Discovering Workflow Changes with Time-Based Trace Clustering

Rafael Accorsi, Thomas Stocker

Discovering Workflow Changes with Time-based Trace Clustering

Rafael Accorsi and Thomas Stocker

Albert-Ludwigs-Universität Freiburg, Germany
{accorsi,stocker}@iig.uni-freiburg.de

Abstract. This paper proposes a trace clustering approach to support process discovery of configurable, evolving process models. The clustering approach allows auditors to distinguish between different process variants within a timeframe, thereby visualizing the process evolution. The main insight to cluster entries is the "distance" between activities, i.e. the number of steps between an activity pair. By observing non-transient modifications on the distance, changes in the original process shape can be inferred and the entries clustered accordingly. The paper presents the corresponding algorithms and exemplifies its usage in a running example.

1 Introduction

Process discovery aims to reconstruct process models from execution logs. Specifically: given a log, each of the cases (i.e. execution traces) is analyzed, eventually producing a (Petri net) model of the process. The reconstructed model provides a basis for other process mining techniques, e.g. conformance checking and case prediction. Such techniques are not only useful for process designers, but essential tools for auditors analyzing process-aware information systems (PAIS).

However, process discovery techniques have problems with regard to the quality of produced models. Especially the emerging trend of PAIS to allow for configurable process models whose structure changes along time poses a challenge for their precision. Roughly speaking, process discovery consolidates all the different process instances into a single model which produces a coarse view of the underlying process.

Trace clustering techniques act as a preprocessing step for process mining [7,8,12], thereby allowing for a fine-grained set of models. The idea is to group traces according to different characteristics and, subsequently, mine a particular set of clusters. While clustering allows for the selective reconstruction of traces, it still fails to mine the complete "history" (i.e. *evolution provenance*) of a business processes, identifying their diverse "tenancies" and how they differ.

This paper presents an approach for time-oriented log-clustering that is able to directly reflect the dynamics of a process's structure. The goal is to cluster the cases belonging to one process structure, while different clusters mean that the process structure varies. The variation happens, e.g., when activities are *added* to (or *deleted* from) the process model or when the *order* of activities is

Fig. 1. Approach overview.

changed. Thinking of process logs as sequential records of triggered activities, these variations are reflected in the log as modifications on the "distance" (i.e. number of entries) between pairs of activities. For instance, if an activity C is inserted between the activities A and B, then the distance between them increments by 1. Our approach clusters traces according to the time point where process variants have been introduced. In doing so, we obtain a chronological ordering of process tenancies which allows an auditor to appreciate the evolution of the original process.

Fig. 1 depicts our approach consisting of two steps. *Firstly*, the distance between the activities is measured. *Secondly*, the traces with similar "distance" behavior are clustered, so that process discovery techniques can be applied. Identified clusters can then serve as input for arbitrary discovery algorithms to construct process models for further analysis, e.g. [2,3].

Altogether, this paper provides the following contributions:

- It introduces the "activity distance" as a basis for trace clustering (Sec. 2).
- It provides a clustering method and the corresponding algorithms (Sec. 3).
- It reports on tests employing the approach and discusses the results (Sec. 4).

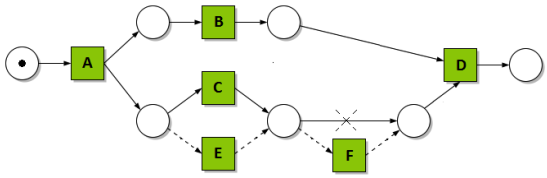
The overall goal is to devise techniques powerful and scalable enough to cope with industry-size data. The current implementation of the technique is a standalone application. In future, the technique is going to be added to ProM [14] and be part of the Security Workflow Analysis Toolkit (SWAT) [4], thereby making it possible for people to experiment with it.

2 Extracting Activity Distances

This section introduces the shape of logs taken into account, introduces the concept of "distance", and defines algorithms to extract the distances.

2.1 Logs and Distances

We assume that log entries have a timestamp and are chronologically ordered. Fig. 2 shows a workflow as a Petri net with a corresponding log. Adding the



Trace	Activities
1	A B C D
2	A C B D
3	A C B D
4	A B C D
5	A B C F D
6	A E F B D
7	A B E F D

Fig. 2. Workflow net and corresponding log.

dashed edges leads to another version of this workflow with the two additional activities E and F. Traces (also called cases) related to the new workflow structure appear in the log below the dashed line. Petri nets are often used as a meta-model to describe workflow models. They consist of transitions (rectangles) that stand for workflow activities and places (circles) that can contain tokens used to model the control flow. The execution of activities is modeled in terms of "firing" transitions. Transitions fire if they are "enabled" which means that there is at least one token in each input place. Upon firing, they take one token from each input place and put one token in each output place. The net in Fig. 2 contains a single enabled transition which is A. See [11] for details on Petri nets.

