Activity Mining by Global Trace Segmentation

Christian W. Günther, Anne Rozinat, and Wil M.P. van der Aalst

Information Systems Group, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{c.w.gunther,a.rozinat,w.m.p.v.d.aalst}@tue.nl

Abstract. Process Mining is a technology for extracting non-trivial and useful information from execution logs. For example, there are many process mining techniques to automatically discover a process model describing the causal dependencies between activities. Unfortunately, the quality of a discovered process model strongly depends on the quality and suitability of the input data. For example, the logs of many real-life systems do not refer to the activities an analyst would have in mind, but are on a much more detailed level of abstraction. Trace segmentation attempts to group low-level events into clusters, which represent the execution of a higher-level activity in the (available or imagined) process meta-model. As a result, the simplified log can be used to discover better process models. This paper presents a new activity mining approach based on global trace segmentation. We also present an implementation of the approach, and we validate it using a real-life event log from ASML's test process.

Keywords: Process Mining, Event Log Schema Transformation, Trace Segmentation.

1 Introduction

Process mining technology attempts to extract non-trivial and useful information about real-world processes from their "footprints", i.e., from event logs recorded by IT systems that support these processes [1]. However, the value of a process mining analysis strongly depends on the quality and suitability of the input event log data. First and foremost, many process mining approaches require that the event log is cleaned to remove any kind of noise. Further, it is necessary that the structure of the event log is in line with the process meta-model that is used for interpretation of the results. The latter requirement, i.e. a proper alignment between the event log structure and an understood process meta-model, is often not fulfilled by event logs extracted from real-life systems. There may be misalignments due to a number of reasons:

– The log recording mechanism is faulty, or is not properly configured.
– The configuration of the log recording mechanism does not correspond to the understood process meta-model.
– The level of abstraction on which logging occurs does not correspond to the desired level of abstraction for log analysis.

These are only three examples for a number of situations which may lead to misalignment. The first example points out an actual problem, which may not be possible to resolve. However, the other examples point to misalignment that can be fixed by a set of techniques for *event log schema transformation*.

Event log schema transformation modifies the level of abstraction of event logs. It may be applied to any layer of the log (e.g., to the whole log, traces, activities, etc.). Put simply, event log schema transformation works by clustering and splitting artifacts within one layer, e.g., clustering events within traces, or partitioning the log into several "sublogs". This transformation effectively propels the respective layer onto a different level of abstraction. Therefore, by using event log schema transformation, an event log can be adjusted to correspond to a specific process meta-model, so that its analysis becomes more meaningful. Note that these transformation techniques can also be used in the absence of an explicit process meta-model. In such use cases it is helpful for easing log analysis, e.g. by reducing the number of artifacts within an event log.

In this paper we focus on *trace segmentation*, which denotes the identification of coherent subsequences of events within traces. Trace segmentation can be used both for identifying activity clusters in low-level logs, as well as for finding more homogeneous, tacit sub-traces within a longer, unstructured trace. The remainder of the paper is organized as follows. We first discuss the topic of trace segmentation (Section 2). Then, we introduce a new approach based on global trace segmentation, which can be used for activity mining (Section 3). Subsequently, the implementation of this new activity mining approach is presented (Section 4). We evaluate the approach by applying it to a complex real-life event log from ASML (Section 5). Finally, we discuss related work (Section 6) and conclude the paper (Section 7).

2 Trace Segmentation

An ideal event log for process mining analysis is well-structured and on an appropriate level of abstraction. In many real-life situations, these requirements are, however, not fulfilled. Often, real event logs are recorded on a very low level of abstraction. Events in these logs are identifying miniscule activities within the system, which cannot be easily related to activities in the process model imagined by the analyst. It is not that these high-level activities are not represented in the event log at all, rather that their representation is scattered among many low-level events. This dissociation of activities makes it very hard for process analysts to correctly relate the observed behavior to any available, or imagined, process meta-model.

Trace segmentation is an important event log schema transformation technique, which makes such low-level logs more understandable and easier to analyze. The fundamental idea of trace segmentation is illustrated in Figure 1. The starting point is a low-level trace of events (shown on the bottom of Figure 1). Trace segmentation attempts to identify coherent *sub-sequences* of events within the trace, i.e., to "cut up" the trace into a number of event clusters. In the example in Figure 1, four clusters have been identified in the trace.

The rationale behind trace segmentation is that every cluster of low-level events is supposed to represent the execution of a higher-level activity in the (available or imagined) process meta-model. It is also important that these event clusters are properly categorized (i.e., clustered) into *types* of clusters. This allows for the discovery of cor-

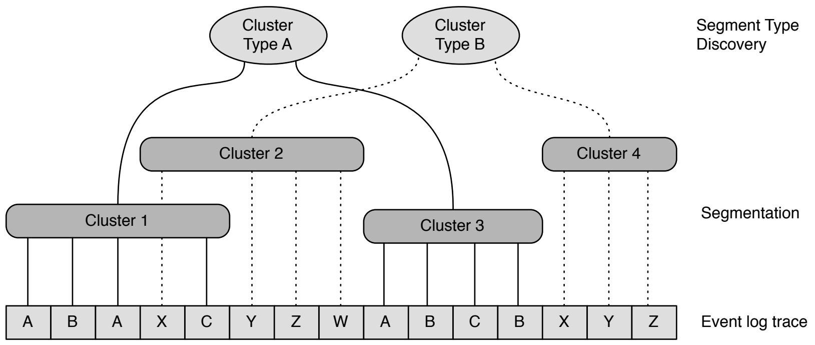


Fig. 1. Schematic description of trace segmentation.

responding activity types, which are supposed to result in a comparable sub-sequence of low-level events. With respect to the example in Figure 1, two cluster types A and B have been identified, each supported by two clusters. Clusters of type A consist of events from event classses A, B, or C, while clusters of type B are constituted by events of classes X, Y, Z, and W.

