is clear that these three groups form three different clusters, i.e., they are not overlapping too much with each other and the larger the model size is, the more its cloud positions towards the y-axis. This indicates that the size of process model can influence the delta-fitness value. Reason is that the fitness value is measured by the average of how much each pair of order relations matches each other in the candidate model and in the variants (cf. Formula 5),

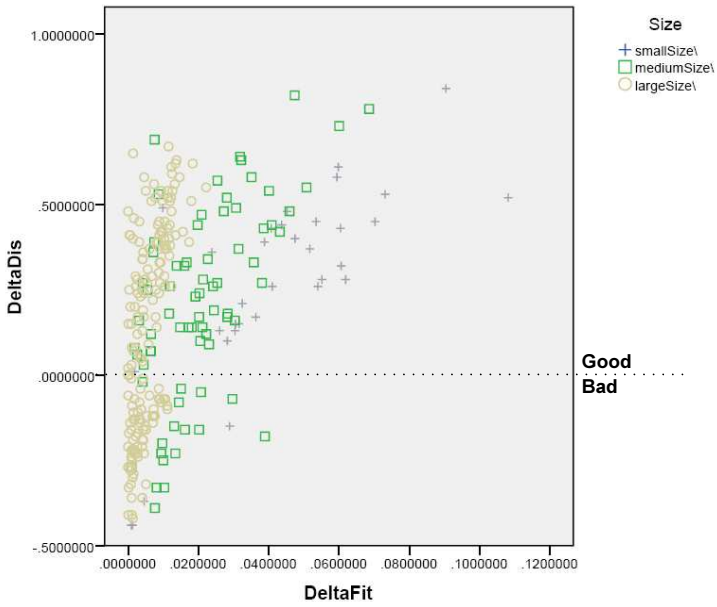



**Fig. 18.** Construct the search tree

therefore since one change operation only change one activity, it would have higher influence on the small model compared to on the big ones. Therefore, it is more reasonable to analyze the correlation of the groups of different sizes separately.

We use Pearson correlation to measure the correlation between the delta-fitness and delta-distance [35]. Let $X$ be the delta-fitness and $Y$ be the delta-distance. We obtain $n$ data samples $(x_i, y_i)$ where $i = 1, \ldots, n$. Let $\bar{x}$ and $\bar{y}$ be the mean of $X$ and $Y$, let $s_x$ and $s_y$ be the standard deviation of $X$ and $Y$. The Pearson correlation can be computed using Formula 6.

$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y} \tag{6}$$

We additional tested whether the correlations are significant, i.e., whether the correlation coefficients are significantly different from 0 [35]. The results are summarized in Table 5.

| | Number of data | Correlation | Probability of $r = 0$ | Significantly Correlated |
|---|---|---|---|---|
| Small-sized | 33 | 0.762 | <1.0E-8 | Yes |
| Medium-sized | 74 | 0.589 | <1.0E-8 | Yes |
| Large-sized | 177 | 0.623 | <1.0E-8 | Yes |

**Table 5.** Delta fitness and delta distance correlations of groups with different sizes

It is clear that the correlation coefficients obtained from all three groups are *significant and high*. The high positive correlation between delta-fitness and delta-distance indicates that when we could find a model with higher fitness value, we would have very high chance also to reduce the average weighted distance and therefore should change the old model to the newly discovered one. A correlation is normally considered high if it is larger than 0.5 [35]. In our case, all three groups show high correlation coefficients, especially when compared with most other heuristic or genetic algorithms in computer science, where the coefficients between the fitness value and the local optimum are mostly low or even negative [14].

### 7.3 Correlation Comparison

In the former subsection we have discussed the correlation between the delta-fitness and delta-distance. Since it is an important value to evaluate the performance of our algorithm, we would also like to know whether the coefficient will change when dealing with process models of different sizes. It is important to know since it directly reflects the scalability of the algorithm. If the correlation coefficient does not change when dealing with small process models or large ones, the scalability of our algorithm is consequently good since the performance is stable when dealing with process models of different sizes.

When purely looking at the correlation coefficients from Table 5, it is difficult to derive any trend since the lowest correlation value is obtained from the medium-sized models. More importantly, since we have different number of data samples from the three groups, these three correlation should have different "credibility", the correlation derived from 177 data points should be more reliable than the one we obtain from 33 samples. To compare correlation values while considering the size of the data sample, we need to employ statistical approach again.