Process logs and traces are defined as follows:

Definition 1. *Let A be a set of activities of a process. $\sigma \in A^*$ is a log trace and $L = \{\sigma_1, ..., \sigma_n\} \in \mathcal{P}(A^*)$ is a process log. The order of traces is given by the starting time of the corresponding instance and reflected on the indexes σ_i.* ⊣

Loops in process models can cause activity names to occur several times within the same trace. For the sake of simplicity, we do not consider loops. The distance between any two process activities within a trace is defined as the number of intermediate activities. Considering the first trace of the log in Fig. 2, the distance between A and D is 2. Formally distance is defined as follows:

Definition 2. *Let $\sigma = \{a_1, ..., a_n\}$ be a trace containing activities a_i. The **distance** between any two subsequent trace-activities $a_i, a_j \in \sigma$, $j>i$ is defined as $d_\sigma(a_i, a_j) := j - i - 1$.* ⊣

If two activities happen sequentially within a process, their distance remains constant over time, otherwise the distance varies in fixed boundaries conditioned by a minimum and maximum distance. Consider the activity pair (A, D) in Fig. 2: $d(A, D) = 2$ holds within the first 4 traces, whereas the distance $d(B, D)$ ranges in the interval $[0; 1]$.

Structural changes in the process cause interval variations for activity-pair distances. Fig. 3 shows the distance progress for the activity pair (B, D) in Fig. 2. Within the first four traces the distance remains inside the interval $[0, 1]$, but with trace 5 this interval is enlarged to $[0, 2]$. Depending on the number

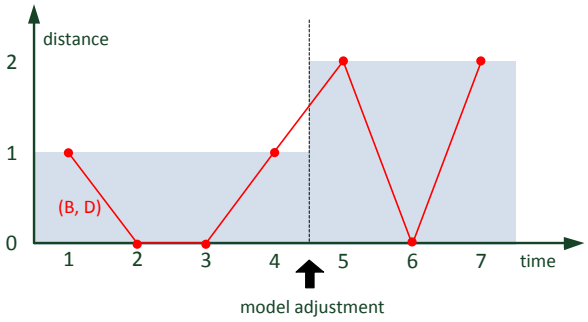


Fig. 3. Distance graph of the activity pair (B,D).

Trace	Activities	Distances						
		(A,B)	(A,C)	(A,D),	(B,C)	(B,D)	(C,B)	(C,D)
1	A B C D	0	1	2	0	1	-	0
2	A C B D	1	0	2	-	0	0	1
3	A C B D	1	0	2	-	0	0	1
4	A B C D	0	1	2	0	1	-	0
	intervals	[0;1]	[0;1]	2	0	1	0	[0;1]

Fig. 4. Distance matrix for the upper part of the log in Fig. 2.

and position of inserted or removed workflow activities and links between them, distance intervals are enlarged, reduced or moved (same value range, but different starting point, i.e. $[1,3] \rightarrow [2,4]$).

2.2 Extracting Activity Distances

Algorithm 1 constructs a distance matrix containing distances of activity-pairs as a preprocessing step for the clustering procedure presented in Sec. 3. For every pair of subsequent activities, this matrix contains their distance within every trace of the log. Fig. 4 shows the distance matrix for the traces 1–4 in Fig. 2.

Definition 3. *Let $L = \{\sigma_1, ..., \sigma_n\}$ be a log and A the set of activities in L. A distance matrix M_L holds for every $a', a'' \in A$ a distance list $D_{a',a''} := [d_{\sigma_1(a',a'')}, ..., d_{\sigma_n(a',a'')}]$. Distances for a specific index $k \in [1;n]$ are represented by $D_{a',a''}[k] := d_{\sigma_k(a',a'')}$. The number of observations within a set $D'_{a',a''} \subseteq D_{a',a''}$ is defined as: $obs(D'_{a',a''}) := |\{d_{\sigma(a',a'')} \in D' \mid d_{\sigma(a',a'')} \neq null\}|$.* ⊣

Inserted "null" values (denoted −) stand for the fact that not all the traces contain all possible activitiy-pairs. This ensures that the distance lists have equal sizes, which is required by the clustering algorithm. Alg. 1 shows how the distance matrix is built.

Algorithm 1 Build distance matrix

```
procedure BuildDistanceMatrix(workflow log L = {σ_1, σ_2, ...})
    for all k = 1, |L| do
        for i = 1, |σ_k| − 1 do
            for j = i, |σ_k| do
                D_{σ_k[i],σ_k[j]}[k] ← (j − i − 1)
            end for
        end for
        insert null for all other activity-pairs.
    end for
end procedure
```

3 Clustering Log Traces

Our trace clustering approach is interactive and consists of two phases. The first phase determines the *cluster cuts* for every observed activity pair (a', a''), i.e. the time-points (trace numbers) at which distance interval changes of (a', a'') suggest to end one cluster and to begin a new one. The second phase merges these individual results by counting the number of activity pairs at time point i that would cut the log at that position. As a result, one can browse through a chart showing these counts and, starting with a rather low window size, check how changing this parameter affects the accumulated cluster cuts.

