Trace Clustering in Process Mining

Conference Paper *in* Lecture Notes in Business Information Processing · September 2008

DOI: 10.1007/978-3-642-00328-8_11 · Source: DBLP

3 authors, including:

Minseok Song
Pohang University of Science and Technology
73 PUBLICATIONS **4,682** CITATIONS

SEE PROFILE

Wil Van der Aalst
RWTH Aachen University
1,261 PUBLICATIONS **71,085** CITATIONS

SEE PROFILE

Trace Clustering in Process Mining

M. Song, C.W. Günther, and W.M.P. van der Aalst

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

Abstract. Process mining has proven to be a valuable tool for analyzing operational process executions based on event logs. Existing techniques perform well on structured processes, but still have problems discovering and visualizing less structured ones. Unfortunately, process mining is most interesting in domains requiring flexibility. A typical example would be the treatment process in a hospital where it is vital that people can deviate to deal with changing circumstances. Here it is useful to provide insights into the actual processes but at the same time there is a lot of *diversity* leading to complex models that are difficult to interpret. This paper presents an approach using *trace clustering*, i.e., the event log is split into homogeneous subsets and for each subset a process model is created. We demonstrate that our approach, based on log profiles, can improve process mining results in real flexible environments. To illustrate this we present a real-life case study.

Key words: Process mining, trace clustering, process discovery, data mining, K-means, Quality threshold, SOM, case study

1 Introduction

The field of process mining is concerned with extracting useful information about process execution, by analyzing these event logs. Process mining approaches can be used to measure the *conformance* of process executions to a prescribed process model [10], or to *extend* a known process model with additional information extracted from event logs. The major application of process mining, however, is *discovery*, i.e. the extraction of abstract process knowledge (e.g., process models) only from event logs [1, 2, 4, 9, 12].

For well-structured processes, e.g. workflows, the discovery aspect of process mining has limited appeal, since it is bound to confirm the prescribed characteristics of the process. However, most real-life business processes are *not strictly enforced* by the supporting information system. This means that, while there exists a notion of the process, involved actors are able to deviate from it, or even completely ignore it. Examples for such *flexible environments* are healthcare, product development, or customer support. In these environments, discovering the *actual process* which is being executed is valuable. Such assessment of the current situation is a prerequisite for any process improvement or quality control endeavor.

There are, however, inherent problems of applying process mining to flexible environments. Since these environments typically allow for a wide spectrum of potential behavior, the analysis results are equally *unstructured*. The typical problems observed in resulting process models are an overwhelming number of task nodes, and equally large numbers of relations, resulting in the stereotypical "spaghetti models". One dominant factor contributing to unstructured mining results is *diversity* of an event log, i.e. single cases differ significantly from one another. It is, in such cases, a valid assumption that there are a number of *tacit process variants* hidden within one event log, each significantly more structured than the complete process.

The problem with diversity, i.e. that a set of cases are structured very differently, obviously grows with the number of cases being analyzed. Thus, one solution to this problem is to reduce the number of cases which are analyzed at once. Tacit processes will usually not be explicitly known, i.e. there is no available knowledge on how to partition the set of cases. One can, however, measure the similarity of cases and use this information to divide the set of cases into more homogeneous subsets.

In this paper, we present a trace clustering methodology which implements this *divide-and-conquer* approach in a systematic manner. It is based on a set of *profiles*, each measuring a number of *features* for each case from a specific perspective. Based on these *feature matrices*, a number of *distance metrics* can be applied to compute the relative distance between any two cases in the log. Finally, data clustering algorithms can be applied, grouping closely related cases into subsets. These subsets can subsequently be analyzed independently from one another, which improves the quality of mining results significantly for flexible environments. We have implemented this approach in the context of the ProM framework, and verified its effectiveness in various case studies.

This paper is organized as follows. The next section introduces a running example, which is used to illustrate concepts. Section 3 introduces log profiles, which are used to characterize cases. Subsequently, Section 4 describes algorithms for clustering cases based on these profiles. Section 5 demonstrates the usefulness of our approach using a real-life case study (a hospital). Related work is featured in Section 6, and Section 7 concludes the paper.