Since the sampling distribution of Pearson correlation analysis is not normally distributed, we first need to perform a *Fisher's Z transformation* to covert the Pearson correlation to a normally distributed variable [35]. Let $r$ be a correlation coefficient, we can perform Fisher's Z transformation using Formula 7.

$$Z(r) = 0.5 \times (\ln(1 + r) - \ln(1 - r)) \tag{7}$$

The distribution of $Z(r)$ has two important attributes: first it is normally distributed and second it has a known standard error of $\frac{1}{\sqrt{n-3}}$ where $n$ equals the number of data samples for computing the Pearson correlation $r$. We therefore can compare the difference between two correlations $r_1$ (obtained from $n_1$ data samples) and $r_2$ (obtained from $n_2$ data samples) using Formula 8.

$$\rho(r_1, r_2) = \frac{Z(r_1) - Z(r_2)}{\sqrt{\frac{1}{n_1-3} + \frac{1}{n_2-3}}} \tag{8}$$

The difference $\rho(r_1, r_2)$ follows approximately Standard Normal Distribution [35]. Table 6 shows the pairwise comparison results of the correlations from the

three groups of different sizes. In all three comparisons, the correlation coefficients are not significantly different from each other, i.e., they are all statistically same. This indicates that the performance of our heuristic algorithm is *NOT dependent on the size of the models*, i.e., the algorithm can scale up on dealing with large size process models without sacrificing its performance.

| | $\rho$ value | Probability of being same | Significant? |
|---|---|---|---|
| Small-sized V.S. Medium-sized | 1.51 | 0.130 | Yes |
| Medium-sized V.S. Large-sized | -0.4 | 0.689 | Yes |
| Small-sized V.S. Large-sized | 1.37 | 0.170 | Yes |

**Table 6.** Paired correlation coefficients

### 7.4 Monotonicity Test

In this subsection, we will test whether our heuristic search algorithm always perform important change operations (changes that have higher delta-distance value) at the beginning. We therefore perform two tests. The first test is to see how much we are able to reduce the distance by only performing a limited number of change operations. Another test is to see whether the delta-distance is monotonically decreasing.

**How Much the Top n% Change Operations Can Do** In our simulation, we do not control the search depth, i.e., we allow the algorithm continue as far as better models can still be discovered. As mentioned before, one important feature of our algorithm is its ability to control how many change operations we want to perform. This implies that the important ones should be put at beginning. One approach is to compute how much the top $n\%$ changes have achieved on reducing the average weighted distance.

For example, consider the search results in Table 2. We have perform in total 3 change operations to discover the best model. In total, the average weighted distance reduced 1.75 from 4 as based on the original reference model $S$ to 2.25 as based on $R_3$ (cf. Table 2). We therefore can analyze how important the changes at beginning is by compute how much it has reduced the average weighted distance. For example, the first change operation reduces the average weighted distance by 0.8, compared to the overall distance reduction by 1.75, we have accomplished $0.8/1.75 = 45.71\%$ distance reduction by performing only the top one change operation. Since we only have 3 change operations, we can claim that by performing the top 33.33% of the change operations, we can already accomplish 45.71% distance reduction. This therefore can indicate how important the changes at beginning are, the higher the distance reduction has accomplished, the more important these operations are. Similarly, we can also compute how much fitness improvement can be accomplished by performing only the top changes. The results are summarized in Table 7.

| | top 33.33% changes | top 50% changes |
|---|---|---|
| Fitness gain | 57.35% | 74.60% |
| Distance gain | 63.80% | 78.93% |

**Table 7.** Fitness and distance gains if only apply the top changes

Table 7 summarize the average distance and fitness gains by the top 1/3 and top 1/2 change operations. It becomes clear that the changes at beginning are *a lot more important* than the changes performed at end. For example, the top 1/3 change operations has achieved 63.80% distance reduction while the remaining 2/3 change operations only achieved the remaining 36.20% distance reduction. If we only perform the first half of the change operation, we would already obtain around 80% of the distance reduction. This simple analysis has already implied that the changes performed at the beginning are a lot more important than the changes performed at end.

**The Monotonically Decreasing Score** The approach described in former subsection is an abstract approach, i.e., we analyze the effect of a collection of change operations rather than focus on comparing each individual ones. In this subsection, we will focus on each individual change operations, i.e., whether it is really the case that the one performed before is better than the one performed next to it, i.e., whether the delta-distance is monotonically decreasing.