3.1 Determining Cluster Cuts

Sequentially processing traces according to their timestamp (in this case, trace index), the clustering algorithm uses samples to determine the typical workflow behavior in terms of boundaries for the minimum and maximum observed value of activity distances. Such a behavior is determined on the basis of a parameter w (window size) that defines the minimum number of traces used as "training" data (sample size) and the minimum number of traces grouped as a cluster. Note that such a step is required since "change" always relates to typical behavior that has to be defined or measured first. The initial sample contains the first w traces of the log. As long as following traces show typical behavior, they are clustered together, otherwise a new sample of size w is created and used as a basis to determine the "new" typical behavior. The following defines these notions:

Definition 4. *Let $W_{s,t} = \{\sigma_s, ..., \sigma_t\} \subseteq L$ be a sample of a log $L = \{\sigma_1, ..., \sigma_n\}$ with size $|W_{s,t}| \geq w$. The minimum and maximum distances of any two activities a', a'' within W are defined as $min_{a',a''}(W_{s,t}) := min(d_{\sigma_s}(a', a''), ..., d_{\sigma_t}(a', a''))$ and $max_{a',a''}(W_{s,t}) := max(d_{\sigma_s}(a', a''), ..., d_{\sigma_t}(a', a''))$. The corresponding distance interval is defined as $[min_{a',a'}(W_{s,t}); max_{a',a'}(W_{s,t})]$. The projection of a distance list on a sample is defined as: $D_{a',a''}|W_{s,t} := \{d_{\sigma_s(a',a'')}, ..., d_{\sigma_t(a',a'')}\} \subseteq D_{a',a''}$.* ⊣

Definition 5. *The support of a distance γ between (a', a'') within a window W is defined as: $supp_{a',a''}(W_{s,t}, \gamma) := |\{d_\sigma(a', a'') \in W \mid d_\sigma(a', a'') = \gamma\}|$.* ⊣

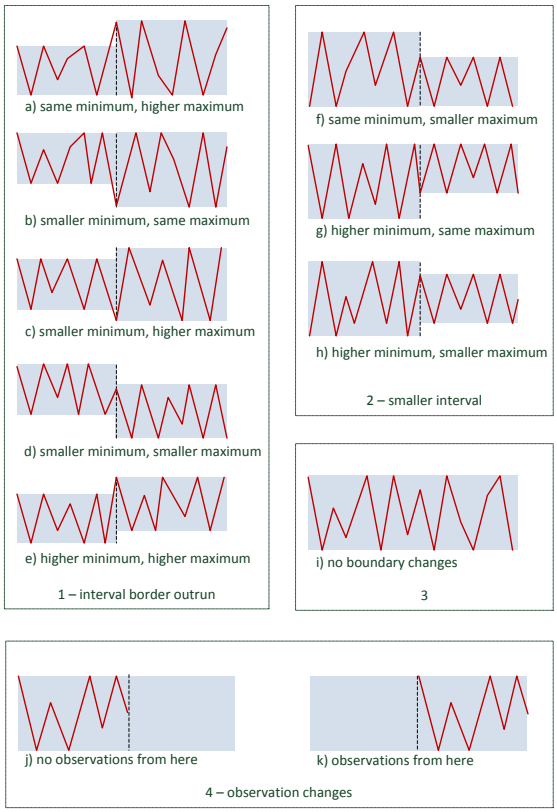


Fig. 5. Possible changes for distance intervals.

Considering a sample $W_{s,i-1}$ that holds all traces accumulated so far, for every (a', a'') of the next trace σ_i it is checked if the actual distance value of (a', a'') changes the corresponding distance interval in $D_{a',a''}|W_{s,i-1}$. To distinguish between momentary and persistent changes, the clustering method uses a w-size lookahead $W_{i,i+w-1}$. Fig 5 shows the possible cases of distance interval changes. In detail there are four different types of interval changes that introduce new clusters:

1. **(C1) Interval border outrun**. The actual distance of (a', a'') *outruns* its typical distance interval within the sample (cases in Group 1 in Fig. 5). To check the persistency of this change, the support of the new distance $d_{\sigma_i}(a', a'')$ within the lookahead is checked, using a threshold τ.
case condition: $d_{\sigma_i}(a', a'') > max(W_{s,i-1}) \vee d_{\sigma_i}(a', a'') < min(W_{s,i-1})$.
persistency condition: $supp_{a',a''}(W_{i,i+w-1}, \tau) \geq \tau$.