2 Running Example

The example process is the repair process of products within an electronic company that makes navigation systems and mobile phones. The process starts with the "Receive an item and a repair request" task (A). The customer sends his broken item to the company and requests repair. After receiving the request, a preliminary check (B) is carried out to find its faults. In parallel, the warranty is checked (C) if the product is a mobile phone. Then, based on the status of the item and the warranty of the customer, repair costs are calculated and passed back to the customer. If the customer decides to repair the product, the product is repaired (E) and subsequently a bill for payment is issued (G). If the product

is a navigation system, a field test (F) is performed in-between repairing the product and issuing payment. If the customer does not want to repair, a cancellation letter (H) is sent. After that, the item is returned (I) and the case is closed.

Table 1 shows an event log in a schematic way. The log is consistent with the process mentioned above. Each row refers to a single case and is represented as a sequence of events. Note that we use the term "case" to refer to a specific row and the term "trace" to the sequence of events within a case. Events are represented by the case identifier (denoted by the row, e.g., 1), activity identifier (first element, e.g., A), and originator (second element, e.g., John). Based on this log, the α-algorithm automatically constructs the Petri net model depicted in Figure 1.

To check the accuracy of the generated model, we can perform a conformance check. In [10] two types of metrics are proposed: (1) fitness and (2) appropriateness. Fitness quantifies how much of the observed behavior is possible according to the model (i.e., "Does the observed process comply with the control flow specified by the process model?"). If a model is able to "replay" the traces in the log, the model may still be inappropriate because it is unnecessary complex or allows for too much behavior (i.e., behavior not supported by observations in the log). Appropriateness tries to capture the idea of Occam's razor, i.e., "One should not increase, beyond what is necessary, the number of entities required to explain anything". In [10] a metric is given for behavioral appropriateness (i.e., "Does the model describe the observed process in a suitable way?").

The model depicted in Figure 1 has a fitness of 0.9 and a behavioral appropriateness of the model is 0.97, i.e., even though there are just a few cases the result is not optimal. Clearly, a better model could be constructed by using a more advanced process mining algorithm. In fact most of the more recent techniques will generate a perfectly fitting model. However, the goal of the example is to show that it is possible to get a better model by clustering cases while still using the α-algorithm.

Case ID	log events
1	(A,John),(B,Mike),(D,Sue),(E,Pete),(F,Mike),(G,Jane),(I,Sue)
2	(A,John),(B,Fred),(C,John),(D,Clare),(E,Robert),(G,Mona),(I,Clare)
3	(A,John),(B,Pete),(D,Sue),(E,Mike),(F,Pete),(G,Jane),(I,Sue)
4	(A,John),(C,John),(B,Fred),(D,Clare),(H,Clare),(I,Clare)
5	(A,John),(C,John),(B,Robert),(D,Clare),(E,Fred),(G,Robert),(I,Clare)
6	(A,John),(B,Mike),(D,Sue),(H,Sue),(I,Sue)

Table 1. Example process logs (A: Receive a item and repair request, B: Check the item, C: Check the warranty, D: Notify the customer, E: Repair the item, F: Test the repaired product, G: Issue payment, H: Send the cancellation letter, I: Return the item)

If we divide the log into several subgroups and construct process models, we can derive more accurate models. For example, based on the tasks, the log can

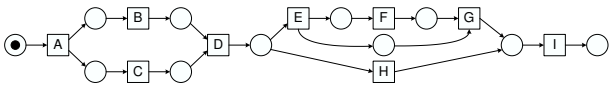


Fig. 1. The example process model

be divided into three groups. The first group consists of cases where a navigation system needs to be repaired (i.e., cases 1 and 3), i.e., the cases where the "Check the warranty" task is missing but with the "Test the repaired product" task. The second group corresponds to the process of repairing a mobile phone (i.e., cases 2 and 5). These cases do not have the "Test the repaired product" task, but have the "Check the warranty" task. The third group corresponds to cases where a repair is canceled, i.e., case 4 and case 6 belong to this group. Figure 2 shows process models from each group again constructed using the α-algorithm. Both the fitness and behavioral appropriateness of these three models is 1.000. This shows that trace clustering can support the identification of process variants corresponding to homogenous subsets of cases. Moreover, the constructed models have a much better quality.