Trace segmentation has two main use cases. The first one is *activity mining*. In activity mining, trace segmentation is applied to elevate the log's level of abstraction, i.e., to analyze the event log from a higher-level point of view. In the example in Figure 1, the trace would have been simplified to a sequence of four events (i.e., the clusters), from two event classes (i.e., the cluster types). The second use case is *trace discovery*. In trace discovery, the discovered event sub-sequences are interpreted as hidden traces of a process described by their cluster type. Regarding the example, the process type represented by cluster type A would have two traces A, B, A, C and A, B, C, B.

Both activity mining and trace discovery can be implemented by trace segmentation techniques. In the following section we propose a new global approach towards trace segmentation, which is based on the correlation between event classes.

3 Global Trace Segmentation Approach

As explained earlier, trace segmentation is based on the idea that subsequences of events, which are supposed to be the product of a higher-level activity, are identified. This approach focuses on the global correlation between *event classes*, i.e. types of events. From the co-occurrence of events in the log, we derive the relative correlation between their event classes.

Our approach for global trace segmentation can be outlined as follows.

- We use the notion of a *global event class correlation*, describing how closely related the event classes found in the log are. Event class correlation is derived from the log, i.e., event classes whose events frequently occur together in the log are highly correlated.

– Based on event class correlation, we infer a *hierarchy of event classes*. This hierarchy represents an abstraction of the set of original event classes on multiple levels. All event classes found in the log are successively combined into clusters, representing higher-level types. Note that the clusters created in this step do not refer to groups of events, but are clusters of event classes.
– In this hierarchy of event classes, an arbitrary level of abstraction can be chosen for trace segmentation. The clusters of event classes on that level of abstraction are then *projected* onto the event log. Subsequences of events, whose event classes belong to one cluster, are considered *segments* of the log. These segments, i.e. clusters of events, are the result of global trace segmentation.

The global approach described here works *top-down*. The association of each event to its respective higher-level cluster is not directly derived from its local, surrounding events, but rather established from the global correlation of its event class to other event classes.

In the remainder of this section, we first show how the global event class correlation is determined, which is the foundation of this approach (Section 3.1). Then, we describe how, based on this correlation, our algorithm builds a global event class hierarchy (Section 3.2). Finally, we explain how this hierarchy is used to enable the actual global trace segmentation in an adaptive manner (Section 3.3).

3.1 Scanning Global Event Class Correlation

Our global approach for trace segmentation is based on the notion that there is a global relationship between event classes in the log. This relationship between event classes is then projected onto the actual event instances in the log, i.e. events inherit their mutual relationship from their respective event classes.

We can express this relationship between event classes in a correlation function.

Definition 1 (Event class correlation). *Let C be a set of event classes. The function $ecc \in C \times C \rightarrow \mathbb{R}_0^+$ assigns to each tuple of event classes a certain* correlation *value. The larger this value is, the more related the two respective event classes are.*

In our approach we determine the correlation function between event classes by scanning the complete log. We start with a matrix of $C \times C$, initialized with zero values before the actual scanning pass. While traversing the log, this matrix is updated for each following relation that is found. Note that this correlation matrix, as well as the correlation function itself, is symmetric, i.e. $ecc(X, Y) = ecc(Y, X)$. During the scanning pass, this symmetry needs to be maintained by the algorithm.

The scanning pass of this algorithm is illustrated in Figure 2. In this example, the scanning is currently inspecting an event of class A, on the bottom of the trace. We call the event currently under consideration the *reference event*. Looking at the directly preceding event of class H, the scanner can establish an observation of the co-occurrence between event classes H and A, which means that their relationship is strengthened. Correspondingly, the correlation matrix value for $ecc(A, H)$ is incremented by i, the increment value (which is usually set to 1).

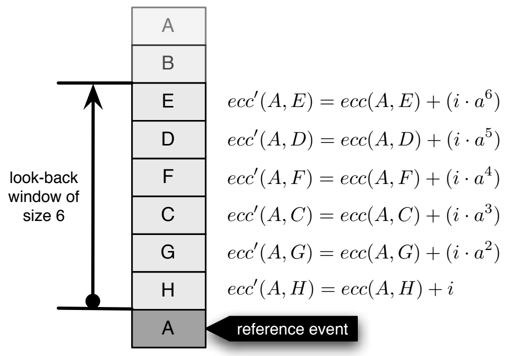

A
B
E
D
F
C
G
H
A
look-back window of size 6
reference event
$ecc'(A,E) = ecc(A,E) + (i \cdot a^6)$
$ecc'(A,D) = ecc(A,D) + (i \cdot a^5)$
$ecc'(A,F) = ecc(A,F) + (i \cdot a^4)$
$ecc'(A,C) = ecc(A,C) + (i \cdot a^3)$
$ecc'(A,G) = ecc(A,G) + (i \cdot a^2)$
$ecc'(A,H) = ecc(A,H) + i$

Fig. 2. Scanning global event class correlation.

