

k-process: Model-Conformance-based Clustering of Process Instances

Florian Richter, Florian Wahl, Alona Sydorova and Thomas Seidl

LMU Munich, Germany
{richter, seidl}@dbs.ifi.lmu.de
{f.wahl,alona.sydorova}@campus.lmu.de

Abstract. Clustering of objects is a prominent task in many applications. The goal is to uncover similarities or correlations for later reasoning. In process mining trace clustering reveals groups of process instances which behave similar in some dimensions, i.e. usage of same resources, time profiles, activity sequences. Conservative clustering on event sequences is usually performed by mapping case attributes into a vector space and applying a vector-based clustering method on these data points. The choice of a suitable vector-space is not trivial and influences the mined results. Our novel approach is designed as a process oriented alternative and avoids a vector-space embedding. Combining the simplicity of *k*-means as a clustering strategy and the expressiveness of process models as cluster representatives results in a data-driven aggregation of instances instead of a tedious vector space evaluation.

Keywords: Trace Clustering, Log Partitioning, Process Mining

1 Introduction

Clustering is one of the most fundamental tasks in the wide field of data science. The various domains of its applications are as numerous as the number of clustering techniques. Objects of different domains and of various shapes shall be aggregated based on the similarity of their object properties. In this work we focus on the aggregation of event sequences.

In the prominent field of process mining [1], event sequences are the main research focus and represent process executions. Especially large enterprises store these executions, which consist of their daily operations and data in the form of log files. Mostly relying on enterprise-resource-planning software every action in the business is logged. The approaches to transform this raw logged data into valuable business intelligence are covered by the term business process mining. However, processes occur in many domains, from low-scale scenarios like serving customers in a restaurant to high-scale applications like customer support in a global company.

Especially in large-scale applications with many process executions not all instances are performed in the same way. Either due to changes over time caused

Copyright ©2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

by changing requirements or by different actors performing the same task differently, a common process can be split into subprocesses. Each event sequence can be a member of one of the subprocesses or a candidate for more than one subprocess. Some applications require the option to allow sequences to be outliers or noise. In this work we perform partition clustering which assigns every sequence to exactly one cluster. Although events can contain various information like their duration or used resources, we only consider the sequences of actions and perform trace clustering. As an example, imagine an airplane boarding process. The time spent for the boarding phase is usually a very costly process and every delay reduces the income of the airline. Here it is beneficial to identify potential clusters, which refer to groups of people, that take more time due to more complex boarding steps. After identification of difficult and time-consuming customer groups, the process can be redesigned by shifting actions to other time frames, e.g. loading wheelchairs in advance, pre-validate complex travel grants, put families with small children into extra queues etc.

Our novel approach k -process clusters sequences of events depending on how well they fit into a common process model. We utilize the very well known k -means framework as a baseline and substitute the two main steps centroid determination and cluster assignment with appropriate process mining methods. Process models can be discovered with process mining techniques like the α -Miner [2]. Process models provide a suitable representation of trace clusters. Fitness computations like token replay perform the cluster assignment then.

2 Related Work

Clustering is a very well researched domain in data mining. Even the closer field of sequence clustering, which is very prominent in bioinformatics, is too large to discuss related work sufficiently here. Therefore we will focus only on the clustering of event sequences. An event sequence represents a consecutively perceived execution of actions. Although the perception is sequential, events might also occur concurrently or with repetition. As such event sequences are called traces we refer to this domain by the more popular term of trace clustering. In this clustering domain, large sets of traces called event logs are considered. They are the main starting point of process mining. The goal is to split the whole log into sublogs, so that each sublog describes similar action patterns while the similarity of behavior between two sublogs shall be minimized.

There are several different approaches used for trace clustering. The first established methods rely on a vector space embedding by mapping features of the sequences to points in a vector space. The basic approach is a bag-of-events embedding, which ignores the sequential information and represents the sequence by the frequencies of each sequence item. Bag-of-motifs or n -gram approaches increase the dimensionality of the representation in case of long sequences. Attached meta data can also be embedded into the vector space. This allows to use the full range of clusterings operating on vector spaces like k -means.