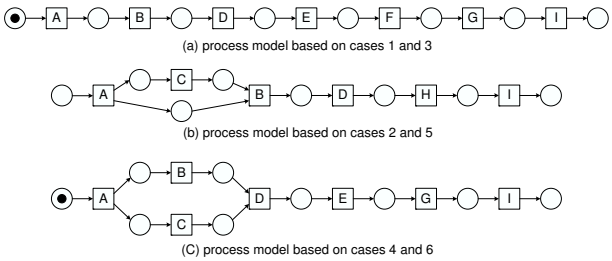


Fig. 2. The derived process models from three groups

3 Trace Profiles

For every clustering application, it is crucial to appropriately design a way to determine the *similarity* of points to be clustered. In this paper, points correspond to cases, i.e., process instances that left a trace in the log. In our trace clustering approach, each case is characterized by a defined set of *items*, i.e., specific features which can be extracted from the corresponding trace. Items for comparing traces are organized in *trace profiles*, each addressing a specific *perspective* of the log.

In the following section, the typical set of information found in an event log is introduced. Section 3.2 describes a set of profiles for characterizing traces, based on that information.

3.1 Information in Event Logs

Event logs are the typical starting point for any process mining endeavor. In order to provide a common basis for process mining research, the *M*ining *XML* (MXML) format has been defined. MXML provides a framework for structuring event logs, which is organized as follows.

The top-level entity of an MXML log is the *WorkflowLog*, which may group any number of *Process* elements. Processes can contain an arbitrary number of *ProcessInstance* elements, each corresponding to a case, i.e. one specific execution of the process. Finally, process instances contain an ordered list of *AuditTrailEntry* elements, i.e. events. Every event needs to have a name, the *WorkflowModelElement*, where two events with the same name are considered to represent the same action. Another mandatory event attribute is the *event type*, identifying lifecycle transitions (i.e., whether the event refers to a task having been started, completed, etc.). Further optional data fields are the event's *timestamp* (i.e., exact date and time of occurrence) and *originator* (i.e., name of a resource having triggered the event). To make the format extensible, every element (i.e., WorkflowLog, Process, ProcessInstance, and AuditTrailEntry) may contain additional *data attributes*, i.e. arbitrary key-value pairs of strings.

3.2 Profiles

Every clustering algorithm attempts to group sets of similar points, whereas for trace clustering, the points to be clustered are log traces. Thus, in order to cluster traces in a meaningful way, we first have to establish *what makes two traces similar*. Event logs in the MXML format contain a large amount of structured information. Some of this information is explicit (e.g., names of events), however there is also derived information, e.g. the number of events within a trace.

In our approach, traces are characterized by *profiles*, where a profile is a set of related *items* which describe the trace from a specific *perspective*. Every item is a *metric*, which assigns to each trace a specific numeric value. Therefore, we can consider a profile with n items to be a function, which assigns to a trace a vector $\langle i_1, i_2, ..., i_n \rangle$. *Profiling* a log can be described as measuring a set of traces with a number of profiles, resulting in an aggregate vector (containing the values for each measured item in some defined order). These resulting vectors can subsequently be used to calculate the *distance* between any two traces, using a *distance metric* (cf. Section 4.1).

Table 2 shows the result of profiling the example log from Section 2 with two profiles. The *activity profile* defines one item per type of activity (i.e., event name) found in the log. Measuring an activity item is performed by simply counting all events of a trace, which have that activity's name. The *originator profile*, which is also shown in this example, is similar to the activity profile. Its items are the originators of the log, counting how many events have been caused by each originator per trace. Each row of the table corresponds to the profile vector of one trace in the log.

Table 2. Activity and originator profiles for the example log from Table 1.