2. **(C2) Smaller interval**. The actual distance of (a', a'') *remains inside* the typical interval but persistently reduces the size of the corresponding distance interval (cases in Group 2 in Fig. 5).
   case condition: $d_{\sigma_i}(a', a'') \geq min_{a',a''}(W_{s,i-1}) \wedge d_{\sigma_i}(a', a'') \leq max_{a',a''}(W_{s,i-1})$
   persistency condition: $(min_{a',a''}(W_{i,i+w-1}) > min_{a',a''}(W_{s,i-1}) \wedge max_{a',a''}(W_{i,i+w-1}) \leq max_{a',a''}(W_{s,i-1})) \vee (max_{a',a''}(W_{i,i+w-1}) < max_{a',a''}(W_{s,i-1}) \wedge min_{a',a''}(W_{i,i+w-1}) \geq min_{a',a''}(W_{s,i-1}))$
3. **(C3) Distance observations from here**. While there are no distance observations for (a', a'') within $W_{s,i-1}$, the actual trace σ_i provides a distance value (case j in Group 4 in Fig. 5).
   case condition: $obs(D_{a',a''}|W_{s,i-1}) = 0 \wedge d_{\sigma_i}(a', a'') \neq \textbf{null}$
4. **(C4) No distance observations from here**. While there were distance observations for (a', a'') in $W_{s,i-1}$, the actual trace σ_i contains a phase where no distance values for (a', a'') are observed (Case k in Group 4 in Fig. 5).
   case condition: $obs(D_{a',a''}|W_{s,i-1}) \neq 0 \wedge obs(D_{a',a''}|W_{i,i+w-1}) = 0$

As long as the distances of observed traces do not introduce new distance intervals, they are put in the same cluster. Once a new interval is detected, the typical behavior is calculated again by choosing a new sample on the basis of the next w traces. This approach allows the identification of modifications and the time point from which they hold, thereby allowing time-based clustering. If distance values were observed both for (a', a'') and (a'', a') within the actual cluster $W_{s,i-1}$, the activities a' and a'' are likely to be in parallel. Since there typically are lots of interval changes within the same process model for parallel activities, distance interval changes for such activity pairs are not considered as indicators for workflow changes.

3.2 Consolidating Cluster Cuts

After determining the cluster cuts for every observed activity pair individually, the suggested cut positions are combined by counting the number of activity pairs for every time point (trace number) that have a cluster cut at this position and visualized as cluster cut graphs. With the help of a prototypical implementation it is possible to generate such graphs for different window sizes on the fly which gives auditors the chance to identify promising log regions for change point detection in an interactive and intuitive manner.

Besides high peaks – which clearly indicate a workflow change –, regions with an accumulation of smaller peaks are interesting, too. Not in every case a process change affects a large number of activity pairs at once, but rather progressively over a set of traces. For the same reason, peaks or accumulations occur slightly shifted with respect to the actual changing point, especially in case of minor process changes and when "old" traces are still executable within the new model.

Besides the peaks indicating structural process changes, there may be other smaller peaks. They occur if within the same model a set of paths is preferred

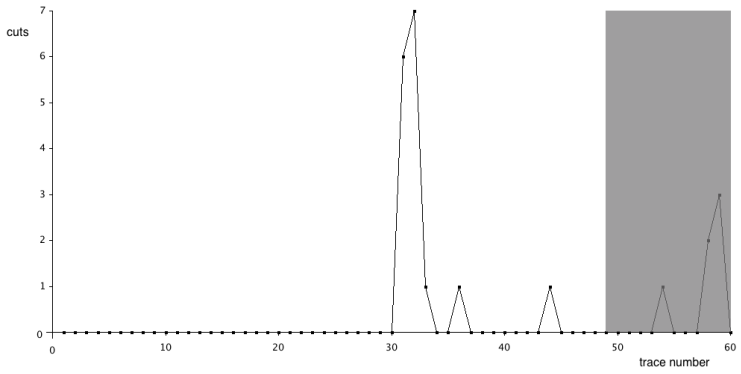


Fig. 6. Combined cluster cuts for an example log of the nets in Fig. 2.

for a while and can be interpreted as "intra-model changes". Depending on the chosen window size and structure of considered workflow models the number of peaks strongly varies. The smaller the window size, the more changes are detected. Considering a smaller number of traces limits the typical behavior which causes the probability of observing unseen/unknown behavior to grow. An interesting characteristic of cluster cut graphs is, that in most cases the peaks for "real" changes seem to be more stable with varying window sizes.