Most of the monotonicity test in data mining or artificial intelligence provide binary answers, i.e., the data sample is either monotonic or non-monotonic [20, 36]. These monotonicity test are too restrict here since one problematic number can kill the whole monotonicity test, especially considering the fact that a heuristic algorithm is only a reasonable "guessing". Statistical monotonicity test (e.g., [4]) can not be applied either, since there are only on average 4.73 change operations performed in each group. This number is too low to conduct any creditable statistical analysis. Therefore, we use the following method to test the monotonicity.

For one group of dataset, let $n$ be the number of change operations performed on the original reference model to discover the end result. Let $x_i, i = 1 \dots n$ and $x_j, j = 1 \dots n$ be the delta-distance of number $i_{th}$ and number $j_{th}$ change operations. The monotonically decreasing score $\mu$ can be computed using Formula 9.

$$\mu = \frac{\left|\left\{(x_i, x_j) | (i < j) \wedge (x_i \geq x_j)\right\}\right|}{n \times (n-1)/2} \tag{9}$$

$\mu$ measure the monotonicity by comparing every pair of change operations. If for two change operations $i, j$ with $i < j$, i.e., $i$ performed before $j$, we obtain delta-distance $x_i$ is larger or equal to $x_j$, it means change $i$ and $j$ follow monotonically decreasing. Otherwise, not. Clearly we obtain $\mu \in [0, 1]$ and if $\mu = 1$ holds, delta-distance keeps perfect monotonically decreasing, or if $\mu = 0$ holds, delta-distance would keep monotonically increasing. The higher $\mu$ is the higher

delta-distance follows the trend of monotonically decreasing. Similarly, we can test the monotonicity of delta-fitness. The results are summarized in Table 8.

| | average $\mu$ | average $\mu$ with 5% error rate |
|---|---|---|
| Fitness | 0.9942 | 0.9987 |
| Average weighted distance | 0.6682 | 0.6858 |

**Table 8.** The average monotonic decreasing score $\mu$ as obtain from the simulation

Besides the average monotonic decreasing score $\mu$, Table 8 also shows the average monotonic decreasing score $\mu$ by allowing 5% error rate, i.e., if $i < j$ and $x_i \geq x_j \times (1 - 0.05)$, we still consider it monotonic decreasing just to avoid rounding errors. It becomes clear that the delta-fitness is almost perfectly monotonically decreasing while such trend on delta-distance is not very strong. The difference is due to the fact that the correlation between these two values are not perfect. Therefore, we can not claim that the delta-distance keeps decreasing as the search continues. The monotonic decreasing on the delta-distance is only *strong at high abstraction level*, e.g., the top 1/3 change operations would accomplish around 2/3 of the distance reduction. It is *not very strong* when comparing each individual change operation.

### 7.5 Pruning threshold training

If we revisit our delta-fitness and delta-distance graph as plotted in Fig. 18, it is clear that there are still quite some "bad" data points. Those are the points with positive delta-fitness but negative delta-distance. Though these bad data points can never be prevented due to the feature of heuristic algorithm, we can at least improve it, i.e., reduce the chance such bad point from appearing.

When looking at these "bad" points, we found them most time at the down-left corner of the chart. It indicates that when the delta-fitness is low, the chance of getting a negative delta-distance will get bigger. In order the quantitatively evaluate it, we introduce the concept *precision* here.

Let $X$ be the delta-fitness and $Y$ be the delta-distance. We obtain $n$ data samples $(x_i, y_i)$ where $i = 1, \ldots, n$. Given a delta-fitness value $x$, we can compute *precision* $p(x)$ using Formula 10:

$$p(x) = \frac{\left|\left\{(x_i, y_i) | (x_i \geq x) \wedge (y_i > 0)\right\}\right|}{\left|\left\{(x_i, y_i) | x_i \geq x\right\}\right|} \tag{10}$$

Given a delta-fitness value $x$, precision $p(x)$ measures the ratio of "good" data samples (with delta-distance larger than 0) among data samples with delta-fitness value larger or equal to $x$. The higher $p(x)$ is, the more "good" sample we have in the data sample. Such measurement is widely used in the fields like information retrieval [3] or data mining [36]. Fig. 19 depicts the precision values $p(x)$ for different delta-fitness value $x$.

