Improving the Usability of Process Change Trees Based on Change Similarity Measures

Georg Kaes(✉) and Stefanie Rinderle-Ma

Faculty of Computer Science, University of Vienna, Vienna, Austria
{georg.kaes,stefanie.rinderle-ma}@univie.ac.at

Abstract. Flexible process management systems store information about conducted process change operations in change logs. Change log analysis can provide users who are responsible for planning and executing upcoming adaptations with valuable information. Change trees represent change logs emphasizing the temporal relation between change operations such that users can immediately see which change sequences have been applied in the past. Similar to most process mining approaches, change trees currently build upon label equivalence. However, labels only provide restricted information about a change operation. Hence this paper investigates how process change similarity can be employed to compare changes, i.e., similar change operations are aggregated in the tree as they appear in a change sequence. A user experiment shows the increased efficiency of the aggregated change sequences: users find relevant information faster than in a change tree based on label equivalence.

1 Introduction

Flexible business process management systems allow users to change their processes according to the current requirements [1]. Information about the conducted change operations are stored in a change log, which can be used to analyze past change operations.

For analyzing change logs, different approaches have been developed: While *change processes* [2] focus on causal dependencies between process change operations, the *change tree* [3] focuses on temporal relations between change operations, i.e. the inherent change sequences. Using a change tree, one can see in which ways a certain kind of process instance has evolved, and which evolution paths are more frequently used than others. Analyzing change sequences can provide valuable information: Imagine a nursing home with hundreds of patients, where each patient's therapy plan is represented by a process instance. Each task in this process instance represents an activity which has to be completed in order to improve the health status of a patient. Whenever an adaptation to a therapy instance is required (e.g., because the patient feels sick), his process instance has to be adapted, thus generating a new change in his change sequence. Information about the change sequence of a specific patient can be used to analyze what has been done, e.g., when evaluating the effectiveness of a patient's therapy plan. Following, it can be analyzed whether the patient's treatment was unique, or similar to other, comparable situations.

Change trees are a data structure based on process change logs which emphasize such change sequences. They are constructed by aggregating equal change operations on the same position of change sequences in a single node, and generating sibling nodes for different change operations. However, change trees compare process change operations based on their label only. If two change operations have the same label, they are considered to be equal. For many situations, such an assumption proves to be too general. Depending on the analysis question at hand, for example, it could be more interesting to compare process change operations based on the required resources, temporal constraints, or other attributes of the change operation. In other words, comparing change operations shall be shifted from *label equivalence* towards comparing their *semantics*.

The idea behind is to replace label equivalence with *change similarity measures* (cf. [4]). They calculate the similarity of two process change operations from different change perspectives [4], e.g., change attributes or change effects on the adapted process instance. This enables to compare change operations regarding, e.g., the required resources, time constraints, or affected control and data flow.

For example, the *Change Resource Similarity (CRS)* [4] calculates the similarity of the required resources of two change operations which insert activities into a process instance in an interval between 0 and 1: If two inserted process fragments require e.g. 3 nurses each, CRS returns 1, if they require totally different resources, it returns 0. If a nurse wants to know which resources are usually required for the treatment of a specific disease, she could investigate the change tree where similar change operations regarding CRS are aggregated in one node.

Replacing label equivalence by change similarity measures poses several challenges. If changes are similar, different options for merging them in the resulting change tree become possible. This necessitates theoretical considerations on how a change tree can be built in this case. Moreover, the interpretation of a change tree based on similarity measures is different from one based on label equivalence. Finally, it has to be investigated which benefits arise from employing change similarity instead of label equivalence for change trees. This leads to the following research questions:

RQ1: How can change sequences be analyzed in a semantic way? Which measures can be used? How do these measures have to be applied to a set of change sequences?

RQ2: Does the aggregation of change operations in change trees help the user to find relevant information faster than in change trees based on label equivalence?

In this paper, we first analyze how similarity measures for process change operations can be applied to sets of change sequences as they can be found in a change log, thus allowing us to answer different questions about change sequences

($\rightarrow$ RQ1). This is followed by an evaluation with potential users of the change tree ($\rightarrow$ RQ2).

This paper is structured as follows: Sect. 2 introduces the basic notations which are required for the remainder of the paper. This is followed by a general introduction to the application of process change operation similarity to change trees (Sect. 3). In Sect. 4, we show how similar and equal change operations can be aggregated. In our evaluation (Sect. 5) we present an experiment conducted with over 60 users who analyzed change sequences using our approach. The paper concludes with related work (Sect. 6) and a summary (Sect. 7).

2 Fundamentals

This paper focuses on the control flow aspects of process changes and defines process schema S as S := (N,E), where N denotes the set of nodes (activities and gateways), and E denotes the set of control flow edges (c.f. [5]). A change operation Δ transforms process schema S into adapted schema S', formally:

Definition 1 (Process Change Operation). *A process change operation Δ is defined as a tuple $\Delta := (t, f, p, S)$ where*

- *t denotes the type of the change operation.*
- *f denotes the process fragment which is used by the change operation.*
- *p denotes the position of the change operation.*
- *S denotes the process schema or instance the change operation is applied to.*

During runtime changes are applied to a set of process instances I running on a schema S. These changes are stored in a *change log* $\mathcal{C}$. Specifically, for each $i \in I$, $\mathcal{C}$ stores the temporally ordered set of change operations that have been applied to i. An example of a change log is given in the left pane of Fig. 1.