Fig. 6 shows the resulting graph after applying Algorithm 2 to a log containing 30 randomly generated traces for the first net followed by 30 traces of the second net in Fig. 2. Using a window size of 11, the cluster cut graph shows some high peaks around trace 30 which means that a high number of activity pairs experienced "change" in this region. This is in fact the region where the underlying model for the generated log traces changed. From the point on where the lookahead contains less traces than the window size, more changes are detected because the persistency of changes is based on a smaller number of traces than it is the case before. Simply combining the last w traces to one cluster or unifying them with the last detected cluster is a rather strict approach. We decided keep the cut information and mark the "critical region" (gray area within the graph) to give auditors the chance to decide on its usefulness. Alg. 2 provides the method to determine cluster cuts for particular activity pairs.

4 Testing the Clustering Approach

The clustering method is tested using the non-trivial process shown in Fig. 7. It contains 15 different activities that are composed using parallel and conditional branches. Its size and structural complexity is not exceeded by the vast majority of industrial processes. In order to show the detection rate of workflow changes, a set of process variants was generated, based on the change patterns by Weber et

al.[17]. Using process simulation techniques, test logs containing traces for each variant are generated, serving as input for the change point detection method.

Algorithm 2 Determine cluster cuts for specific activity pair

function GETCUTS(log L, distance matrix D, window size w, activity pair (a', a''))
  $window, lookahead, C \leftarrow \emptyset$
  for $index = 0, |L| - 1$ **do**
    if $|window| < w$ **then**
      $window \leftarrow window \cup \sigma_{index}$
      $lookahead \leftarrow lookahead \cup \sigma_{index+w}$
    else
      $actualValue \leftarrow D_{a',a''}[index]$
      if $actualValue \neq$ **null** but it always is in $window$ **then**
        NEWCLUSTER(index-1)
      else if $actualValue =$ **null** but there are non-null values in $window$ **then**
        NEWCLUSTER(index-1)
      else
        $violated \leftarrow actualValue < min_{a',a''}(interval) \vee actualValue > max_{a',a''}(interval)$
        $conform \leftarrow actualValue \geq min_{a',a''}(interval) \wedge actualValue \leq max_{a',a''}(interval)$
        $parallelPair \leftarrow obs(D_{a'',a'}|window) = 0$
        if $violated \wedge supp_{a',a''}(lookahead, actualValue) \geq \tau \wedge \neg parallelPair$ **then**
          NEWCLUSTER(index-1)
        else if $conform$ **then**
          $higherMin \leftarrow min_{a',a''}(lookahead) > min_{a',a''}(interval) \wedge max_{a',a''}(lookahead) \leq max_{a',a''}(interval)$
          $lowerMax \leftarrow max_{a',a''}(lookahead) < max_{a',a''}(interval) \wedge min_{a',a''}(lookahead) \geq min_{a',a''}(interval)$
          if $(higherMin \vee lowerMax) \wedge \neg parallelPair$ **then**
            NEWCLUSTER(index-1)
          end if
        end if
      end if
    end if
  end for
  return C
end function
procedure NEWCLUSTER(endIndex)
  $C = C \cup endIndex$
  $interval, lookahead \leftarrow \emptyset$
end procedure

Instead of focusing on simple operations (add/remove activities), we consider more complex operations to allow for a more intuitive understanding and interpretation of the change types that can be successfully detected. See [17]

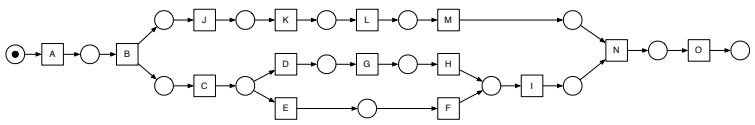
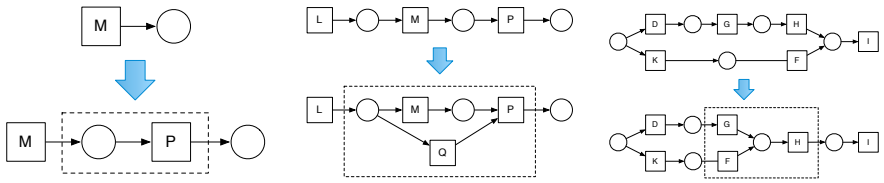


Fig. 7. Process for generating process variants and trace clustering.