The vector space embedding of traces was first explored by Greco et al. [11]. Song et al. [13] introduced trace profiles which allow different perspectives and changes of focus on certain aspects in the traces. Ferreira et al. [10] clustered traces by building first-order Markov models for each cluster with the expectation-maximization technique. In [4] Bose et al. introduced a specialized edit distance on traces and proposed an agglomerative hierarchical clustering approach. The same year they developed a second approach presented in [5] that uses subtraces of arbitrary length instead of *n*-grams to represent traces as feature vectors.

These techniques usually yield a partitioning of traces into sublogs such that the clusters have a high intra-similarity and a low inter-similarity as desired. However, the similarity of the sublog traces is not considered. Thus these techniques can be aggregated as passive trace clustering while we now traverse to active trace clustering approaches, which was firstly introduced by De Weerd [6]. ActiTraC, as presented in [6] takes the process discovery perspective into account. Traces are not mapped into a vector space but are greedily aggregated to clusters until the fitness of the model for this cluster drops. Sun et al. [14] also followed the hierarchical clustering method with compound trace clustering CTC. They developed a top-down trace clustering splitting logs in sublogs such that the complexity of each sublog matches the average complexity. The log with the highest model complexity is then split again.

Our novel approach follows the principle of active trace clustering. However, the discussed works consist of hierarchical top-down approaches or feature space embeddings. To the best of our knowledge there are no model-aware trace clustering techniques so far without using a hierarchy. We want to propose an active trace clustering method which allows to specify the number of clusters without defining any distance functions. We achieve this minimal parameter setting by using the framework of *k*-means while avoiding a vector space embedding to keep the model-awareness.

3 Preliminaries.

Now we want to give a brief but formal introduction of the terminology used in process mining.

Every process allows a set of actions to be performed which are formally called activities. We refer to the finite set of all valid activities as \mathcal{A} . This is usually an application dependent set of textual action descriptors. A single execution of an activity is called an event. Events can be assigned to a common case using an identifier. We assume that the cases can be identified using a natural number for simplicity.

Definition 1. *An event $e = (c, a, t) \in \mathbb{N} \times \mathcal{A} \times \mathbb{N}$ is a tuple consisting of a case identifier c , an activity a and a timestamp t .*

Each event describes the execution of a certain activity corresponding to the case c at time t . We can continue with cases and logs:

Definition 2. A log $\mathcal{L} : \mathbb{N} \times \mathcal{A} \times \mathbb{N} \rightarrow \mathbb{N} \cup \{0\}$ is a multiset of events.

Semantically an event is a unique occurrence of an activity. Nevertheless, it can happen that the same case contains repetitions of a particular activity. If such events occur consecutively in a very small time frame, they might share the same event representation. In most applications the temporal granularity is usually chosen with regards to the frequency of events. This means that the resolution of the timestamps prevents multiple occurrences of identical events. As this is rather a technical issue we will consider only sets instead of multisets. Finally, we can introduce the definition for traces:

Definition 3. Let \mathcal{L} be a log and c a case identifier. A **trace** is a finite sequence of events $\sigma_c = (e_1, e_2, \dots, e_n)$ where $\pi_c(e_i) = c$ for all $1 \leq i \leq n$. Furthermore, the trace is ordered based on the event timestamps, formally $\pi_t(e_i) \leq \pi_t(e_j) \iff i \leq j$.

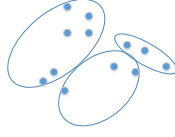

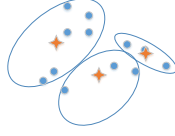

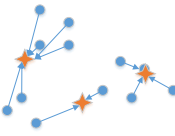
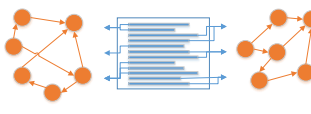
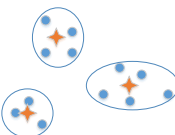
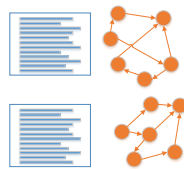
Although a trace consists of consecutive events, the ordering does not enforce a strict linear structure for the trace clustering. In sequence clustering the similarity of two sequences is usually determined by the order and occurrence of its sequence objects. Considering trace clustering, the similarity is depending on the process. Two traces might be different regarding their activity ordering. However if the process allows both execution orders, we can consider both traces as similar. In the following we will introduce our novel approach which puts emphasis on this important aspect by design. Using process mining algorithms includes the process view into the clustering.