A *change tree* is a data structure representing change log information and is defined as follows:

Definition 2 (Change Tree, Based on [3]). *Let $\mathcal{C}$ be a change log and $\mathcal{D}$ be the set of all change operations contained in $\mathcal{C}$. Then the change tree T is defined as a rooted multiway tree $T := (r, V, E)$ with*

1. *$r := \emptyset$ is the unique root node*
2. *$V \subseteq \mathcal{D} \times \mathbb{N}_0$*
3. *$E \subseteq V \times V$*
4. *$\forall$ paths p from root r to node $v = (\Delta, n) \in V$ with $n > 0$: p corresponds to the change traces of n changed process instances in $\mathcal{C}$.*
5. *$\forall$ leaf nodes $v = (\Delta, n) \in V$: $n > 0$*

The change tree is constructed from a process change log. The root node represents the unchanged process schema. Each change operation which has been applied on a process instance is represented by a node in the tree. The traces in the change tree from the root nodes to the leaf nodes represent the temporal relationships between the applied change operations: If Δ_1 has been applied before Δ_2, it is closer to the root node.

Figure 1 shows an example for a change tree based on a process change log that consists of six traces $t_1, t_2, \ldots, t_6$. There are only two ways in which the process instances have been adapted in the beginning, i.e., Δ_1 and Δ_3. Following Δ_1, there is one trace where Δ_2 has been applied, and two traces where Δ_1 has been applied again. Both traces end at this point. Each node in a change tree contains two values: The first value is the label of the change operation which has been applied at this point in the change traces, the second value represents the number of change sequences which terminate at this change operation. This can be checked by consulting the change log, where indeed two change traces contain two applications of Δ_1, and one trace contains Δ_1 followed only by Δ_2.

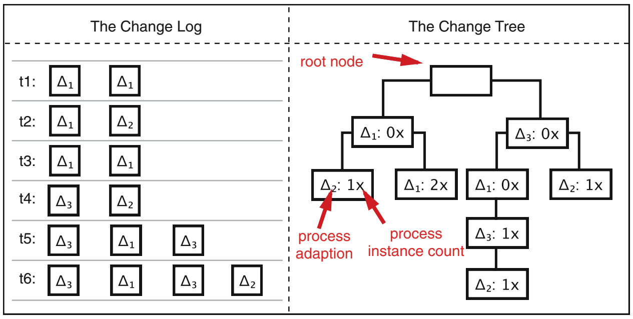

Fig. 1. Change tree representation of an example change log

The structure of the change tree emphasizes multiple features of the change log it is based on: The *breadth of the change tree* is an indicator for the diversity of the applied process change operations. The broader the change tree, the more diverse the change operations have been. The *height of the change tree* indicates how many change operations have been applied to one or more change traces. The number of change traces in a certain branch in relation to the total number of change traces in the log shows which change operation paths have been used more frequently. The diversity of the change operations which have been applied on a certain level can be seen from the number of *sibling nodes*.

So far, in a change tree, two (potentially sibling) change operations have only been aggregated in one node they were equivalent regarding their labels; the attributes and effects of the change operations were not considered at all. Using label equivalence has been discussed as a potential limitation in the area of process mining (cf. e.g., [6]). For example, it would be possible that two activities have equal labels, but launch different actions based on the context they are executed in, or that two activities with different labels are actually semantically equivalent (i.e., the launched actions are the same). Hence, for certain analysis questions, using an equivalence notion that refers to the semantics of activities would be useful [7]. The same holds for comparing change operations.

Hence, this work proposes the aggregation of nodes in change trees based on process change similarity instead of using label equivalence. In the following, we denote the similarity of two change operations Δ_1 and Δ_2 as $sim(\Delta_1, \Delta_2) \in [l, 1]$. For some similarity measures, $l = 0$ holds, for others $l = -1$ is defined. Which similarity measure is of interest, depends on the question at hand. When resources required to complete activities are sparse (e.g., nurses in the nursing domain, or professionals in the production domain), a resource-based view on the change log might be of interest. Using such a resource-based view, one can see more easily which resources have been required, and based on this information, it might be possible to optimize future process adaptations. For such a situation, the *Change Resource Similarity (CRS)* [4] can be used: It compares the resources used by the fragments of two change operations Δ_1 and Δ_2, which are inserted into or deleted from a process schema. If the resources are exactly the same, and both change operations insert or delete, it returns 1. If the resources are exactly the same, but one change operation inserts, and the other deletes, it returns -1. If the required resources are different, it returns a value between $(-)1$ and 0. Contrary, the attribute based similarity measure $sim_{attr}(\Delta_1, \Delta_2)$ returns a value in the range of $[0, 1]$ [4]. It is calculated by defining the similarity for each attribute of the two process change operations (fragments, positions, operation types and schemas), and weighing the results. Using weights enables to prefer certain attributes when comparing the change operations. Attributes can be even ignored by assigning a weight of 0 to them.