Case ID	Activity Profile									Originator Profile								
	A	B	C	D	E	F	G	H	I	John	Mike	Sue	Pete	Jane	Fred	Clare	Robert	Mona
1	1	1	0	1	1	1	1	0	1	1	2	2	1	1	0	0	0	0
2	1	1	1	1	1	0	1	0	1	2	0	0	0	0	1	2	1	1
3	1	1	0	1	1	1	1	0	1	1	1	2	2	1	0	0	0	0
4	1	1	1	1	0	0	0	1	1	2	0	0	0	0	1	3	0	0
5	1	1	1	1	1	0	1	1	0	2	0	0	0	0	1	2	2	0
6	1	1	0	1	0	0	0	1	1	1	1	3	0	0	0	0	0	0

Based on the typical information found in event logs we can derive various profiles. Some additional examples are:

Transition: The items in this profile are *direct following relations* of the trace. For any combination of two activity names $\langle A, B \rangle$, this profile contains an item measuring how often an event with name A has been directly followed by another event name B. This profile is useful for comparing the *behavior* of traces.

Case Attributes: The items of this profile are the data attributes of the case. In many practical situations, traces are annotated with meta-information, which can be compared by this profile.

Event Attributes: The items in this profile are the data attributes of all events in the log. Item values are measured according to how many events in a trace are annotated with the respective attribute. This profile can capture similarity of traces by comparing the meta-information of their contained events.

Performance: In contrast to other profiles, this profile has a predefined set of items. The *size* of a trace is defined as its number of events. When timestamp information is available, further items measure the *case duration*, and the *minimum*, *maximum*, *mean*, and *median* time difference between events for each trace.

The profiles mentioned so far, capture the information typically available in event logs. When additional information is available in an application domain, it is however straightforward to extend our approach with *custom profiles*. These may define an additional set of items based on domain knowledge, e.g. the value of the shopping basket in a web-based ordering process. By carefully designing such custom profiles, the precision (and, thus, the quality) of trace clustering can be increased significantly.

4 Clustering Methods

In this section, we explain clustering methods. We first explain several distance measures to calculate the similarity between cases. Then we briefly describes four clustering techniques used in this paper.

4.1 Distance Measures

Clustering techniques can use several distance measures to calculate the similarity between cases. The clustering results are influenced by the distance measures. In this section, we explain several distance measures that can be used in clustering.

The profile can be represented as a n-dimensional vector where n indicates the number of items extracted from the process log. Thus, case c_j corresponds to the vector $\langle i_{j1}, i_{j2}, .., i_{jn} \rangle$, where each i_{jk} denotes the number of appearance of item k in the case j. To calculate the distance between cases, we use three distance measures in this paper. They are Euclidean distance [3], Hamming distance [6], and Jaccard distance [11]. They are defined as follows.

- Euclidean distance(c_j,c_k) $= \sqrt{\sum_{l=1}^{n} |i_{jl} - i_{kl}|^2}$
- Hamming distance(c_j,c_k) $= \sum_{l=1}^{n} \delta(i_{jl}, i_{kl})/n$,
  where $\delta(x, y) = \begin{cases} 0 \text{ if } (x > 0 \ \wedge \ y > 0) \vee (x = y = 0) \\ 1 \text{ otherwise} \end{cases}$
- Jaccard distance(c_j,c_k) $= 1 - (\sum_{l=1}^{n} i_{jl} i_{kl}) / (\sum_{l=1}^{n} i_{jl}^2 + \sum_{l=1}^{n} i_{kl}^2 - \sum_{l=1}^{n} i_{jl} i_{kl})$,

4.2 Clustering Algorithm

In this section, we briefly explain clustering algorithms. We use K-means, Quality Threshold (QT), Agglomerative Hierarchical Clustering (AHC), and Self-Organizing Maps (SOM). Using the trace profile concept presented in Section 3, the four clustering techniques can be applied to process mining and create classes of homogenous cases.

K-means Clustering K-means clustering is the most commonly used in practice among partitioning methods, which constructs k clusters by dividing the data into k groups.

Quality Threshold Clustering Quality Threshold clustering (QT) was developed for the field of bioinformatics, or more specifically for the clustering of coexpressed genes. Although it is more computationally expensive than K-means clustering, it is *predictable* (i.e., guaranteed to return the same set of clusters over multiple runs) and does not require to specify the number of desired clusters in advance. Clustering is guided by a *quality threshold*, which determines the maximum diameter of clusters.