(a) $M_0 \xrightarrow{(C1)} M_1$ (b) $M_1 \xrightarrow{(C2)} M_2$ (c) $M_4 \xrightarrow{(C5)} M_5$

Fig. 8. Examples of workflow changes by applying change patterns.

for an in-depth description of the considered change patterns. In this paper, the following set of change patterns to derive process variants is considered:

- **(C1) Serial Insert**: Insert a process fragment between two directly succeeding workflow activities.
- **(C2) Conditional Insert**: Insert a process fragment between two activity sets with an additional condition.
- **(C3) Parallel Insert**: Insert a process fragment between two activity sets with an additional condition.
- **(C4) Delete**: Remove a process fragment.
- **(C5) Move**: Shift a process fragment to another position.
- **(C6) Swap**: Swap a process fragment with another one.
- **(C7) Replace**: Replace a process fragment with another one.
- **(C8) Extract Sub Process**: Extract an existing process fragment and replace it by a "reference activity".
- **(C9) Parallelize**: Parallelize previously sequential process fragments.
- **(C10) Inline Sub Process**: Replace a "reference activity" by the referenced process fragment.
- **(C11) Embed Fragment in Conditional Branch**: Bind the execution of an existing process fragment to a condition.
- **(C12) Embed Process Fragment in Loop**: Add a loop construct surrounding an existing process fragment.

Successively applying these patterns to the original process M_0 in Fig. 7 results in a set of 13 models $M_0, ..., M_{12}$. Each model M_i was created by applying pattern **(Ci)** to the previous model M_{i-1}. Due to space limitations the process variants are not explicitly depicted in this paper, but Fig. 8 exemplary shows how some of the variants are formed.

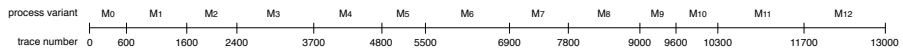


Fig. 9. Log with different trace numbers for each process variant.

All process variants have been modeled with PNML and simulated with a proprietary simulation module of SWAT [4]. In total, we create a process log with 13.000 traces. The flog in Fig. 9 contains different numbers of traces for different process variants successively.

To evaluate the effectiveness of our clustering method we first check if it successfully detects the change points between the different process variants. Fig. 10 shows the resulting cluster cut graph for the log in Fig. 9 in the clustering-application prototype. With the chosen window sizes, most of the change points are clearly identifiable by considering the highest peaks. It should be mentioned that using other window sizes does not significantly change the shape of the resulting graphs but simply results in a higher or lower number of additional peaks around change points.

However there are peaks which are not caused by model changes and/or cannot be directly obtained by graph analysis because they do not produce noticeable peaks, e.g. the model M_{10} and M_{11}. Taking a deeper look at this specific change reveals that the only difference between the two models is that an execution condition is added to a mandatory process fragment of M_{10} which additionally allows traces where the fragment does not occur. In contrast to the removal of a process fragment which is obtainable by the absence of distance observations for affected activity pairs in following traces, in this case the detection fails because the fragment may or may not occur and so there are no activity pairs whose distance is persistently altered by the change.

Another change that is not clearly identifiable is from M_4 to M_5 (see Fig. 8(c)) by moving the last activity (H) out of a conditional branch the corresponding join. Since changes between parallel activity pairs are omitted by the clustering method and only 2 sequential activities are affected by this change (namely E and F) the low peak at trace 4.800 is hardly surprising. The more locally constrained changes are and the less activities are affected, the harder their detection. This also applies to the change from M_5 to M_6 which is caused by swapping the nearby activities N and P. Although in this case the change is still identifiable within the graph in Fig. 10. Still, the cluster cut graphs provide valuable information for the identification of process changes which shows the appropriateness of monitoring activity pair distances for this purpose.

Change Localization. Besides the detection of change points, another goal of time-based trace clustering is *change localization* [6]. For this, the regions of change in the process models are of interest. In our approach, the region of change is obtained by considering the corresponding activity pairs for the peaks in the cluster cut graph. Zooming in on trace 600 in Fig. 10, e.g., reveals two high peaks, one in trace 601 with 18 related activity pairs and another in trace

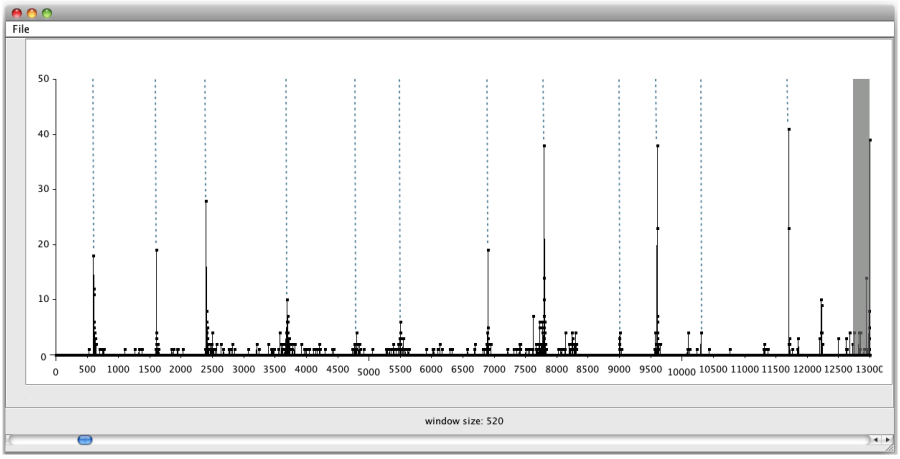


Fig. 10. Resulting cluster cut graph for the log in Fig. 9.

group 1	group 2	group 3
(A,N) (A,O) (A,P)	(P,N) (P,O)	(C,P)
(B,N) (B,O) (B,P)		(D,P)
(E,N) (E,O) (E,P)		(G,P)
(F,N) (F,O) (F,P)		(H,P)
(I,N) (I,O) (I,P)		(L,P)
(J,N) (J,O) (J,P)		(M,P)
(K,N) (K,O) (K,P)		

Table 1. Activity pairs for the peaks around trace 600 in Fig. 10.