In our approach, the scanning pass uses a *look-back window* for evaluating each event. This means that if the look-back window's size is six (as in the example in Figure 2) the scanner will consider the last six events which have preceded each reference event. Note that this algorithm for scanning the event log is similar to the one used in [10]. When evaluating events in the look-back window, the scanner will attenuate its measurement exponentially, based on an attenuation factor a, where $0 < a < 1$. For any event x in the look-back window, where y is the reference event, the correlation matrix will be updated as follows: $ecc(c(x), c(y)) = ecc(c(x), c(y) + (i \cdot a^n)$, where n is the number of events located between x and y in the trace. This approach improves the capture of relationships within activity clusters where behavior is more unstructured, i.e., where the order of low-level events may change frequently.

After the scanning pass has evaluated all events in all traces of the log, a reliable correlation function between event classes is established, as expressed in the aggregated correlation matrix. Our correlation function thus considers two event classes as more related, if events of these classes frequently occur closely together in traces of the log.

3.2 Building the Event Class Cluster Hierarchy

After the correlation function between event classes has been established in the scanning pass, we have a global measurement of relationship between event classes in the log. Based on this correlation function, our approach builds a hierarchy of event classes, successively combining the most related event classes into higher-level entities.

For this task, we use an adapted version of the *A*gglomerative *H*ierarchical *C*lustering (AHC) algorithm, which is a well-known data clustering technique [3]. The *primitives* to be clustered are the event classes found in the log, and we apply AHC using the correlation function established in the scanning pass. Note that the clusters created here refer to higher-level activity types, i.e., we are actually inferring *types of clusters* whose instances are event clusters in the log.

The AHC algorithm can be described as follows:

1. The set of entities E initially consists of all primitives, i.e. event classes.

2. Find the two entities a and b in E which have the *largest correlation* value $ecc(a, b)$ for all tuples of entities in E.
3. Create a new event class cluster x, which contains a and b.
4. Remove a and b from E, and add x to E.
5. If E contains more than one entity, continue with the next iteration at step 2.

Thus, in each iteration the AHC algorithm merges two entities, and thus successively combines all event classes into one cluster representing all event classes. To be able to merge also clusters with event class primitives, or with other clusters of event classes, we need to specify an appropriate correlation function for clusters. For this task we use the notion of *complete linkage* [3], which defines the distance of two clusters as the maximum distance between any elements from each cluster. Note that our notion of correlation is inversely related to the notion of distance used in data clustering. Therefore, the correlation between a cluster and an event class is also defined as the minimal correlation of any element of the cluster to that respective event class.

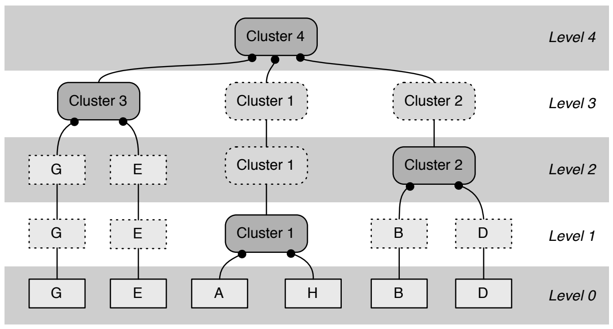


Fig. 3. AHC event class cluster tree.

By applying this AHC algorithm to the set of event classes in a log, we construct a hierarchical tree structure of event classes and event class clusters. Figure 3 shows an example of this structure. The initial set of event classes is depicted at the bottom, consisting of event classes G, E, A, H, B, and D.

During the first iteration of the AHC algorithm, event classes A and H are determined to have the maximum correlation value of all event classes. Consequently, they are combined into a new event class cluster, which is shown as $Cluster1$ in Figure 3. Every iteration of the algorithm creates a new *level of clustering*. For example, the first iteration creates level 1, which differs from the initial level 0 by having event classes A and H removed, and replaced by the newly created $Cluster1$.

3.3 Adaptive Global Trace Segmentation

Once the event class cluster hierarchy has been established, we can apply our global trace segmentation approach. Since this hierarchical structure successively simplifies the set of event classes in each level, we can take advantage of this and allow the analyst to adaptively simplify the event log.

After selecting the desired level of abstraction, corresponding to the different levels in the event class cluster hierarchy, every event in the log is processed as follows. Events whose event classes are still present as primitives in the desired level of abstraction are left untouched. If an event's class is contained in a cluster on the desired level of abstraction, its event name is replaced by the cluster name. After rewriting the log in this manner, repetitions of events which refer to the same cluster are collapsed into one event.