Definition 3 (Process Change Operation Similarity Measure). *Let Δ_1 and Δ_2 be two change operations as defined in Definition 1. Further, let $l \in \mathbb{R}$, $l < 1$. Then, the similarity measure is defined as $sim(\Delta_1, \Delta_2) \in [l, 1]$.*

Which process change operation similarity measure is used highly depends on the analysis question: If resources are of interest, a resource-based similarity metric such as CRS may be useful; if control-flow related features are of interest, a more attribute-based measure might be favorable. However, the approach presented in this paper abstracts from the implementation of concrete similarity measures. Thus, any measure which calculates a similarity $sim(\Delta_1, \Delta_2) \in [l, 1]$ can be used as a basis for creating the aggregated change tree.

3 Applying Similarity Measures to Change Logs

To discriminate between traditional change trees (based on label equivalence) and the change trees based on similarity measures as presented in this paper, we will refer to the latter by the term *aggregated change tree*. The term stems from the fact that multiple similar change operations will be *aggregated* in a single node. Figure 2 depicts an example of a change log, a change tree and an aggregated change tree. The middle pane shows a change tree according to Definition 2 based on the change log in the left pane: It consists of two levels, where the first level contains the four change operations Δ_1, Δ_2, Δ_3 and Δ_4. The change traces t_2 and t_3 additionally contain a second change operation, Δ_1. The right pane depicts the *aggregated change tree*. In the following sections, we will discuss how this aggregated change tree is created.

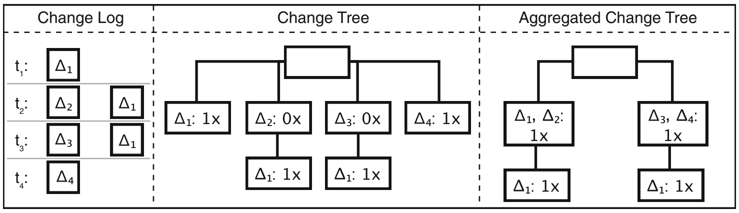

Fig. 2. Basic example

To preserve the change sequences of a change log, which are represented in a change tree, only change operations are considered for aggregation which would be on the same level of the change tree. While it would be possible to aggregate change operations in an aggregated change tree vertically (by aggregating change operations which happened consecutively), the horizontal aggregation as presented in this paper preserves the basic properties of the change tree: As the change tree, the aggregated change tree should, when read from the root to the leafs, represent the set of change sequences from a process log. If consecutive change operations would be aggregated as well, this property would be lost.

The only difference between the aggregated change tree lies within the structure of the nodes. Instead of a node V containing a single change operation (as defined in Definition 2), V now contains a set of similar change operations:

Definition 4 (Aggregated Change Tree). *Let $\mathcal{D}$ be a set of change operations as defined in Definition 2. An aggregated change tree $ACT := (r, V, E)$ is a change tree $T := (r, V, E)$ where $V \subseteq 2^{\mathcal{D}} \times \mathbb{N}_0$.*

There are two border cases for the number of nodes present in the aggregated change tree (ACT) compared to the change tree: Due to the horizontal aggregation, the minimum number of nodes equals the depth of the original change tree, since the ACT contains only one node for each level of the change tree. If however no nodes can be aggregated, the ACT contains exactly as many nodes as the change tree.

In order to decide whether two change operations Δ_1 and Δ_2 should be aggregated in a single node, a similarity threshold value $\tau \in [l, 1]$ is specified where l corresponds to the lower bound of the similarity measure of interest. If the similarity measure $sim(\Delta_1, \Delta_2) \geq \tau$, Δ_1 and Δ_2 are aggregated in a single node; else, two separate nodes are created. In the next section, we will present two approaches for generating aggregated change trees based on a similarity measure, a threshold τ, and a change log.

4 Generating the Aggregated Change Tree

Depending on the chosen similarity threshold τ, the aggregated change tree can be created in different ways. Section 4.1 presents the algorithms for $\tau < 1.0$ and Sect. 4.2 follows up with an illustration for $\tau = 1.0$.