Agglomerative Hierarchical Clustering Unlike K-means and Quality Threshold, Agglomerative Hierarchical clustering (AHC) gradually generate clusters by merging nearest traces, i.e., smaller clusters are merged into large ones. The result of AHC is usually illustrated as a dendrogram that shows hierarchy of clusters.

Self-Organizing Map Self-Organizing map (SOM) is one of neural network techniques and is used to map high dimensional data onto low dimensional spaces (e.g. 2D). The network includes a number of neural processing elements (neurons or nodes) which are usually represented as rectangular or hexagonal grids. Each of them is connected to the input. The aim of SOM is grouping similar cases close together in certain areas of the value range. Similar cases are mapped onto the same node or neighboring nodes in the SOM.

Using the trace profile concept presented in Section 3, four clustering techniques described in this section can be applied to process mining and create classes of homogenous cases.

5 Case Study

To validate the approach discussed in this paper, we have performed several case studies. This section explains one of them in detail. Note that, we used the Prom framework to perform the case studies. ProM[1] has been developed to support various process mining algorithms. We have *implemented the trace clustering plug-in in ProM* to support the methods described in this paper. It allows us to cluster cases in a log and further apply other process mining techniques to each cluster.

The case study uses a process log from the AMC hospital in Amsterdam, a large academic hospital in the Netherlands [8]. The raw data contains data about a group of 619 gynecological oncology patients treated in 2005 and 2006 and for which 52 diagnostic activities have been recorded. The process for gynecological oncology patients is supported by several different departments, e.g. gynecology, radiology and several labs. In the log, patients correspond to cases, thus there are 619 cases. The log has 3,574 events and 34 departments are involved in the process execution.

We use the heuristic mining algorithm to derive the process model. Figure 3 shows the process model for all cases obtained using the Heuristics Miner. The generated model is spaghetti-like and too complex to understand easily, since it consists of a lot of activities and many links exist between the activities. We showed this diagram to domain experts in the hospital and they failed to understand the process.

[1] See http://www.processmining.org for more information and to download ProM and the trace clustering plug-in developed in the context of this paper.

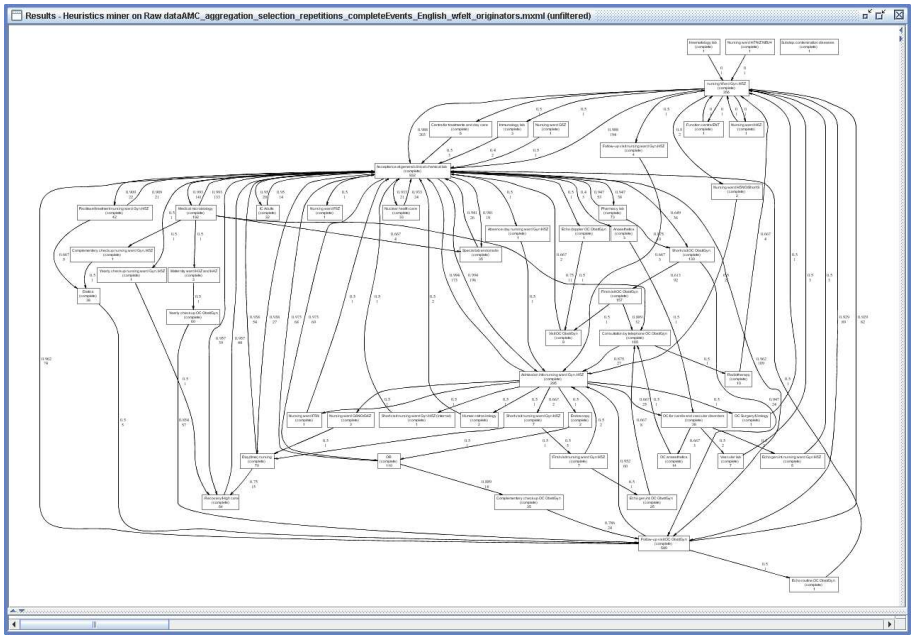


Fig. 3. Process model based on the entire log

Since the clustering results are usually influenced by the characteristics of data, we performed several experiments with various distance measure and clustering techniques in Section 3. In this particular case, the combination of the Euclidean distance and the SOM algorithm is performing better than others. Note that, we only consider the *activity profile* defined in Section 3. Figure 4 shows the clustering result obtained by applying the *Trace Clustering plug-in*. Eleven clusters are obtained from the log. In the figure, the cases in the same cell belong to the same cluster. The figure also shows a contour map based on the so-called *U-matrix* [3] value of each cell. It is very useful to show the distance relationships in a high dimensional data space.