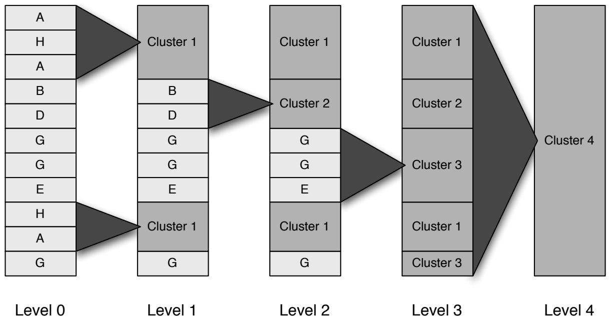


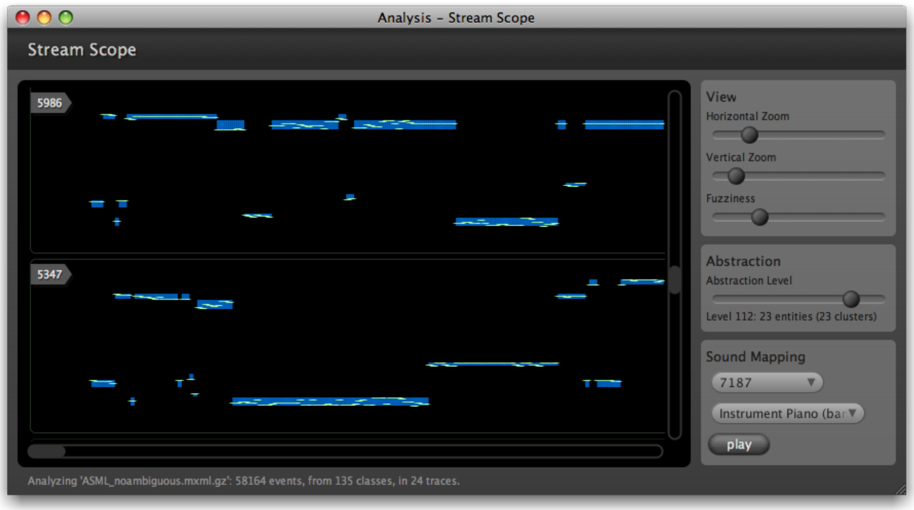
Fig. 4. Step-wise global trace segmentation.

This procedure is illustrated in Figure 4. An original trace of the log is shown on the left, i.e. this trace is on level 0 and thus not simplified yet. For every level of abstraction in the event class cluster hierarchy we can now apply our trace segmentation approach. In our example from Figure 3, event classes A and H were combined to $Cluster1$ in level 1. The example trace in Figure 4 has three events of class A and two events of class H. After rewriting these events, they can be combined into two occurrences of $Cluster1$, thus simplifying the log. As shown in Figure 4, this procedure can be applied for every level of the event class cluster hierarchy. It successively reduces the number of event classes, and the number of overall events, in the log.

While this example shows the application of global trace segmentation for activity mining, it can be equally applied for trace discovery. With respect to level 1 in Figure 4, the algorithm has discovered two traces A, H, A and H, A, both describing the tacit subprocess denoted as $Cluster1$.

4 Implementation and Visualization

The global approach for trace segmentation has been implemented as the *Stream Scope* plugin in the *Pro*cess *M*ining (ProM) framework [1]. While the focus of this implementation is on activity mining, extending it for trace discovery purposes would be rather trivial.



Level 112: 23 entities (23 clusters)

7187

Instrument Piano (ba

Analyzing 'ASML_noambiguous.mxml.gz': 58164 events, from 135 classes, in 24 traces.

Fig. 5. Stream Scope visualization of global trace segmentation.

When the plugin is applied to an event log, the correlation function is scanned from the log and the event class cluster hierarchy is created. This initialization of the plugin is very fast. After the initialization, the result view of the plugin is shown, as depicted in Figure 5. The left side of this view is devoted to the actual stream scope log visualization, while the right side offers parameters to adjust both the visualization and the trace segmentation.

The stream scope visualization is based on the event class cluster hierarchy, and shows every trace in the log separately. Every event class found in the log corresponds to a vertical coordinate, while the horizontal coordinate represents the sequence of events as they occur in the trace (Note that the stream scope visualization uses an equidistant display of events, i.e. actual time is not considered for horizontal coordinates). Every event in a trace is represented by a green dot in the visualization.

For ordering event classes on the vertical axis of the visualization, the stream scope plugin uses the event class cluster hierarchy. This hierarchy tree is walked in a *depth-first* manner, thereby assembling a list of leave nodes (i.e., event classes) in the order in which they are found. As a result, event classes are ordered in a manner which reflects their correlation, i.e. more related event classes are situated closer together in this order. The hierarchy shown in Figure 3 shows an example of this in Level 0, which essentially gives an example of event classes ordered in this manner, from left to right. The resulting view can be compared to a musical notation. Patterns of co-occurring events can be easily recognized by their locality, i.e. their being situated closely together.

On the right side of this view, the user can set the desired level of abstraction with a slider. This will result in a display of blue background areas in the visualization, covering the span of event classes combined in each cluster found. Note that this display of coherent cluster blocks is only possible due to the reordering of event classes from the hierarchy tree, which ensures that clusters are non-interrupted vertical subsequences.

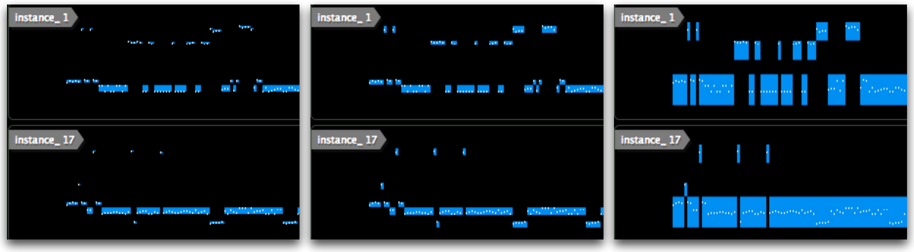


Fig. 6. Global trace segmentation on three levels of abstraction.