4.1 Aggregation for Similar Process Change Operations

When aggregating similar change operations in one node, the similarity threshold τ has to be reached for all similarity measures between the change operations in the aggregation node. Figure 3 shows an example of such an aggregation: In pane 1, a change log is depicted which contains three change traces consisting of the process change operations Δ_1, Δ_2 and Δ_3. Pane 2 shows the similarity of the three change operations: $sim(\Delta_1, \Delta_2) = 0.75$, $sim(\Delta_2, \Delta_3) = 0.8$ and $sim(\Delta_1, \Delta_3) = 0.6$. Pane 5 shows the change tree which would be generated based on label equivalence (c.f. [3]). Depending on the chosen similarity threshold τ, different change operations can be aggregated in one node: If $\tau = 0.6$, all three change operations can be aggregated in one node (c.f. aggregated change tree depicted in pane 6). If $\tau = 0.7$, only change operations Δ_1 and Δ_2, or change operations Δ_2 and Δ_3 can be aggregated in one node, but not Δ_1, Δ_2 and Δ_3, since the similarity between Δ_1 and Δ_3 is below τ. The similarity measure is not transitive. Thus, for $\tau = 0.7$, two aggregated change trees are possible (c.f. panes 7 and 8). These two change trees are the results of two different aggregation strategies, which are explained at the end of this section.

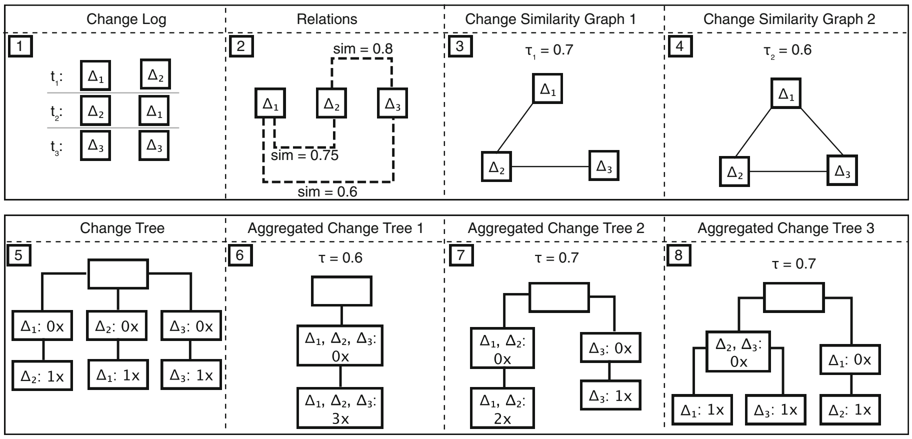

Fig. 3. Transitivity example

The similarity measures between all change operations which can be aggregated in one node need to be above the threshold τ. In order to find the subset of change operations for which this condition holds, we introduce the *change similarity graph*. (c.f. Definition 5).

Definition 5 (Change Similarity Graph). *Let $\mathcal{D}$ be a set of change operations, $sim(\Delta_i, \Delta_j)$ be a similarity measure, and τ the similarity threshold. The Change Similarity Graph $\Gamma := (\mathcal{D}, E)$ is an undirected graph where $E \subseteq \mathcal{D} \times \mathcal{D}$ denotes the set of edges. There exists an edge between change operations Δ_i, $\Delta_j \in \mathcal{D}$, iff $sim(\Delta_i, \Delta_j) \geq \tau$.*

Pane 3 and 4 of Fig. 3 show the similarity graphs for two thresholds $\tau_1 = 0.7$ and $\tau_2 = 0.6$ for the process change operations Δ_1, Δ_2 and Δ_3 from the example given above. In the similarity graph for $\tau_1 = 0.7$, there exist only edges between change operations Δ_1 and Δ_2, and between Δ_2 and Δ_3. There is no edge between Δ_1 and Δ_3, since its value lies below the threshold at 0.6.

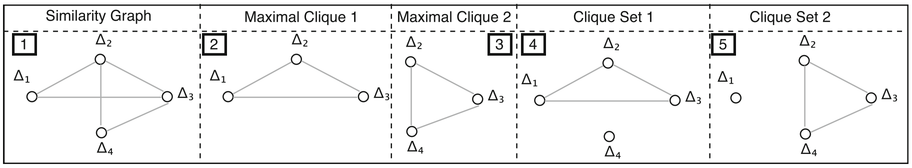

Fig. 4. Set of maximal cliques in a change similarity graph

Using the similarity graph as a basis, the problem of finding the subset of change operations where the similarity of all change operations exceeds the threshold τ can be seen as the problem of finding all maximal cliques in the graph. A maximal clique in an undirected graph is a *maximal complete subgraph* [8], that is a set of nodes and edges from that graph, where there exists an edge between each node. Using a set of cliques as a basis, one could aggregate all change operations which are in the same clique, since the similarity of those change operations exceeds τ.

The graph depicted in the third pane of Fig. 3 ($\tau = 0.7$), contains two cliques: One contains change operations Δ_1 and Δ_2, the other contains change operations Δ_2 and Δ_3. In the graph in the fourth pane ($\tau = 0.6$), there exists one clique which contains all three change operations. The *Bron-Kerbosch algorithm* [8] takes an undirected graph such as the change similarity graph as input, and returns all maximal cliques of this graph. Figure 4 shows an example for a set of maximal cliques in a change similarity graph. Pane 1 shows the similarity graph which contains four different process change operations. By applying the Bron-Kerbosch algorithm [8], two maximal cliques (depicted in panes 2 and 3) are generated. Obviously, Δ_2 and Δ_3 appear in more than one clique. If both cliques were used directly to generate the nodes of the aggregated change tree, Δ_2 and Δ_3 would appear in more than one node.