4 Algorithm

The central idea of k -process is to use the k -means framework and replace mean computation and cluster assignment with techniques from process mining.

We start by randomly assigning each trace to one of the k cluster candidates which partitions the log into k sublogs. Random assignment is a common strategy for the start of k -means. We use process discovery to determine a representative model for each cluster candidate. So in each iteration process models for all k clusters are recomputed first. As shown in Tab. 1, the process model is used as a centroid for the particular cluster. In the cluster assignment step we determine for each trace the model, that allows the best replay for this trace. This is the bottleneck of this method as conformance checking can be slow for many traces, especially for a large number of activities. Like in k -means we repeat these steps until stability of cluster assignment is reached or a given iteration threshold is exceeded. We used the Heuristic Miner [3, 15] for Model Discovery. Other discovery algorithms can also be applied as long as they generate a model usable for conformance checking. It generates workflow nets, a special case of Petri nets. Token Replay [3] is used for Conformance Checking by replaying traces on a model and handles also varying trace lengths.

Table 1. Comparison of the traditional *k*-means framework for vector data with our novel framework *k*-process incorporating model discovery and conformance checking.

	<i>k</i> -means	<i>k</i> -process
Preprocessing	Vector-space embedding	-
Start Assignment	Assign all points to <i>k</i> partitions 	Assign all sequences to <i>k</i> sublogs 
Centroid Determination	Calculate mean for each partition 	Discover process model for each sublog 
Cluster Assignment	Assign each point to the nearest cluster centroid 	Assign each sequence to the best fitting model 
Output after Convergence	Partitioning of points represented by mean points 	Partitioning of sequences represented by process models 

4.1 Adjustments of the Heuristic Miner

Process discovery is performed for exploratory analysis and conformance checking is mostly used for a small series of traces to check their validity regarding an existing model. We use both methods exhaustively as many iterations can contain many re-discoveries and a magnificent number of conformance checks. The performance of the process discovery is mainly neglectable. The conformance checking, however, reduces the performance significantly. Each trace has to be

replayed k times in each iteration. Therefore we adjust the Heuristic Miner by reducing the accuracy of the fitness computation. We do not need a most accurate fitness score as we only need to determine the best fitting model. In the next step the models are recomputed again, which causes the determined fitness scores to become deprecated after each iteration. To describe parallel executions and forks in the process model the Heuristic Miner uses AND and XOR-relations. Token replay can be performed faster if the model only contains AND-conjunctions, so we discard all XOR-disjunctions.

For a given multiset of activity pairs (a_i, a_j) where a_j occurs directly after a_i , a logical expression can be computed to describe the relations between activities. The output of the Heuristic Miner is a so called Causal Matrix which contains input and output expressions for each considered activity. To efficiently obtain a WF-net which meets the aforementioned requirement we use a simple and straightforward approximation of the relations between activities. First of all we consider only the output expressions and ignore the input expressions which saves a lot of computation time. Furthermore, we only use expressions in the Conjunctive Normal Form (CNF). To approximate an appropriate logical expression for an output set which fulfills these requirements we present the following algorithm: Given an output set O of an activity X we select the biggest subset S where all nodes are in an AND-relation, formally:

$$\begin{aligned} & - \forall a, b \in S : X \Rightarrow_{\mathcal{L}} a \wedge b \\ & - \forall Z \subseteq O : (\forall a, b \in Z : X \Rightarrow_{\mathcal{L}} a \wedge b) \Rightarrow ((S = Z) \vee (|Z| \leq |S|)) \end{aligned}$$

For definition of an AND-relationship see [3, 15].

Therefore, in the expression we generate the elements of S are then in an AND-relation. If this set is empty, all elements are in a XOR-relation and we are done. Otherwise we obtain a conjunction of activities. In the beginning each disjunction of this conjunction only consists of one activity. Then each element $e \in (O \setminus S)$ is checked if it can be added to a disjunction. An activity is added to a disjunction if the number of elements of this disjunction which are in XOR-relation to this activity divided by the total number of elements in this disjunction is higher than a *threshold*. Formally, an element e is added to a disjunction D when $\frac{|\{\forall a \in D | X \Rightarrow_{\mathcal{L}} a \vee e\}|}{|D|} > 0.5$.