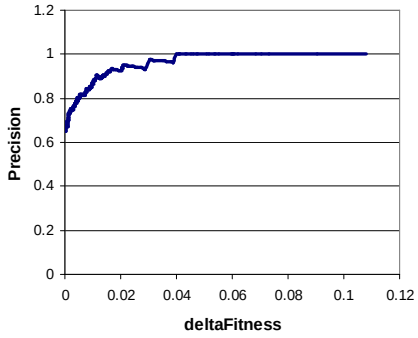



**Fig. 19.** Delta-fitness and precision chart

It becomes clear from Fig. 19 that, the lower the delta-fitness is, the lower the precision is. When consider only the data samples with delta-fitness larger than 0.0401, all the corresponding delta-distance are positive, i.e., they are all "good" samples. The precision keeps reducing until it reaches 65.14% when considering all the data points. This has indicated that a lot of "bad" data samples are with low delta-fitness values.

The precision analysis indicates that we can probably improve our heuristic mining algorithm by determine a threshold value of delta-fitness. Since most of the "bad" data samples are obtained with low delta-fitness values, we will only allow to perform a change if the delta-distance value is larger than this threshold value. In the following we introduce two approaches to discover such threshold based on our simulation data.

**Classification Tree** We first introduce an approach using classification tree [24]. By learning from the "good" and "bad" data samples, we should be able to classify them by a threshold delta-fitness value. Let $X$ be the delta-fitness and $Y$ be the delta-distance. We obtain $n$ data samples $(x_i, y_i)$ where $i = 1, \ldots, n$. We can assign each data sample a binary value $z_i$ being "$TRUE$" or "$FALSE$" depending on the value of $y_i$. If $y_i > 0$ holds, $z_i$ is "$TRUE$", otherwise, $z_i$ is "$FALSE$". We therefore can build a classification tree using the delta-fitness $x_i$ and the binary variable $z_i$. We choose the algorithm *C4.5* to build the classification tree [24], and use *Weka*, which is one of the most popular open-source data mining tools, to compute the result [45]. The details about the classification algorithm or the data mining tool are out of the scope of this paper; the important issue is that they are standard method to perform such classification. The resulting classification tree is shown in Fig. 20.

The simple classification tree has divided the data samples into two groups based on the values of delta-fitness. When delta-fitness $x_i$ of a certain data sample $(x_i, y_i)$ is less or equal to 0.001516, the classification tree will classify it as "$FALSE$", i.e., it is more often the case that $y_i$ is lower than 0. On the contrary, if $x_i$ is larger than 0.001516, it is more likely to obtain a positive delta-distance

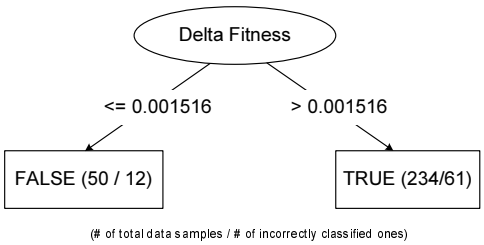



**Fig. 20.** The classification tree build based on delta fitness values

value. The classification tree is also not 100% precise: 12 out of 50 data samples with delta-fitness lower than 0.001516 actually provide positive delta-distance while 61 out 234 data samples with delta-fitness larger than 0.001516 provide negative delta-distance values. Though the classification tree is not perfectly precise, it is already the best we can build purely based on the delta-fitness value.

The discover threshold 0.001516, therefore can help us improve the performance of our heuristic algorithm. We shall only perform a change if the improvement of the fitness value is larger than such threshold. In this case, we are expected to reduce the chance of performing a wrong change, i.e., a change which makes the distance between the discovered model and the variants even large (negative delta-distance). If we filter the data sample by this threshold value of delta-fitness, we will increase the precision of the whole data sample from 65.14% to *73.93%*. Correspondingly, the average reduction on average weighted distance per group will also improve from 0.765 to *0.879*, i.e., we can discover better models by setting a threshold value to guide our heuristic search.

**Threshold by Overall Distance Gain** In the former subsection, we present a standard data mining approach to improve our algorithm, while in this subsection, we introduce a more initiative and straight forward approach to discover a threshold value. One disadvantage of the above mentioned classification tree is that it can not consider the importance of a "good" or a "bad" data sample. The threshold is trained by a binary decision variable $z_i$ which is either "$TRUE$" or "$FALSE$". It does not consider how much "$TRUE$" or how much "$FALSE$" by directly measuring the average weighted distance value. Here, we introduce a method to discover a threshold by considering the delta-distance value.