By using this approach, we obtained several clusters of reasonable size. In this paper, we show only the results for cluster (1,2) and cluster (3,1) that contain 352 cases and 113 cases respectively. Figure 5 shows the two heuristic nets derived from these two clusters. Figure 5(a) shows the result from the cluster (1,2). Even though the cluster has 352 cases, which are more than half of the entire cases, it has only 11 activities and the generated model is relatively simple. However the model from the cluster (3,1) is as complex as the original process model (cf. Figure 3). A closer inspection of these two clusters by domain experts showed that cluster (1,2) (Figure 5(a)) corresponds to most of patients who are diagnosed by another hospital and are referred to the AMC hospital for treatment. They only visit the department once or twice and are referred to another department for treatment. Interpreting the cluster (3,1) was difficult because of its complexity. However, though only small number of activities and links are removed from the

original model, it enables the experts to interpret the meaning of the cluster. The cluster (3,1) (Figure 5(b)) corresponds to patients who are not diagnosed. Thus they need more complex and detailed diagnostic activities. Hence these clusters correspond to meaningful processes. This confirms that trace clustering indeed makes it possible to separate different kinds of processes from a process log by dividing it into several groups in which cases have similar properties.

6 Related Work

There is a growing interest in process mining. Process mining allows for the discovery of knowledge based on so-called "event logs", i.e., a log recording the execution of activities in some business processes [2]. Recently many techniques and tools for process mining have been developed [2]. The mainstream of process mining is to discover process models from process logs. It aims at creating a process model that best describes the set of process instances. To check whether the modeled behavior matches the observed behavior, the research on conformance checking has been performed. The concept of conformance and several measures are proposed in [10, 4]. Many case studies have been performed to show the applicability of process mining, e.g., [1]. There are some approaches to handle complex process models from real-life process logs, Günther and van der Aalst [5] proposed the fuzzy mining technique that allows for simplifying process models highlighting important flows. Greco et al. [4] used trace clustering to discover expressive process models. They only used activities and their transitions to make clusters. However, in our approach, we generate profiles considering several perspectives such as control-flow, organization, data, etc. Thus, it is possible to derive clusters based on not only activities and their transitions, but also

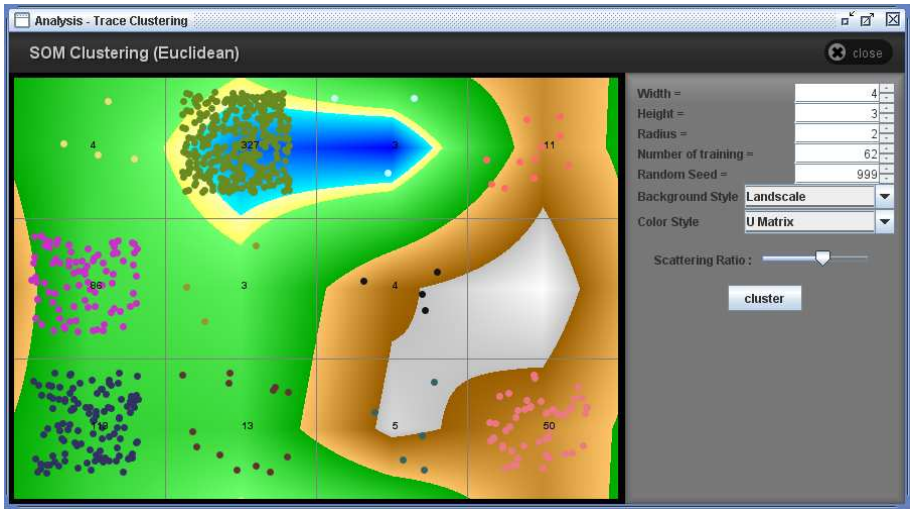
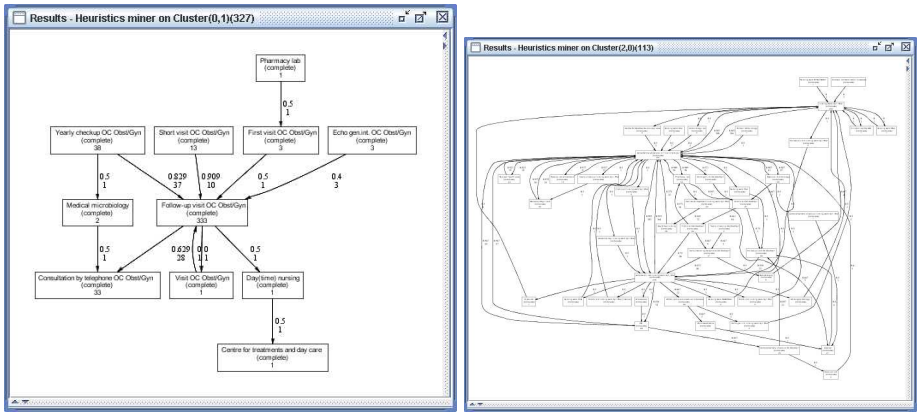