We obtain an output expression in CNF for every activity which should be considered in the process model. On this basis, a WF-net can be easily generated. At first we create a labeled transition for each activity. Then we need to translate the output expression into places and arcs. Fig. 1 illustrates an exemplary process model. The determined expressions are $A \rightarrow (B \vee C) \wedge (D \vee C)$, $B \rightarrow F$, $C \rightarrow F$, $D \rightarrow E$, $E \rightarrow G$, $F \rightarrow G$, and $G \rightarrow \text{None}$.

As all expressions are in CNF, we can translate these conjunctions directly to AND-Splits. An AND-Split in a WF-net is represented by arcs from a transition to places, so we create a place for each element of the conjunction (which is a disjunction) and insert an arc from each origin activity to each of its new places. The next step is to insert arcs from the places to the activities of the disjunction. Finally we insert source and sink places.

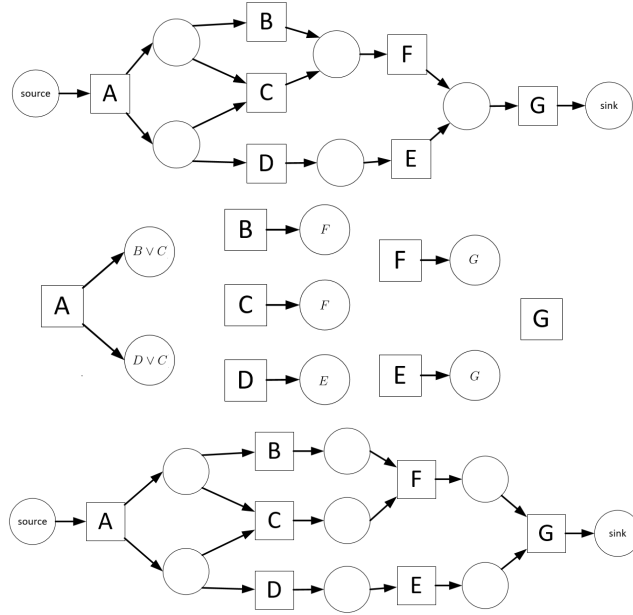


Fig. 1. A process model is transformed into an approximated WF-net by replacing all XOR-disjunctions.

5 Experimental Evaluation

We used real world event logs from the Business Process Intelligence (BPI) Challenge from years 2012 [7], 2015 [8], and 2017 [9]. The datasets can be downloaded from the 4TU.Centre for Research Data¹. In the following these datasets are denoted by L_{12} , L_{15} and L_{17} . Table 2 gives an overview of the size of these event logs.

Table 2. Overview over real life event logs

event log	number of traces	number of activities	number of events
L_{12}	13.087	24	262.200
L_{15}	5.649	500	262.628
L_{17}	31.509	26	1.202.267

The event logs L_{12} and L_{17} describe the same loan application process of a Dutch financial institute, and have 6 activities in common. For our experiments we used excerpts from both of these event logs each containing 100 traces. Then we merged them to one event log and used k -process in order to see if the

¹ https://data.4tu.nl/repository/collection:event_logs_real

merged log is split into the two original event logs again. For each application we randomly sampled excerpts anew. In the following we denote the random sampled event logs as $L_{12} \cup L_{17}$.

The event log L_{15} consists of 5 event logs each from a different Dutch municipality. We processed it the same way as the event logs above. The excerpts also contain 100 traces and we tried to split the merged event log into 5 event logs with the help of k -process. L_{15} denotes in the following the random sampled event logs.

5.1 Examination of clustering goodness measure

We know the ground truth clustering of the event log $L_{12} \cup L_{17}$ which is the splitting in the two original event logs of the years 2012 and 2017. We randomly assigned traces of $L_{12} \cup L_{17}$ to one of the two possible clusters to obtain a randomly generated clustering. Then we compared this clustering with the ground truth to determine the F1-measure and calculated the Silhouette-Coefficient [12] for the generated clustering. Figure 2 (a) shows the achieved F1-Measure of the random clustering against the Silhouette-Coefficient of the clustering for each application. One can clearly see that the value of the Silhouette-Coefficient linearly correlates with the F1-measure, which can be seen from the linear regression line in Figure 2 (a). The value of the Pearson coefficient of 0.995 also indicates a nearly perfect positive linear correlation.