Let $X$ be the delta-fitness and $Y$ be the delta-distance. We obtain $n$ data samples $(x_i, y_i)$ where $i = 1, \ldots, n$. We can compute the sum of delta-distance $\eta x$ by a certain threshold delta-fitness $x$ using Formula 11:

$$\eta(x) = \sum_{i=1, x_i \geq x}^{n} y_i \tag{11}$$

$\eta(x)$ measures the sum of the delta-distance after filtering out the data samples with delta-fitness lower than a given $x$. Fig. 21 depicts the curve of $\eta(x)$ according to the value of $x$. We specially zoom in the part with $x$ in the interval [0, 0.006].

$\eta(x)$ keeps increasing as $x$ decreases. It is easy to understand since the lower $x$ is, the less data samples will be filtered out. However, $\eta(x)$ reaches its maxim of 51.72 when $x$ equals 0.0014, and starts to decrease as $x$ decreases. This indicates that in the interval [0, 0.0014) of delta-fitness, there are more data samples with negative delta-distance. If use 0.0014 as the threshold value to guide our heuristic search, i.e., we only perform a change if the improvement of fitness value is larger than 0.0014, we can additional reduce the average weighted distance per group from 0.765 to *0.892*. The corresponding precision for all data samples will also increase from 65.14% to 73.22%.

Please note that the threshold value trained using our simulation data would be a case specific value. It can not be generalized that using these threshold values can always improve the performance of our algorithm. However, it is useful to perform such analysis in this paper in two aspects. First, we have indicated that when the delta-fitness value is low, the algorithm bear a higher chance of performing a wrong move, i.e., the user should be careful when discovering a model with only a little better fitness value. Secondly, using the suggested approaches, user can train their own threshold value based their own data set. We present the corresponding training method in this paper just to provide a guideline about how to apply and/or improve our heuristic mining algorithm to a domain-specific field, i.e., user can train a threshold value to make our heuristic mining algorithm works better for their own problem.

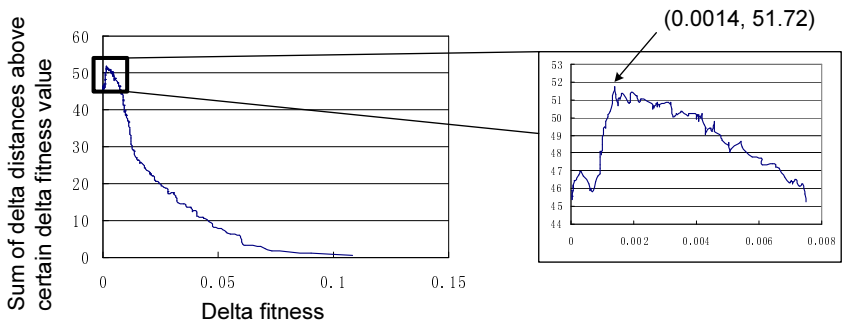



**Fig. 21.** The sum of the delta distances with delta fitness values above certain threshold

## 8 Related Work

Though heuristic search algorithm is widely used in various fields like data mining [36], artificial intelligence [20], machine learning [24], only few approaches

using heuristic for process variant management exist. In particular, there is few adequate solution for learning from the adaptations applied to configure a collection of process variants out of a given process model.

Structural process changes during runtime and approaches for flexible process configuration have been intensively discussed in literature for several years [28, 29, 41]. A comprehensive analysis of theoretical and practical issues related to (dynamic) process changes, for example, has been provided in the context of the ADEPT2 change framework [25]. Furthermore, there exist theoretical frameworks for dynamic structural changes of Petri nets [37]. Based on these theoretical considerations, the BPM suite ADEPT2 [27] and tools for configurable workflow models [7] have emerged.

There exist approaches which provide support for the management and retrieval of separately modeled process variants. As an example, [19, 18] allows storing, managing, and querying large collections of process variants within a process repository. Graph-based search techniques are used in order to retrieve variants that are similar to a user-defined process fragment. Obviously, this approach requires profound knowledge about the structure of stored processes, an assumption which does not always hold in practice. Apart from this, no techniques for analyzing the different process variants and for learning from their specific customizations are provided.