Figure 6 shows an excerpt of a streamscope visualization for two event log traces. This excerpt is shown on three different levels of abstraction. On the left, the log is projected onto 76 event classes. 34 of these event classes are in fact clusters of event classes. The clusters of events referring to these clustered event classes have a solid blue background in the visualization. In the center of Figure 6, the log has been projected onto 18 event classes, all of which are in fact clusters of event classes. One can see that especially events, whose classes are more located towards the top of the visualization, have been combined into larger clusters, when compared to the previous abstraction. Finally, the visualization on the right shows the log projected onto four event classes, all of which are clusters.

The plugin provides a projection of the log, which corresponds to the currently selected level of abstraction, to the framework. Thereby, this simplified log is also available for other mining and analysis techniques. Further, the plugin allows the user to control the visualization, both by adjusting its size (i.e., zooming in or out) and by applying a "blur" filter, which can help in identifying patterns of event classes visually. Finally, the plugin allows the user to "play back" traces in audio, by mapping each event class onto a musical note. This experimental feature harnesses the capabilities of human audio perception to intuitively recognize musical patterns.

5 Application to ASML's Wafer Scanner Test Log

In a previous case study [7, 8], we have analyzed ASML's test process for wafer scanners. ASML is a leading manufacturer of wafer scanners, complex and expensive machines which play an integral part in the production of integrated circuits. We have analyzed ASML's test process for wafer scanners based on event log data recorded during the wafer scanner test phase. The actions in this test phase can be considered on two levels of abstraction: While the actual event log data could only be obtained on a rather low level of abstraction, detailing the *single tests* which had been executed, ASML also provided us with an explicit mapping of low-level tests to so-called *job steps*. Using this mapping, a higher-level event log, describing the test process on a job step level, could be derived. This lower-level and higher-level log was eventually used for analysis by more traditional process mining approaches.

Even for the higher-level, job step-mapped log our analysis has yielded relatively unstructured and complex spaghetti models. These models are not only cumbersome to read and hard to interpret. When compared to the reference process model provided by ASML, they indicate serious deviations in practice, i.e., the test process is in fact not executed according to specification. These deviations are due to the complexity of the testing process (e.g., failing tests may trigger the re-execution of earlier phases in the test procedure to account for changed parameters or replaced components). They are thus, in principle, expected and inherent to the test process at ASML. However, at the same time the provided grouping of low-level tests into higher-level job steps, just like the reference process itself, are created manually. Since job steps represent the *idealized* process steps, they may not reflect the actual *reality* of testing wafer steppers in practice.

Ideally, a job step, as defined by the reference process, should refer to a self-contained part of the testing process, which can be considered completed if all its tests have been successfully executed. However, if these job steps do not represent an appropriate grouping of low-level tests, this can lead to unnecessary re-executions of earlier job steps, and thus can introduce deviations to the reference process. If tests could be better grouped into job steps, the high-level process model would more accurately reflect the true process of testing wafer scanners in ASML. Furthermore, if there are strong dependencies between tests that are contained in separate job steps, this information can be used to improve the process by duplicating or re-positioning tests, so as to reveal problems earlier on, and thus shorten the completion time of the overall test procedure (by avoiding re-executions of large parts of the test process).

For investigating the suitability of the current grouping of tests into job steps, we have analyzed the original ASML log on the test code level with the global trace segmentation approach presented in this paper. If the job step compositions defined by ASML are indeed appropriate, trace segmentation should be able to rediscover the original job steps, as clusters of test codes, from the event log.

In fact, Figure 5 shows part of the visualization of this technique applied to the ASML test log. In total, 23 clusters have been derived from the event log, so as to correspond to the 23 job steps defined by ASML. We can see that the clusters, as discovered by global trace segmentation and highlighted in blue in Figure 5, cover contiguous and extensive parts of traces. Thus, we can assume that the low-level behavior captured in these clusters represents actually related tests, which are frequently executed in close proximity. We subsequently compared these 23 clusters obtained by the global trace segmentation approach to the original 23 job steps. Consider Figure 7, which visualizes the relations of the obtained clusters (clusters *1–23*) to the original job steps (‘ae’–‘zero’). We can observe relationships between clusters and job steps in two directions.

1. One can see that many clusters contain (parts of) different job steps, thus indicating a strong relation between these job steps. For example, cluster *23* completely contains the job steps ‘d’, ‘i’, ‘l’, and ‘m’, whereas cluster *1* fully covers job steps ‘x’, ‘v’, and ‘u’. Another example are the job steps ‘ue’ and ‘oe’, which seem to be highly connected (cf. clusters *2*, *8*, and *7*).
2. Furthermore, many job steps are actually split-up and represented by multiple clusters. Consider, for example, job step ‘j’, which is in part associated to cluster *23* while another part of the same job step is associated to cluster *1*.