Following, the similarity graph has to be split up into a *unique clique set*, containing several cliques where each node is present in exactly one clique. This can be done in a straightforward fashion: We start with a similarity graph $\mathcal{G}$ and an empty unique clique set $\mathcal{U}$. In the first step, the cliques of $\mathcal{G}$ are created, all nodes contained in the biggest maximal clique in $\mathcal{G}$ are removed from the graph, and this biggest maximal clique is added to $\mathcal{U}$. This step is now repeated with the remaining nodes of the graph, until no nodes are left in the graph. If there exist multiple biggest maximal clique sets, multiple solutions are possible, as seen in the following example:

Applying this procedure to the example given in Fig. 4 yields the following results: We start with an empty unique clique set $\mathcal{U} = \emptyset$. If we start with the first maximal clique $\{\Delta_1, \Delta_2, \Delta_3\}$ (depicted in pane 2) this clique is added to $\mathcal{U}$, and these nodes are removed from the similarity graph. Following, the similarity graph now only contains $\{\Delta_4\}$, which is the last maximal clique left in the graph; thus, it is added to $\mathcal{U}$, resulting in the unique clique set depicted in pane 4. If we start with the second maximal clique set $\{\Delta_2, \Delta_3, \Delta_4\}$ (depicted in pane 3), $\{\Delta_1\}$ would be the second clique, and we would end up with the unique clique set depicted in pane 5. Thus, two unique clique sets are created.

These two unique clique sets can now be used to generate the change tree nodes. Depending on the which set is chosen, either Δ_1, Δ_2 and Δ_3 are aggregated in one node, and Δ_4 is in the other node, or Δ_2, Δ_3 and Δ_4 are aggregated in a node, and Δ_1 is in its own node. Two distinct features can be used as a basis for deciding which process change operation should be aggregated in a node: (a) the most similar change operations should be aggregated or (b) the number of nodes in the tree should be minimized.

Figure 5 shows a change log with four change operations Δ_1 to Δ_4. Pane 2 depicts the similarity relations between them. In the remainder of this section, this example will be used to explain the different aggregation strategies.

Aggregating the Most Similar Change Operations. Algorithm 1 shows the strategy which can be used to aggregate the most similar change operations on one level into an aggregated change tree node. It is based on the creation for change trees as defined in [3] and starts with the first level of change operations in the log, containing Δ_1 to Δ_4 (c.f. example from Fig. 5).

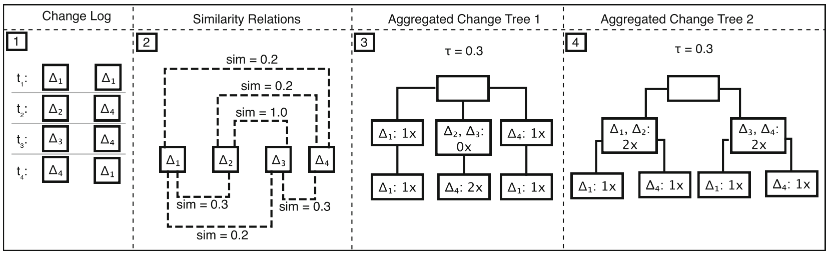

Fig. 5. Example change tree for different strategies

First, the unique clique sets are created. For a $\tau = 0.3$, two unique clique sets are created: $\{\{\Delta_1\}, \{\Delta_2, \Delta_3\}, \{\Delta_4\}\}$ and $\{\{\Delta_1, \Delta_2\}, \{\Delta_3, \Delta_4\}\}$. The optimal clique set is defined as the unique clique set where the overall similarity of the contained nodes is maximal. This is calculated and compared to the similarity of other unique clique sets in the function *get-optimal-cliqueset*. Initially, it iterates over a set of unique clique sets. For each unique clique set, if a clique contains more than one Δ, the similarity of all change operations in this clique is calculated, and its sum is divided by the number of total nodes in the clique. For the first clique set containing 3 cliques, the similarity of Δ_2 and Δ_3 is 1.0, which results in an overall similarity of $\frac{1}{3}$. For the second unique clique set containing 2 cliques, the similarity is 0.3 for both cliques, thus, the resulting similarity is also 0.3. Since $\frac{1}{3} > 0.3$, the first clique set $\{\{\Delta_1\}, \{\Delta_2, \Delta_3\}, \{\Delta_4\}\}$ is chosen. Based on this clique set, for each clique, a node is created and the traces are added for creating the child nodes with the change operations in the next recursive iterations. For each change operation in the clique, the corresponding change trace is checked: If the change operation is the last change operation of this trace, the node's counter is incremented; else, the trace is added to the traces, which will be considered in the next recursive call. Thus, all relevant traces and change operations will be considered for the child level. The resulting aggregated change tree is presented in pane 3 of Fig. 5.