In process mining, a variety of techniques has been suggested [38, 44, 6, 39]. As illustrated in [16], traditional process mining is different from process variant mining due to its different goals and inputs. [13] presents a method to mine configurable process models based on event logs, but is still focusing on discovering process models from event logs rather than reducing efforts for process configuration.

There are few techniques which allow to learn from process variants by mining recorded change primitives (e.g., to add or delete control edges). For example, [2] measures process model similarity based on change primitives and suggests mining techniques using this measure. However, this approach does not consider important features of our process meta model; e.g., it is unable to deal with silent activities and it does also not differentiate between AND- and XOR- branchings. Similar techniques for mining change primitives exist in the field of association rule mining [36] and maximal sub-graph mining [12] as known from graph theory [33]; here common edges between different nodes are discovered to construct a common sub-graph from a set of graphs. Similar constraints hold for "subdue" discover as commonly applied in the field of bioinformatics, where "subdue" represent a certain sub-structure of genes or proteins, which has a certain chemical or biological behavior [11].

The ProCycle system enables change reuse at the process instance level to effectively deal with recurrent problem situations [30, 42]. ProCycle applies case-based reasoning techniques to allow for the semantic annotation as well as the retrieval of process changes. Based on this the respective process adaptations can be re-applied in similar problem context to configure other process instances later on. If the reuse of a particular change exceeds a certain threshold, it will

become a candidate for adapting the process schema at the type level; i.e., for evolving this schema accordingly and thus for considering the change for future process instances as well. Though the basic goal of ProCycle is similar to our approach, the techniques applied are much more simpler and do not consider variation in changes.

To mine high level change operations, [8] present an approach based on process mining techniques, i.e., the input consists of a change log, and process mining algorithms are applied to discover the execution sequences of the changes (i.e., the change meta process). However, this approach simply considers each change as individual operation so that the result is more like a visualization of changes rather than mining them. [15] has provided an approach to discover a reference model by learning from a collection of variants. However, it is not able to take the original reference model into consideration can consequently can not control the updating procedure of the reference model. None of the discussed approaches are sufficient in supporting the evolution of reference process model towards an easy and cost-effective model by learning from process variants in a controlled way.

## 9 Summary and Outlook

The main contribution of this paper is to provide an heuristic search algorithm supporting the discovery of a reference process model by learning from a collection of block-structured process variants. Adopting the discovered model will make the process configuration easier since it would require less effort (measured by the number of change operations) to configure these variants. Our heuristic algorithm can also take the original reference model into consideration so that the user can control how much the discover model is different from the original one. In this way, we can not only avoid spaghetti-like process model but also control how much changes we want to perform, i.e., we can choose only perform the important changes at beginning while ignore the less important change at the end. The algorithm can also automatically determine which activities should or should not be considered in the reference model, filtering or pre-analysis of the activity sets are not needed using our algorithm.

We have evaluated our algorithm by performing a simulation. Based on the simulation results, we can conclude that:

1. The fitness function of our heuristic search algorithm is correlated with the average weighted distance with high correlation coefficient. This indicates good performance of our algorithm since the approximation value we use to guide our algorithm is nicely correlated to the real one.
2. Our algorithm can also scale up since its performance measured by the correlation between the fitness and distance is not dependent on the size of models, i.e., it keeps good perform when dealing with even large size process models.
3. When discovering the reference model by change the original one, the important changes, which largely reduce the average weighted distance to the

variants, are performed at the beginning. The simulation results indicate that the first 1/3 change operations will result in about 2/3 of the overall distance reduction.

At end, we have also suggested two approaches to improve our heuristic search algorithm by learning a threshold value. Though the results may not be generalized to all cases, the suggest approach can also support users to adapt our algorithm with their own domain-specific knowledge.

The results look promising while there are still more work needed to be done. As the algorithm will take relatively long time when encountering large process models, it would be useful to improve the search algorithm to make it faster. It would also be useful if we can integrate out algorithm with other process mining algorithms so that it does not purely focus on reducing the average weighted distance of the reference model but also take the behavior perspectives into consideration [38].

## 10 Acknowledgment

Here we thank Matthias Wettstein for his contribution in the statistic correlation analysis.