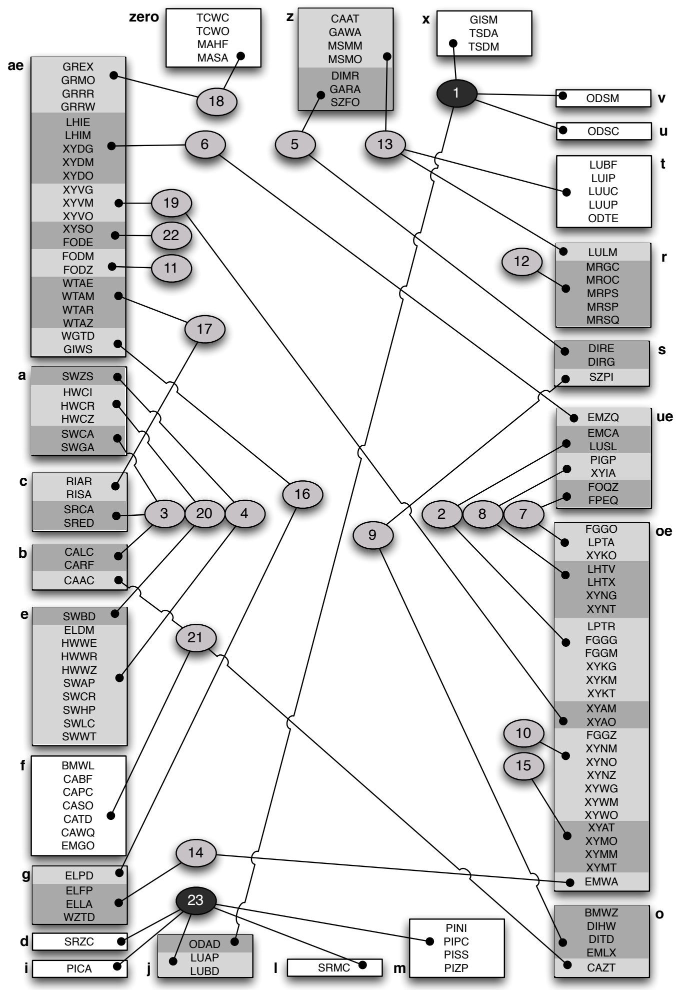
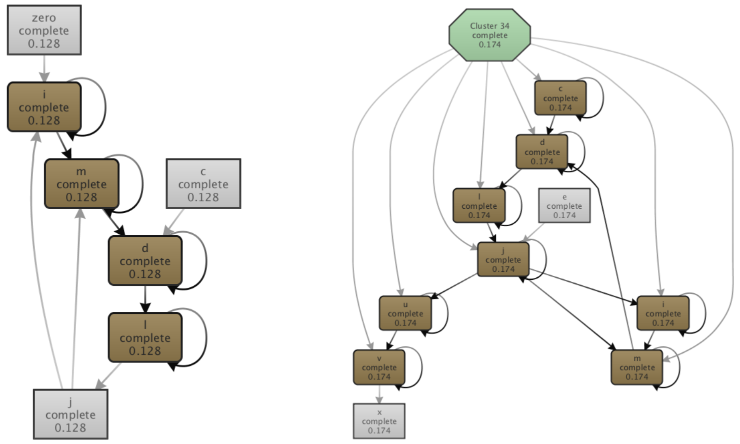


Fig. 7. Relationship between the clusters discovered by global trace segmentation (indicated by numbers), and the original job step composition (indicated by 'ae', 'zero', 'a', etc.). The background color of tests in the job steps indicate their association to the clusters discovered by global trace segmentation.

While relationship (1) can in part also be detected by analyzing the process model discovered from the job step-mapped event log, especially relationship (2) generates additional insight that could not be obtained via the more traditional process mining techniques. This can be illustrated by Figure 8, which depicts two clusters[1] that were

[1] Note that the Fuzzy miner is able to cluster highly correlated nodes in the process model. The graphs in Figure 8 show the nodes which have been clustered in a darker color, while the regular nodes in the model, to which these are connected, are visualized in a lighter color.



(a) Small cluster (reflects activity cluster *23*).

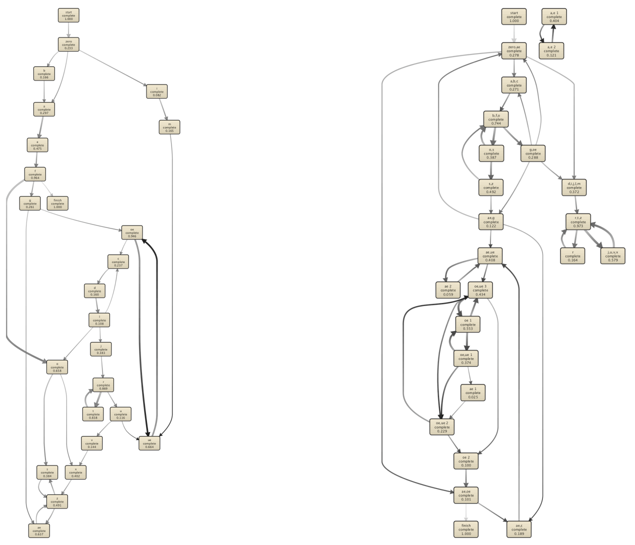
(b) Larger cluster (reflects activity cluster *1* and *23*, which share job step *j*).