603 with 11 related pairs. The resultant set of activity pairs for these two peaks is depicted in Table 1 and shows that most activity pairs refer to the activities $\{P, N, O\}$. Considering the change from Model M_0 to model M_1 in Fig. 8(a), this set contains the newly inserted activity P being the most referenced activity (it is contained in 15 out of 29 activity pairs), plus its successors.

5 Discussion

While our experiments show that time-based clustering is feasible for different types of changes, some general observations wrt the window-based apply.

There are various parameters that influence change point detection; the window size seems to be the most important one. Depending on this parameter, changes may become undiscoverable. This can either happen when the window size is too small which results in a higher number of peaks and "real" changes become indistinguishable from momentary changes (*diffusion*), or when changes are *masked* due to the inability of detecting clusters whose sizes are smaller than

the window size itself (this is an inherent disadvantage of window-basing). Instead of relying on an optimal window size, our approach solves this problem by allowing users to identify changes by analyzing the variance of cluster cut graphs with respect to different window sizes. To the best of our knowledge, this is the first approach allowing such kind of interactive change point detection.

Besides the window size, the variance of cluster sizes also plays a role. If the number of traces relating to different process variants significantly differ, there is no window size that can uncover all change points without masking or diffusing others. In case of similar trace numbers, the precision of change point detection is proportional to the average number of traces relating to the process variants. The more traces can be taken into account on determining the typical process behavior, the more precisely the process characteristics can be captured.

Another insight is that the difficulty to to detect changes strongly depends on the number of affected activities. For most patterns, structural characteristics of the original and the resulting process have influence on the ability to detect change points. Due to the observations we did so far, the number of activities that are affected by a change and the structural complexity of the underlying process (degree of parallelity) seem to exert the highest influence. However there are change patterns, that are generally hard to detect independent of these structural characteristics such as **(C11)**.

6 Related Work

Trace clustering approaches aim to group process traces according to some criteria. One of the first attempts in this area is multi-phase process mining [15,16]. Initially generating a process model for each individual process instance/case, these instance graphs are stepwise aggregated based on their similarity to introduce more general models. Greco et al. [8] propose a clustering technique in form of a stepwise refinement of a set of workflow schemas modeling specific usage scenarios. Starting with a single workflow schema describing the behavior of the overall process, in each refinement step the model with the lowest "soundness value" is selected for being refined. While "completeness" measures the covering rate of a model with respect to a process log, "soundness" is able to decide on over-generalization by measuring the percentage of enactments of the mined model that find some correspondence in the log [8].

Instead of iteratively checking the appropriateness of mined models to decide on further clustering, some approaches operate at trace level and focus on trace-similarity. Utilizing *k-means* for clustering, Song et al. [12] use features to characterize sets of traces. Trace-similarity is measured by distances of feature-vectors (i.e. the number of activities of a trace). Bose et al. [6] map the problem of multiple sequence alignment in bioinformatics on the trace-clustering problem to identify groups of similar traces. In [10] Lakshmanan et al. utilize spectral graph analysis to compute the difference of cluster-graphs to identify changes. Obtained models reflect different process characteristics but are loosely coupled with each other, hence failing to show the change of process models along time.

Recent mining approaches consider change in workflows [9]. In [10] Lakshmanan et al. utilize spectral graph analysis to compute the difference of clustergraphs to identify changes. However by using fixed cluster sizes, this approach is rather imprecise. Workflow dynamics in the sense of context changes can be characterized as *concept drift*, which is a well-studied paradigm in the data mining area. Bose et al. [5] provide methods to handle concept drift in process mining by showing that workflow changes are indirectly reflected in workflow logs and change point detection is feasible by examining activity relations.

In contrast to [5], our approach does not consider ordering relations (follows/precedes) but variations on the distance of activity pairs within a trace (the number of intermediate activities) as structural property. Sequentially processing traces according to time, traces whose structure differs from the typical trace structure so far are treated as indicators for new clusters. Proceeding this way ensures that a log file is partitioned into chronologically subsequent clusters, whose sizes depend on the frequency of changes in process structure. However, we do not identify the kind of change, but focus on the identification of time points of permanent process changes.