For the second experiment we used the event log of the first municipality from the event log L_{15} denoted by $L_{15,1}$. We created a copy of $L_{15,1}$ with reversed order of events in each trace and denote this new log as $L_{15,\bar{1}}$. Analogously to the first experiment we merged the two event logs ($L_{15,1} \cup L_{15,\bar{1}}$) and generated random clusterings. Figure 2 (b) shows the achieved F1-measure of the random clustering against the Silhouette-Coefficient of this clustering, for multiple applications. Again the Silhouette-Coefficient correlates linear to the F1-measure even if the Pearson coefficient with a value of 0.990 is slightly worse as in the first experiment. Both experiments show that the Silhouette-Coefficient with the Levenshtein-Distance as distance function is a suitable indicator for the goodness of trace clusterings.

5.2 Comparison between k -process and n-grams approach

To compare the proposed k -process method and the n-grams approach we applied both trace clustering techniques to the event logs $L_{12} \cup L_{17}$ and L_{15} . For the n-grams approach we used $n = 2$ since we found out that this setting produced the best results.

We clustered $L_{12} \cup L_{17}$ using the following parameter setting for the Heuristic Miner: *Min-Occurrence-Threshold*=0.3; *Dependency Relation Threshold*=0.5. Figure 3 (a) shows average values of the Silhouette-Coefficient after each iteration of the n-grams approach and k -process. By observing the increase of the Silhouette-Coefficient values one can clearly see how k -process iteratively improves the clustering. In contrast to that, the n-grams approach was not able to

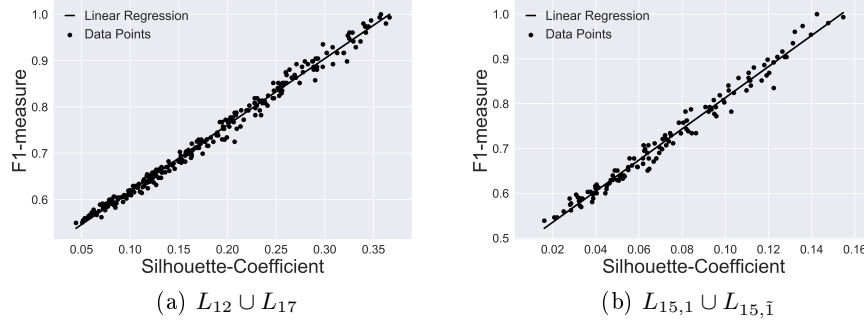


Fig. 2. F1-measure and the Silhouette-Coefficient for randomly generated clustering for $L_{12} \cup L_{17}$ and $L_{15,1} \cup L_{15,\bar{1}}$.

improve the clustering: the average value of the Silhouette-Coefficient stagnates remaining around 0.31. We also applied k -process to L_{15} with the following parameters for the Heuristic Miner: *Min-Occurrence-Threshold*=0.1; *Dependency Relation Threshold*=0.3; *AND-Threshold*=0.1. As one can see in Figure 3 (b) k -process produced better results than the n-grams approach for this event log as well.

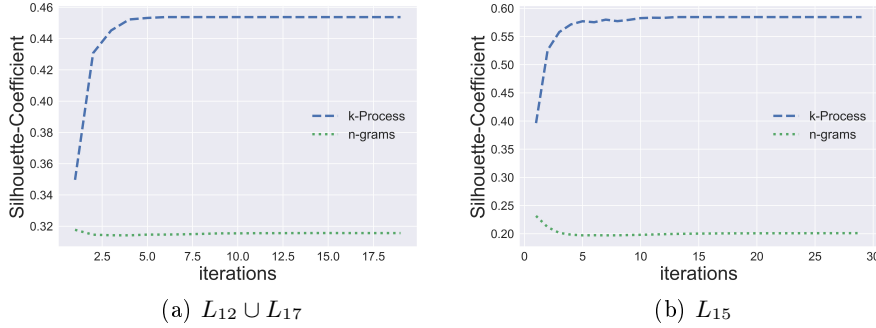


Fig. 3. Average values of the Silhouette-Coefficient after each iteration of the n-grams approach and k -process applied to $L_{12} \cup L_{17}$ and L_{15} .