Algorithm 1. Aggregating the most similar change operations

```
Input:
  – Change log $\mathcal{C}$
  – Similarity measure $sim(\Delta_1, \Delta_2)$
  – change similarity graph $\Gamma := (\mathcal{D}, E)$

Output: Aggregated Change Tree ACT
Begin root = create-node(null,null);
create-children(root, $\mathcal{C}$, 0); return $\mathcal{ACT}$
function create-children(parent,traces,position)
    nodes = $\emptyset$, deltas = $\emptyset$, cliquesets = $\emptyset$
    foreach trace $\in$ traces do
        deltas = deltas $\cup$ {trace.get-changeoperation(position)}
    generate-cliqueset($\mathcal{D}$, $\emptyset$, cliquesets)
    //variable cliquesets now holds the set of clique sets
    optimal-cliqueset = get-optimal-cliqueset(cliquesets)
    //variable optimal-clique set now holds the optimal clique set
    foreach clique $\in$ optimal-cliqueset do
        //creates a new node containing the change operations and links it to the parent
        node
        node = create-node(clique.changeoperations, parent)
        nodes = nodes $\cup$ {node}
        foreach delta $\in$ clique do
            //get-trace-of(delta) finds the trace this change operation is contained in
            trace = get-trace-of(delta)
            if trace.get-changeoperation(position+1)==null then
                node.count++
            else
                node.nexttraces = node.nextraces $\cup$ {trace}
    foreach node in nodes do
        create-children(node,node.nexttraces,position+1)
function get-optimal-cliqueset(cliquesets)
    highest-sim = 0, optimal-set = $\emptyset$
    foreach cliqueset $\in$ cliquesets do
        sim = 0
        foreach clique $\in$ cliqueset do
            if |clique| > 1 then
                //calculate similarity in clique
                sim = $\frac{\sum_{i,j \in clique, i \neq j} sim(i,j)}{|clique|}$
        if optimal-set == $\emptyset \vee$ sim > highest-sim then
            highest-sim = sim, optimal-set = cliqueset
```

Minimizing the Number of Nodes. The second strategy minimizes the number of nodes in each level of the aggregated change tree. Algorithm 2 shows this strategy. It replaces the function *get-optimal-cliqueset* with a different algorithm which prefers the unique clique set with the minimum number of cliques. Since each clique is represented by a node in the aggregated change tree, this results in the lowest number of nodes *per level*. For the first call of the function in the example given in Fig. 5, this would return the clique set $\{\{\varDelta_1, \varDelta_2\}, \{\varDelta_3, \varDelta_4\}\}$, thus resulting in the aggregated change tree depicted in pane 4.

Algorithm 2. Generating the minimum amount of nodes per level

```
function get-optimal-cliqueset(cliquesets)
    min-count = 0, optimal-set = $\emptyset$
    foreach cliqueset $\in$ cliquesets do
        if optimal-set == $\emptyset \vee$ min-count < |cliqueset| then
            min-count = |cliqueset|, optimal-set = cliqueset
```

Discussion: We have presented two strategies for aggregating similar change operations in change tree nodes. Which one should be used highly depends on the structure of the change log: For most situations, aggregating the most similar change operations in one node seems logical. In the nursing home, for example, one could aggregate the most similar therapy adaptations in a single change tree node. If however the resulting number of nodes would become very large due to the structure of the change log, minimizing the number of change tree nodes instead can enhance the tree's readability, thus simplifying further analysis of the change log.

4.2 Aggregation for Equal Process Change Operations

For the aggregation of equal change operations ($\tau = 1.0$), no change similarity graph is required, since the equality relation is transitive ($\text{sim}(\varDelta_1, \varDelta_2) = 1.0 \wedge \text{sim}(\varDelta_2, \varDelta_3) = 1.0 \implies \text{sim}(\varDelta_1, \varDelta_3) = 1.0$). The example we use to explain the aggregation of equal change operations is based on the change log as presented in Fig. 2[1]. While the middle pane shows the change tree based on label equivalence as defined in [3], the right pane shows the resulting aggregated change tree. Assume that $sim(\varDelta_1, \varDelta_2) = 1.0$, $sim(\varDelta_3, \varDelta_4) = 1.0$, and all other similarities <1.0. The aggregated change tree is created as follows: First, an empty root node with no label and no parent is created. Next, each "column" in the change log is iterated over: The first column contains change operations $\varDelta_1, \varDelta_2, \varDelta_3$ and $\varDelta_4$. The second column contains $\varDelta_1$ (following $\varDelta_2$ respectively $\varDelta_3$). For the first column, the similarities between all change operations are checked: If $sim(\varDelta_i, \varDelta_j) = 1.0$, the change operations are aggregated in one node - if not, they are split up in two nodes. In this example $sim(\varDelta_1, \varDelta_2) = 1.0$ and

[1] The full version of the algorithm can be found on our project homepage https://cs.univie.ac.at/project/APES.