7 Summary and Further Work

This paper presents a trace clustering approach to support process discovery of configurable, evolving process models. The clustering approach allows users to distinguish between different process variants within a timeframe, thereby visualizing the process evolution. The main insight to cluster entries is the "distance" between activities, i.e. the number of steps between an activity pair. The evaluation shows that the approach is promising. Still, there are several issues that remain to be addressed. The following presents the most important issues:

1. **Loop support.** The clustering method at this point does not support workflow models containing loops which definitely is its biggest disadvantage as it limits its applicability. We are confident that an enhancement in this direction is straightforward and can be done without negatively influencing the effectiveness of the method.
2. **Identification of change patterns.** Although our experiments convey an impression of the identifiability of change patterns, a more systematic approach is needed to provide a well-founded basis for their effective detection.
3. **Feature set.** So far our approach solely relies on the activity pair distance metric, but incorporating additional metrics may enable more precise change point detection. For change patterns like **(C11)** where the identification of change points is not straightforward, additionally considering the length of log traces of subsequent samples may be useful.
4. **Change localization.** As mentioned before, the proposed clustering method provides sufficient information for the localization of change within workflow models. Here, automatic mechanisms for the extraction of change regions from a set of affected activity pairs are envisioned.

References

1. R. Accorsi and T. Stocker. On the exploitation of process mining for security audits: The conformance checking case. In *ACM Symposium on Applied Computing*. ACM, 2012 (to appear).
2. R. Accorsi and C. Wonnemann. Auditing workflow executions against dataflow policies. In *Proceedings of the Business Information Systems*, vol. 47 of *LNBIP*, pp. 207–217. Springer, 2010.
3. R. Accorsi and C. Wonnemann. Strong non-leak guarantees for workflow models. In *ACM Symposium on Applied Computing*, pp. 308–314. ACM, 2011.
4. R. Accorsi, C. Wonnemann, and S. Dochow. SWAT: A security workflow toolkit for reliably secure process-aware information systems. In *Conference on Availability, Reliability and Security*, pp. 692–697. IEEE, 2011.
5. R. Bose, W. v. d. Aalst, I. Zilobaite and M. Pechenizkiy. Handling concept drift in process mining. In *Conference on Advanced Information Systems Engineering (CAiSE 2011)*, 2011.
6. R. Bose and W. v. d. Aalst. Trace alignment in process mining: Opportunities for process diagnostics. In *Conference on Business Process Management*, vol. 6336 of *Lecture Notes in Computer Science*, pp. 227–242. Springer, 2010.
7. A. de Medeiros, A. Guzzo, G. Greco, W. v. d. Aalst, A. Weijters, B. v. Dongen and D. Saccà. Process mining based on clustering: A quest for precision. In *Business Process Management Workshops*, vol. 4928 of *Lecture Notes in Computer Science*, pp. 17–29. Springer, 2008.
8. G. Greco, A. Guzzo, L. Pontieri and D. Saccà. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, vol. 18(8), pp. 1010–1027, 2006.
9. C. Günther, S. Rinderle-Ma, M. Reichert, W. v. d. Aalst and J. Recker. Using process mining to learn from process changes in evolutionary systems. *Int. J. Business Process Integration and Management*, vol. 1, pp. 111–111, 2007.
10. G. Lakshmanan, P. Keyser and S. Duan. Detecting changes in a semi-structured business process through spectral graph analysis. In *Workshops of the Conference on Data Engineering*, pp. 255–260. IEEE, 2011.
11. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, vol. 77(4), pp. 541–580, April 1989.
12. M. Song, C. Günther and W. van d. Aalst. Trace clustering in process mining. In *Business Process Management Workshops*, vol. 17 of *Lecture Notes in Business Information Processing*, pp. 109–120. Springer, 2008.
13. W. v. d. Aalst. *Process Mining – Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
14. B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters and W. v. d. Aalst. The ProM framework: A new era in process mining tool support. In *Conference on Applications and Theory of Petri Nets*, vol. 3536 of *Lecture Notes in Computer Science*, pp. 444–454. Springer, 2005.
15. B. F. van Dongen and W. van der Aalst. Multi-phase process mining: Building instance graphs. In *Conceptual Modeling*, vol. 3288 of *Lecture Notes in Computer Science*, pp. 362–376. Springer, 2004.
16. B. v. Dongen and W. v. d. Aalst. Multi-phase process mining: Aggregating instance graphs into EPCs and Petri nets. In *PNCWB 2005 Workshop*, pp. 35–58, 2005.
17. B. Weber, S. Rinderle and M. Reichert. Identifying and evaluating change patterns and change support features in process-aware information systems. Technical Report. University of Twente, Enschede, The Netherlands, 2007.