5.3 Sensitiveness of parameters of the Heuristic Miner

In this section we show the sensitiveness of the parameters *Min-Occurrence-Threshold* and *Dependency Relation Threshold* of the Heuristic Miner. It depends

on these parameters if the Heuristic Miner generates an appropriate model that is neither over- nor underfitted, so that the reassignment step can work properly. With a good parameter setting this works fine as we have seen in the last section. But it is not obvious how to select these parameters while they could affect the whole clustering result. For instance, in the last section we clustered the event log $L_{12} \cup L_{17}$ with the parameters *Min-Occurrence-Threshold*=0.3 and *Dependency Relation Threshold*=0.5 and almost always received a good clustering. But if we change the parameters' values to *Min-Occurrence-Threshold*=0.2 and *Dependency Relation Threshold*=0.7, the Heuristic Miner generates less appropriate models so that k -process does not reach such good results as when applied with the old parameter setting. Figure 4 (a) shows average values of the Silhouette-Coefficient of the clusterings after each iteration of k -process for the old parameter setting used in 5.2 and for the new parameter configuration. The clusterings produced using this new configuration are slightly worse than those with the old one. To encounter the sensitiveness of the parameters we introduce and evaluate a start heuristic in the following section.

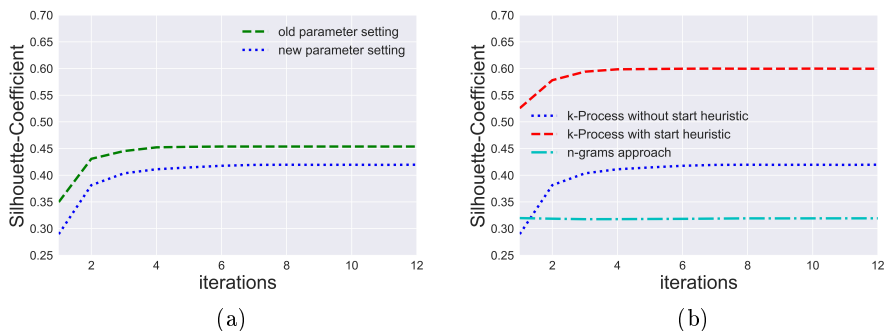


Fig. 4. (a) average values of the Silhouette-Coefficient of the clusterings after each iteration of k -process applied to $L_{12} \cup L_{17}$ using the old parameter setting from 6.3 and the new parameter setting. (b) the Silhouette-Coefficient for the application of k -process on $L_{12} \cup L_{17}$ with and without a start heuristic (using the new parameter setting), and for the application of the n-grams approach.

5.4 k -process with start heuristic

Sometimes the process models generated for each cluster do not differ significantly enough from each other. So, k -process tends to put all the traces into one big cluster while other clusters result empty. To prevent this we introduce a start heuristic, so that the initial clustering no longer follows a random distribution. We also show that using a start heuristic lowers the sensitivity of the parameters of the Heuristic Miner, i.e. even if the parameter setting for the Heuristic Miner

is not optimal *k*-process will still produce good clusterings. We use the n-grams approach with *k*-Means to pre-cluster traces. Thereby, we ensure that there is some essential difference between the initial clusters. Then we apply *k*-process and improve the present clustering.

To test this hypothesis we applied *k*-process once without and once with a start heuristic to the event log $L_{12} \cup L_{17}$. We used the parameter setting for the Heuristic Miner which did not produce very good results without a start heuristic: *Min-Occurrence-Threshold*=0.2; *Dependency Relation Threshold*=0.7. As a start heuristic we used the n-grams approach with n=2. Figure 4 (b) shows the results for *k*-process when applied to $L_{12} \cup L_{17}$ with the not optimal parameter setting with a random clustering initialization, and when a start heuristic is used with the same parameter setting. One can clearly see that the clustering results produced in combination with the start heuristic are much better. We want to remark that the n-grams alone do not produce such a good clustering (see Figure 4 (b)).