## References

1. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.
2. J. Bae, L. Liu, J. Caverlee, and W.B. Rouse. Process mining, discovery, and integration using distance measures. In *ICWS '06*, pages 479–488, Washington, DC, USA, 2006.
3. H.M. Blanken, A.P. de Vries, H.E. Blok, and L. Feng. *Multimedia Retrieval*. Springer, 2007.
4. A.W. Bowman, M.C. Jones, and I. Gijbels. Test monotonicity of regression. *Journal of Computational and Graphical Statistics*, 7(4):489–500, 1998.
5. T.H. Davenport. *Mission Critical - Realizing the Promise of Enterprise Systems*. Harvard Business School, 2000.
6. A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, NL, 2006.
7. F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable workflow models. *Int. J. Cooperative Inf. Syst.*, 17(2):177–221, 2008.
8. 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.
9. A. Hallerbach, T. Bauer, and M. Reichert. Managing process variants in the process lifecycle. In *Proc. 10th Int'l Conf. on Enterprise Information Systems (ICEIS'08)*, pages 154–161, 2008.
10. 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.

11. L. B. Holder, D. J. Cook, and S. Djoko. Substructure discovery in the subdue system. In *In Proc. of the AAAI Workshop on Knowledge Discovery in Databases*, pages 169–180, 1994.
12. J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. In *KDD '04*, pages 581–586, New York, NY, USA, 2004. ACM.
13. M.H. Jansen-Vullers, W.M.P. van der Aalst, and M. Rosemann. Mining configurable enterprise information systems. *Data Knowl. Eng.*, 56(3):195–244, 2006.
14. T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, USA, 1995. Morgan Kaufmann Publishers Inc.
15. 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.
16. C. Li, M. Reichert, and A. Wombacher. Mining process variants: Goals and issues. In *IEEE SCC (2)*, pages 573–576. IEEE Computer Society, 2008.
17. 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.
18. R. Lu and S. W. Sadiq. On the discovery of preferred work practice through business process variants. In *ER*, pages 165–180. Springer, 2007.
19. R. Lu and S.W. Sadiq. Managing process variants as an information resource. In *BPM'06*, pages 426–431, 2006.
20. G. F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Pearson Education, 2005.
21. 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.
22. 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.
23. D. L. Parnas. Software aging. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
24. J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., USA, 1993.
25. M. Reichert and P. Dadam. ADEPT flex -supporting dynamic changes of workflows without losing control. *Journal of Intelligent Info. Sys.*, 10(2):93–129, 1998.
26. M. Reichert, S. Rinderle, and P. Dadam. On the common support of workflow type and instance changes under correctness constraints. In *CoopIS'03*, pages 407–425, 2003.
27. M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with adept2. In *ICDE '05*, pages 1113–1114, Washington, DC, USA, 2005. IEEE Computer Society.
28. S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems – a survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.
29. S. Rinderle, M. Reichert, and P. Dadam. Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases*, 16(1):91–116, 2004.

30. S. Rinderle, B. Weber, M. Reichert, and W. Wild. Integrating process learning and process evolution a semantics based approach. In *Proc. 3rd Int'l Conf. on Business Process Management (BPM'05)*, LNCS 3649, pages 252–267, 2006.
31. S. Rinderle-Ma, M. Reichert, and B. Weber. On the formal semantics of change patterns in process-aware information systems. In *ER'08*, LNCS 5231, pages 279–293, 2008.
32. M. Rosemann and W.M.P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, 2007.
33. K.H. Rosen. *Discrete Mathematics and Its Application*. McGraw-Hill, 2003.
34. R. Sabherwal and Y.E. Chan. Alignment between business and is strategies: A study of prospectors, analyzers, and defenders. *Info. Sys. Research*, 12(1):11–33, 2001.
35. D.J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, 2004.
36. P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.
37. 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.
38. 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.
39. W.M.P van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1128–1142, 2004.
40. B. Weber, M. Reichert, S. Rinderle, and W. Wild. Towards a framework for the agile mining of business processes. In *BPI'05*, LNCS 3812, pages 191–202, 2006.
41. 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.
42. B. Weber, M. Reichert, W. Wild, and S. Rinderle-Ma. Providing integrated life cycle support in process-aware information systems. *Int'l Journal of Cooperative Information Systems (IJCIS)*, 19(1), 2009.
43. B. Weber, S. Rinderle, and M. Reichert. Change patterns and change support features in process-aware information systems. In *CAiSE'07*, pages 574–588, 2007.
44. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integr. Comput.-Aided Eng.*, 10(2):151–162, 2003.
45. I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., USA, 2005.
46. 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.