$sim(\Delta_3, \Delta_4) = 1.0$. All other similarities are smaller than 1. The same is done for the second column. Note that the Δ_1 of the second column cannot be aggregated in one node, since they have distinct parent nodes (one containing Δ_1, Δ_2, and one containing Δ_3, Δ_4). Ultimately the aggregated change tree depicted in pane 3 of Fig. 2 is created.

5 Evaluation

The approach presented in this paper is evaluated in an experiment which aims at two goals, i.e., to analyze whether (a) the change tree can be used as a basis for analyzing change sequences and (b) certain questions can be answered faster by participants using an aggregated change tree, than using a change tree.

Method and Experiment Design. The experiment is based upon a prototypical implementation of the (aggregated) change tree, working in every modern browser using HTML, CSS and JavaScript. This implementation serves as a proof of technical feasability, which can also be accessed without participating in the experiment[2].

The first draft of the experiment was refined using a two stage pretest: During stage one, the questions of the experiment were discussed with three peers who are familiar with the concept of process change operations and change trees. In the second phase, persons from the target audience were asked to participate in the experiment. The target audience of the aggregated change tree are persons from different fields of work, who do not necessarily have to be familiar with process change operations. Thus, the chosen participants were older than 18 years, and were not required to have any knowledge regarding process change operations, or change trees. During the pretest, two major parts of the experiment were optimized: First, the user interface was optimized to be more intuitive to users who are not familiar with the setting, second, the wording of two questions and their explanation was optimized.

The experiment consists of two parts: Part 1 addresses the question whether the change tree (CT) is a better representation of change logs than their raw XML format (XML). Specifically, we want to find out if certain questions regarding the change log (e.g. *How many resources have been used in this change log?*) can be answered faster using a change tree than an XML based log file. The participants were split in two groups and had to answer the same questions based on those two representations. In each question, we measured the time until the (correct) answer was found. The questions for the two representations were based on different, but similar log files to prevent bias due to the participants knowing the correct answer: While group 1 answered questions for the CT based on log A, and questions for XML based on log B, group 2 received the CT questions based on log B, and the XML questions based on log A. This setup also helped us to mitigate the risk of a bias due to different log files: If there was only one

[2] The prototype, the experiment including all questions, and all related data can be found at https://cs.univie.ac.at/project/APES.

group who answered all CT questions based on log A, and all XML questions based on log B, the differences in speed and correctness could be due to the slight differences between those log files, and not due to their representation. To mitigate the risk of distortion due to a learning effect, the order of the questions and representations was randomized. Part 2 of the experiment addresses the usability of the aggregated change tree (ACT) in comparison to the CT. The setup and the participants were the same as in Part 1; however, two log files C and D (different from those of Part 1) were used.

At the beginning of the experiment, the participants were introduced into the topics of processes, process adaptations, change logs and (aggregated) change trees. Following this 10 min presentation, the participants had the possibility to ask questions, and received a link to the experiment. They had three days to participate in the experiment. Such an *uncontrolled experiment setup* resembles a realistic setting, in which aggregated change trees would be used, more closely.

Results: In total, 64 participants participated in the experiment. These participants were divided in two groups as described above. 31 participated in group 1, and 33 participated in group 2. First, the data set was cleaned for participants who obviously entered random numbers. The check was based on two parameters: (1) most of the answers were wrong, and (2) all answers were given in less than two seconds. Answers given that quickly can be seen as an indicator for a participant entering wrong answers on purpose. These patterns were found for one participant from group 1, resulting in 30 valid participants in group 1, and 33 valid participants in group 2.

Table 1. Results of the experiment

XML Log vs CT	Overall		Group 1		Group 2	
	Log	CT	Log	CT	Log	CT
Correct answers	61.11%	68.25%	61.67%	68.33%	60.61%	68.18%
Mean time (sec.)	90.02	35.43	95.28	33.12	83.37	39.97
Median time (sec.)	58.69	29.2	57.51	27.32	59.43	32.14
CT vs ACT	Overall		Group 1		Group 2	
	CT	ACT	CT	ACT	CT	ACT
Correct percentage	65.08%	67.46%	71.67%	68.33%	59.09%	66.67%
Mean time (sec.)	112.54	43.97	138.08	52.84	84.39	35.71
Median time (sec.)	51.73	22.43	61.82	24.18	48.28	21.4

Table 1 summarizes the results from the experiment. The mean and median times refer to the amount of time needed by the participants to find the correct answers. Using a two sample t test, the differences regarding the amount of correct answers given are not significant ($p > 0.05$): the correctness of the answers was not negatively influenced when using CT compared to the XML Log and

ACT compared to CT respectively. However, the time required to find the correct answer is statistically significant for both parts of the experiment ($p < 0.05$): In the first part of the experiment, it can be seen that the mean and median times required to find the correct answer for the CT is much lower than for the XML; in the second part, it can be seen that participants using the ACT require even less time to find the correct answer than participants who use the CT.