6 Conclusion

Clustering is a very prominent data exploration step. Trace clustering in particular helps to identify subprocesses and variants of various process applications. Here we proposed a novel partitioning method based on the well-known *k*-means framework, which consists of finding representatives for partition candidates and redetermine the next representatives until convergence is reached. Instead of a vector space based *k*-means we showed that it is possible to use process model discovery and conformance checking to establish process-aware trace clustering. The quality of the results is better than the results of a clustered vector space embedding. Using the framework of *k*-means, the core steps in *k*-process can be replaced with other discovery and conformance checking methods. This is a major benefit as it allows to adapt the approach to particular applications when needed. Process analysts, who are familiar with process mining techniques, are required to operate the discovery algorithms instead of performing an exhaustive feature engineering to find a suitable vector space embedding.

For future work it would be interesting to experiment with other discovery and conformance checking methods. As both core methods are applied multiple times, faster methods may be preferred to methods producing higher quality models in some scenarios. Adapting online conformance checking methods here might be a promising direction. In the beginning the accuracy is not of great importance and we only need a coarse partitioning of the traces. A strategy to be investigated is to start with a low accuracy high performance assignment and switch gradually to more performance when the final clustering has almost been reached according to cluster metrics like the silhouette coefficient. A hybrid that starts with low quality but fast candidate clustering for the initial steps and ends with more complex methods as a refinement might provide exactly this improvement. For some applications, online clustering might also be needed, as it is interesting to detect deviating subprocesses shortly after they emerge. In many

dynamic applications the process clusters change to quickly and an advantage can only be drawn if the trace clusters are identified fast enough. Then it is possible to analyze them in time and derive useful insights to improve the process. The strategies to determine the initial clustering should be investigated as it determines the number of iterations and the final result, analogous to k -means. Extensive research has been done in this field and we have to find out the suitable strategies for k -process, since processes behave dissimilar to well-researched vector spaces.

References

1. van der Aalst, W.: *Process Mining: Data Science in Action*. Springer (2016)
2. Van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge & Data Engineering* (9), 1128–1142 (2004)
3. Van der Aalst, W.M.: *Process mining*. In: *Process Mining, Data Science in Action, Second Edition*. Springer (2016)
4. Bose, R.J.C., Van der Aalst, W.M.: Context aware trace clustering: Towards improving process mining results. In: *Proceedings of the 2009 SIAM International Conference on Data Mining*. pp. 401–412. SIAM (2009)
5. Bose, R.J.C., van der Aalst, W.M.: Trace clustering based on conserved patterns: Towards achieving better process models. In: *International Conference on Business Process Management*. pp. 170–181. Springer (2009)
6. De Weerd, J., vanden Broucke, S., Vanthienen, J., Baesens, B.: Active trace clustering for improved process discovery. *IEEE Transactions on Knowledge and Data Engineering* **25**(12), 2708–2720 (2013)
7. van Dongen, B.: Bpi challenge 2012 (2012), <https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f>
8. van Dongen, B.: Bpi challenge 2015 (2015), <https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1>
9. van Dongen, B.: Bpi challenge 2017 - offer log (2017), <https://doi.org/10.4121/uuid:7e326e7e-8b93-4701-8860-71213edf0fbc>
10. Ferreira, D., Zacarias, M., Malheiros, M., Ferreira, P.: Approaching process mining with sequence clustering: Experiments and findings. In: *International Conference on Business Process Management*. pp. 360–374. Springer (2007)
11. Greco, G., Guzzo, A., Pontieri, L., Sacca, D.: Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering* **18**(8), 1010–1027 (2006)
12. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* **20**, 53–65 (1987)
13. Song, M., Günther, C.W., Van der Aalst, W.M.: Trace clustering in process mining. In: *International Conference on Business Process Management*. pp. 109–120. Springer (2008)
14. Sun, Y., Bauer, B., Weidlich, M.: Compound trace clustering to generate accurate and simple sub-process models. In: *International Conference on Service-Oriented Computing*. pp. 175–190. Springer (2017)
15. Weijters, A., van Der Aalst, W.M., De Medeiros, A.A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP **166**, 1–34 (2006)