Fig. 4. Trace clustering result



(a) process model for cluster (1,2) (b) process model for cluster (3,1)

Fig. 5. Process models for the clusters (1,2) and (3,1): (a) turns out to be the diagnosis process and (b) turns out to be the treatment process

originators, data, performance, etc. Moreover, unlike [4] we can also influence the attributes used for clustering, thus making the results more intuitive.

The clustering methods [3] we used in this paper are very popular in data mining area. Such data mining techniques have been widely applied in various domains, but their application in process mining (i.e. discovering process models) has been limited [9, 7].

7 Conclusion

Process mining techniques can deliver valuable, factual insights into how processes are being executed in real life. This makes them especially important for analyzing flexible environments, where actors and even process owners are often unaware of how exactly the process is structured. In such domains, however, process mining algorithms tend to generate complex, unstructured process models, which are hard to understand, and thus of limited value. One reason for these problems is diversity, i.e. processes which can generate a set of cases that are structured very differently.

In this paper, we have presented a generic methodology for trace clustering, which can effectively resolve diversity-related problems, by dividing the log into smaller, more homogeneous subsets of traces. We have introduced the concept of trace profiles, which are a suitable means for characterizing and comparing traces. Based on these profiles, any clustering algorithm can be employed for the actual partitioning of the log. We demonstrated the applicability of our approach with a real process log. Our approach has been fully implemented in the

context of the ProM framework, and can be used as a generic log pre-processing operation. Since trace clustering operates on the event log level, it can be used to improve the results of any process mining algorithm. It is noteworthy that both our approach and implementation are straightforward to extend, e.g. by adding domain-specific profiles or further clustering algorithms.

Acknowledgements
This research is supported by EIT, NWO-EW, the Technology Foundation STW, and the SUPER project (FP6). Moreover, we would like to thank the many people involved in the development of ProM.

References

1. W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713–732, 2007.
2. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
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, and L. Pontieri. Discovering Expressive Process Models by Clustering Log Traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.
5. 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.
6. R. W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Tech Journal*, 9:147–160, 1950.
7. C. Li, M. Reichert, and A. Wombacher. Discovering Process Reference Models from Process Variants Using Clustering Techniques. Technical Report TR-CTIT-08-30 Centre for Telematics and Information Technology, University of Twente, Enschede, 2008.
8. R.S. Mans, M.H. Schonenberg, M. Song, W.M.P. van der Aalst, and P.J.M. Bakker. Process Mining in Health Care. In L. Azevedo and A.R. Londral, editors, *International Conference on Health Informatics (HEALTHINF'08)*, pages 118–125. IEEE Computer Society, January 2008.
9. A.K.A. de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2006.
10. A. Rozinat and W.M.P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.
11. P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 2005.
12. 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.