Discussion and Threats to Validity: The experiment shows that the (A)CT can be used as a representation of change logs, since the amount of correct answers is not statistically different from the amount in the XML Log. This effect may also be due to the XML Log being very simple: For Part 1 of our experiment we chose a log with only four change operations in total, in order to not demoralize participants who had to work with the XML representations. Thus, the change log was designed in favor of the XML representation, which may pose a threat to the validity. However, even in such a setting, the time required for the correct answers was statistically significantly lower for CT than for XML Log. For the ACT, the number of correct answers is not statistically significantly different from CT, but significantly less time is required to find a correct answer. Although the uncontrolled setup of our experiment increases the resemblance of our experiment to a realistic setting in contrast to a controlled setup, this decision also poses a threat to validity, since participants have been able to communicate with each other while participating. However, participants who completed the experiment did not receive the correct answers to the questions, and such behavior may also be found in a realistic setting, where users would have to analyze change trees. Thus, the results can be seen as being more realistic, while the negative impact of the threat has been kept to a minimum.

6 Related Work

In [9], the authors present an overview over existing approaches to determine the similarity of two process models. The presented approaches range from label matching similarity, over feature-based similarity estimations up to comparisons of common nodes and edges in process model graphs. These approaches can also be interesting for change operation similarity measures, since they can be used to compare attributes of change operations, like the schema, the applied fragment, or the position of the change. *Van Dongen et al.* discuss causal footprints, *which is a collection of the essential behavioral constraints imposed by a process model* [10] as a representation of a process model's behavior. Using causal footprint's *look-back* and *look-ahead* links, one could analyze and compare the positions of process change operations.

The work presented in [11] provides similarity measures for process instances. To the best of our knowledge, [4] is the only approach dealing with process change similarity. This work exploits the measures proposed in [4], but any other change similarity measure can be used as well.

In contrast to change trees, change processes [2] are a different method to analyze process change logs. Change processes focus on causal relations between process change operations. If a change operation Δ_1 is followed by Δ_2, it means that Δ_2 may be caused by Δ_1. For parallel change operations, such a condition cannot be drawn from the change log.

While change logs provide a solid basis for analyzing conducted change operations and their consequences, they may not always be available. Concept drift [12,13] focuses on analyzing process execution logs to detect any executed changes on the basic schema. By analyzing the activities present in the change log over a long period of time, this approach can estimate if and which changes have been applied to a process schema.

7 Conclusion and Future Work

In this paper we have discussed how process change operation similarity measures, which calculate the similarity of two process change operations, can be extended to sets of change sequences, i.e. change logs. Change trees serve as a basis, since they provide a compact representation of change sequences. An analysis of the similarity of change sequences can provide interesting insights for many situations in different application domains: In the nursing home, for example, nurses can compare sets of therapies which have been applied to different patients over a long time. Using different similarity measures, different features of the change sequences can be analyzed, e.g. the diversity of required personnel, resources or time. The presented approach can be used with any similarity measure which returns a numeric value within a certain range; thus, it is very flexible and extensible. In a practical setting, the choice of the similarity metric has a huge impact on the information being displayed in a change tree. Thus, future work will analyze the impact of using different similarity notions and aggregation strategies in practical settings.

References

1. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30409-5
2. Günther, C., Rinderle-Ma, S., Reichert, M., van der Aalst, W.: Using process mining to learn from process changes in evolutionary systems. Int. J. Bus. Process Integr. Manage. **3**, 61–78 (2008)
3. Kaes, G., Rinderle-Ma, S.: Mining and querying process change information based on change trees. In: Barros, A., Grigori, D., Narendra, N.C., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9435, pp. 269–284. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48616-0_17
4. Kaes, G., Rinderle-Ma, S.: On the similarity of process change operations. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 348–363. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_22
5. Allen, F.E.: Control flow analysis. ACM Sigplan Not. **5**, 1–19 (1970)
6. Ingvaldsen, J.E., Gulla, J.A.: Industrial application of semantic process mining. Enterp. Inf. Syst. **6**, 139–163 (2012)

7. Rinderle-Ma, S., Reichert, M., Jurisch, M.: On utilizing web service equivalence for supporting the composition life cycle. Int. J. Web Serv. Res. **8**, 41–67 (2011)
8. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM **16**, 575–577 (1973)
9. Becker, M., Laue, R.: A comparative survey of business process similarity measures. Comput. Ind. **63**, 148–167 (2012)
10. Van Dongen, B., Dijkman, R., Mendling, J.: Measuring similarity between business process models. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 450–464. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69534-9_34
11. Pflug, J., Rinderle-Ma, S.: Process instance similarity: potentials, metrics, applications. In: Debruyne, C., et al. (eds.) OTM 2016. LNCS, vol. 10033, pp. 136–154. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48472-3_8
12. Bose, R.P.J.C., van der Aalst, W.M.P., Žliobaitė, I., Pechenizkiy, M.: Handling concept drift in process mining. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 391–405. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21640-4_30
13. Carmona, J., Gavaldà, R.: Online techniques for dealing with concept drift in process mining. In: Hollmén, J., Klawonn, F., Tucker, A. (eds.) IDA 2012. LNCS, vol. 7619, pp. 90–102. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34156-4_10