Fig. 8. Two clusters in the job step-based fuzzy model that were grouped based on a smaller and a higher node significance cutoff. In the larger cluster two actually separate sets of job steps are grouped together (and not put in distinct clusters) because a subset of their test codes belongs to the same job step (see *j* in Figure 7).

created by the Fuzzy Miner [6] based on the original job step-mapped log. The left cluster aggregates the four job steps ‘d’, ‘i’, ‘l’, and ‘m’, and thus largely reflects cluster *23* from Figure 7. The right cluster was created by the fuzzy miner after increasing the node significance threshold parameter, thus yielding a model with a higher node aggregation tendency. However, instead of creating two separate cluster nodes, reflecting the correlation of job steps ‘d’, ‘i’, ‘l’, and ‘m’ on the one side (cluster *23* in Figure 8), and reflecting the correlation of job steps ‘x’, ‘v’, and ‘u’ on the other side (cluster *1* in Figure 8), all these job steps are aggregated within a single cluster node in the fuzzy model.

The reason for this is job step ‘j’, which is partly associated to one group, and partly associated to the other group of job steps. Due to the fixed mapping between test codes and job steps (test code names were simply replaced by their corresponding job step names), occurrences of tests belonging to this job step cannot be distinguished by the mining algorithm, and therefore all job steps are grouped together. Here it becomes apparent that, using the global trace segmentation approach, more detailed dependencies between the job steps can be revealed than by simply analyzing the process model. As a consequence, it is easier to detect concrete opportunities to optimize the test procedure, e.g., by repositioning or replicating tests within the reference process, which is followed by test engineers.

Since the clusters obtained by the global trace segmentation approach now offer an alternative way of grouping the low-level tests into higher-level process steps, we have used a log reflecting this new abstraction for process discovery. We have then compared



(a) Based on explicit job step mapping (conformance: 0.28).

(b) Based on previously mined activity clusters (conformance: 0.46).

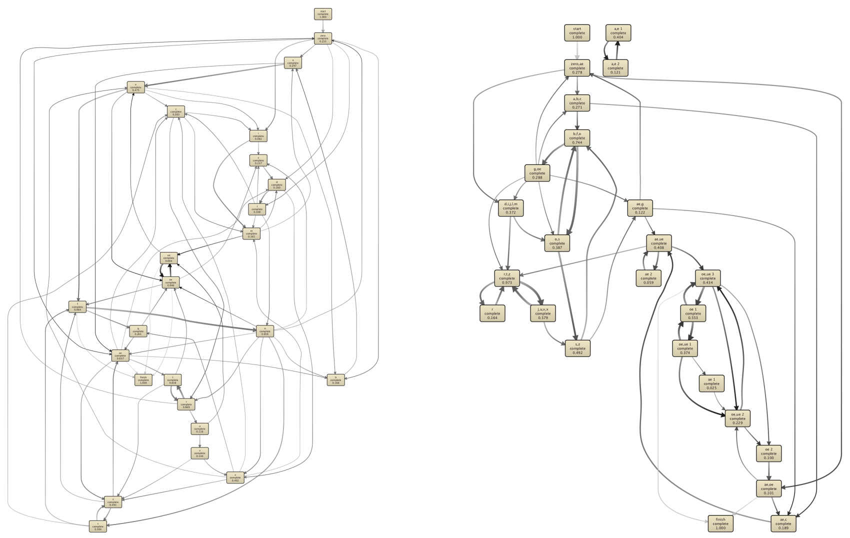
Fig. 9. Two fuzzy models mined with the same parameters, yielding a similar level of complexity. The activity cluster-based model has a significantly better conformance.

the results to the model obtained from the original job step-mapped event log. Figure 9 depicts two models that were discovered with the fuzzy miner while using exactly the same mining parameters. The model on the left was created using the explicit job step mapping provided by the domain experts, while the model on the right was created using the grouping obtained from trace segmentation (instead of cluster names, their job step associations are used in the activity name).

Both models show a comparable level of complexity. However, we can judge the quality of these models better when using the conformance metric defined for fuzzy models [6]. The model in Figure 9(a) has a significantly poorer fuzzy conformance (0.28) than the model depicted in Figure 9(b), which has a conformance of 0.46. This supports the hypothesis that our trace segmentation approach creates a more realistic grouping of the low-level tests than the manual job step mapping, and thus more faithful representations of the real test process flow can be discovered.

Similarly, Figure 10 depicts two process models created with the fuzzy miner, which have the same level of conformance (ca. 0.5). We have adjusted the parameters of the Fuzzy Mining algorithm [6] to yield two models with the same fuzzy conformance. The model on the left has been mined using the original job step mapping and is clearly more complex than the model on the right, which was discovered based on the previously discovered activity clusters.

In this section, we have used our new activity mining approach to evaluate the job step abstraction provided by ASML. We have found that a more suitable abstraction



(a) Based on explicit job step mapping (more complex model).

(b) Based on previously mined activity clusters (less complex model).

Fig. 10. Two fuzzy models with the same conformance (≈ 0.5). The activity cluster-based model is clearly less complex, since it captures the actual steps in the process in a better way.

can be proposed based on the actually observed behavior. This abstraction should be compared to the given job step mapping by a domain expert. Furthermore, 176 out of the total 360 test codes were not included in the given job step mapping at all. This is due to the fact that the job step mapping is manually created and maintained. To be able to compare our discovered clusters to the original job steps, we have removed these events from the log. However, using activity mining it may be possible to create an updated job step mapping for ASML, which also includes these currently unmapped test codes.

6 Related Work

Process mining is a field of analysis that focuses on mining behavioral aspects from log data. Since the mid-nineties several groups have been concentrating on the discovery of process models from event-based data. In [2] an overview is given of the early work in this domain. To also tackle logs from more more unstructured and complex processes, flexible approaches such as Fuzzy mining [6] have been proposed recently.

Regarding trace segmentation, we earlier introduced a local approach [5], which is based on clustering event *instances*, by analyzing their proximity towards other events in their trace. In contrast, the approach presented in this paper focuses on the global correlation between *event classes*, i.e. types of events. Compared to the global approach presented in this paper, the local approach may yield better results, since it allows for one event class to be contained in multiple cluster types. Given an event log containing the same low-level event class in a number of higher-level activities, the global approach presented here cannot discover an accurate set of event cluster types. However, while the local approach may provide more accurate results, it also has serious performance problems. Since for every event in the log a corresponding initial cluster is created, the memory consumption can be very high for large logs. Further, since all initial clusters need to be compared to one another, the runtime complexity is exponential with respect to the size of the log. In contrast, the global approach presented here is very efficient with linear complexity, and is thus suitable even for interactive use.

Related to our notion of trace segmentation are approaches from the data mining domain, which also focus on the clustering of sequential event data. The main difference is that process mining needs to deal with various forms of concurrency. In [5] a more detailed discussion of related work in this area is provided.

There are also other event log schema transformation approaches, which, for example, cluster traces within a log based on the assumption that the event log in fact describes a number of tacit process types [9, 4]. As a consequence, similar traces within one log are grouped into more homogeneous subsets, which can subsequently be analyzed separately and yield more understandable process models than if analyzed altogether.

7 Conclusion

In this paper we have presented a new approach for trace segmentation, which can identify subsequences of events that are supposed to be the product of a higher-level activity. In contrast to earlier approaches, our solution addresses the problem in a global, top-down manner. Based on a global correlation between event classes, these classes are successively clustered into higher-level classes. By projecting these clusters onto the actual events in a log we can reduce the number of unique activities, and thus elevate the event log onto an arbitrary level of abstraction.

We have demonstrated the usefulness of trace segmentation in general, and of our global approach in particular, using the case study of ASML's test process as an example. While ASML had provided us with an explicit model for their test process, the actual testing performed in practice differed significantly, as discovered by process mining. Process discovery can show that the *control flow* of actual process execution is different from an idealized process model. In this paper, we have shown that also the *composition of activities* from lower-level process steps can be verified by using trace segmentation. The event clusters which were discovered by trace segmentation correspond much better to the actual process execution than the idealized job step mapping as envisioned by ASML. Consequently, trace segmentation provides an additional dimension for the verification of real-life processes.

Trace segmentation is also an effective technique for *event log pre-processing*. By elevating the event log onto a higher, more suitable level of abstraction, other process mining techniques (e.g., for control flow discovery) can discover process models which are more suitable for analysis, since they feature activities that correspond to the expectations of analysts.

Future research in the area of trace segmentation should concentrate on finding efficient methods for the recognition of repetitive event patterns, especially for situations where event classes can belong to more than one pattern. To increase the usefulness of the presented approach in practice, future extensions of the implementation should allow the user to (a) manually correct errors of the algorithm, and (b) to provide names for the found higher-level activities, before actually simplifying and further analyzing the log.

Acknowledgements

We want to thank ASML for providing us with their data, and particularly thank Ivo de Jong for his collaboration and interest in process mining.

References

1. W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, Berlin, 2007.
2. 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 and Knowledge Engineering*, 47(2):237–267, 2003.
3. R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley-Interscience, New York, NY, USA, 2000.
4. G. Greco, A. Guzzo, L. Pontieri, and D. Saccá. Mining Expressive Process Models by Clustering Workflow Traces. In *Advances in Knowledge Discovery and Data Mining (PAKDD 2004)*, volume 3056 of *Lecture Notes in Computer Science*, pages 52–62. Springer-Verlag, Berlin, 2004.
5. C.W. Günther and W.M.P. van der Aalst. Mining Activity Clusters from Low-Level Event Logs. BETA Working Paper Series, WP 165, Eindhoven University of Technology, Eindhoven, 2006.
6. C.W. Günther and W.M.P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer-Verlag, Berlin, 2007.
7. A. Rozinat, I.S.M. de Jong, C.W. Günther, and W.M.P. van der Aalst. Process Mining Applied to the Test Process of Wafer Steppers in ASML. *IEEE Transactions on Systems, Man, and Cybernetics–Part C, DOI: 10.1109/TSMCC.2009.2014169*, 2009.
8. A. Rozinat, I.S.M. de Jong, C.W. Günther, and W.M.P. van der Aalst. Process Mining of Test Processes: A Case Study. BETA Working Paper Series, WP 220, Eindhoven University of Technology, Eindhoven, 2007.
9. M. Song, C.W. Günther, and W.M.P. van der Aalst. Trace Clustering in Process Mining. In *Proceedings of the 4th Workshop on Business Process Intelligence (BPI'08), BPM Workshops 2008*, 2008.
10. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.