Introducing *k-traceoids*: A Structure-Preserving Trace Clustering Framework

Umut Nefta Kanilmaz[1,2], Gabriel Marques Tavares[1,2], Daniel Schuster[1,2], Rafael Seidi Oyamada[3], and Thomas Seidl[1,2]

[1] Institute for Database Systems and Data Mining, LMU Munich, Munich, Germany
[2] Munich Center for Machine Learning, Munich, Germany
[3] Leuven Institute for Research on Information Systems, KU Leuven, Belgium

Abstract. Event data often exhibit complex and diverse patterns, as traces generated by the same process can vary significantly due to a high number of process execution variants. Trace clustering techniques help analyze event logs by partitioning event data into smaller groups based on similarity. However, existing approaches face significant challenges, especially with respect to representational quality and trace encoding. Existing techniques that transform traces into a vector space or count activity occurrence typically result in the loss of essential sequencing information, as they ignore the order in which events occur. To address these issues, we propose *k-traceoids*, a structure-preserving trace clustering framework inspired by the k-means clustering method, which operates directly on traces. Our results demonstrate the effectiveness of *k-traceoids* in identifying meaningful clusters and show that our method groups together traces that vectorial approaches would not recognize as similar

Key words: Trace clustering, unsupervised learning, process mining

1 Introduction

Event data derived from modern real-world processes are often complex, ambiguous and lack structure [1], making it difficult to extract meaningful insighgts about process behaviour. Clustering techniques for traces, which represent the specific executions of a process, address this challenge. By partitioning event data into smaller, more homogeneous subsets and grouping similar process instances, these methods simplify the analysis of complex behaviors in e.g. process discovery [2] and anomaly detection [3].

However, many existing trace clustering approaches rely on vector space representations, which require mapping traces to numerical feature vectors [4]. This transformation often leads to a loss of essential information, particularly the order of activities, which in turn limits the quality of the subsequent tasks [5, 6].

To address these limitations, we propose *k-traceoids*, a clustering algorithm inspired by the well-known k-means approach [7]. Unlike existing trace clustering approaches, *k-traceoids* preserves the sequential nature of traces by avoiding vectorial encoding and operating directly on traces. *k-traceoids* uses process models,

not vectors, as centroids to represent clusters which enables a structure-aware comparison of traces.

We evaluate our framework on real-world event logs and demonstrate its effectiveness in uncovering meaningful trace groupings. Our results show that *k-traceoids* can cluster together traces that would otherwise appear highly dissimilar in vector space; i.e., with large distances, but are in fact behaviorally close when compared using process models. This highlights the advantages of using process models as centroids, as they better capture control-flow characteristics than statistical aggregations in vector space.

2 Related Work

Trace clustering methods are typically categorized as distance-based [8, 9] or process model-based [2, 10], with some hybrid approaches [11].

Distance-based clustering methods represent event sequences as numerical vectors and apply algorithms like DBSCAN or k-means. Techniques to create numerical representations range from one-hot encoding [6] and n-grams [12] to learned embeddings [8]. Most approaches use Euclidean distance although recent work explores alternative metrics tailored to process data [13]. However, creating these vector representations is non-trivial [5, 6] and risks the loss of information.

Process-model based clustering techniques, in contrast, retain the structural information of traces. For instance, ActiTrac [2] applies principles of active learning to select a subset of traces for inclusion to clusters. Similar to *k-traceoids*, this method uses process models and fitness functions to determine cluster membership. However, ActiTrac iteratively adds traces to a single cluster until a target fitness threshold is met, then proceeds to form the next cluster. A limitation of this approach is that recalculating the centroid after each iteration can heavily influence the cluster stability. Our approach differs fundamentally: we generate all clusters simultaneously, reassign all traces in a single step, and update all models collectively.

The entropic clustering method proposed in [10] groups process variants by minimizing entropic relevance (ER) scores with respect to cluster-specific DFGs. This method is tightly bound to DFGs and specifically optimized for computing ER scores on them, whereas *k-traceoids* offers a greater flexibility.

Trace clustering is often used as a preprocessing step for tasks such as predictive monitoring [14] and process analytics [15]. A key challenge, however, lies in selecting an appropriate clustering method for a given dataset [4]. *k-traceoids* addresses this by offering a flexible framework that can be adapted to diverse datasets through parameter tuning, rather than requiring method-specific optimization.

3 Framework

This paper explores the topic of trace clustering, proposing a generic algorithm and implementation inspired by k-means. Similarly to k-means, we begin by specifying the number of clusters to detect, then assign traces grouped by variants to clusters in the initalization step. However, instead of centroids, i.e. the arithmetic mean of instances, we use process models to represent each cluster. The objective is to minimize the distance between each trace and its closest process model by maximizing its fitness to a given model, analogous to how k-means minimizes the distance between data points and centroids.

Table 1. Comparison of main concepts between *k-traceoids* and k-means.

Category	***k-traceoids***	**k-means**
Data Type	Traces	Numerical vectors
Initial Assignment	Balances assignment by variant size	Random choice of k centroids, distance-based assignment
Cluster Representation	Process model	Cluster centroid
Distance Measure	Conformance trace to model	Distance to centroid
Reassignment Step	Trace to cluster with highest fitness	Point to closest centroid
Convergence Criteria	Stable trace assignments	Stable point and centroid assignments

Table 1 highlights the main conceptual analogies and differences between our approach and the k-means algorithm. One important difference is that k-means is designed to work on vectorial spaces comprised of real numbers, whereas our approach operates directly on traces. This design decision eliminates the need to find a derived representation of the traces in a numerical vector space and is therefore structure-preserving and avoids information loss from embeddings. Our algorithm groups traces belonging to the same variant and, during initialization, distributes these variants across clusters ensuring the number of traces per cluster is somewhat balanced. Another notable difference is the cluster representation and the corresponding distance measures. For k-means, the cluster is represented by a virtual data point in the cluster center, the centroid, and the distance to the centroid is defined in the same vector space. For our approach, however, a process model represents the cluster, and the distance is measured through the fitness of a trace to a cluster. Convergence is defined in both cases through the stability of assignments, i.e., that the traces, or the data points and centroid assignments, respectively, do not change. Figure 1 describes the workflow of our approach comprising the following steps:

1. Initialization. The input consists of an event log containing n traces distinguished by a unique case identifier. The desired number of clusters, k, as well as the maximum number of iterations $max\_iterations$ to control the

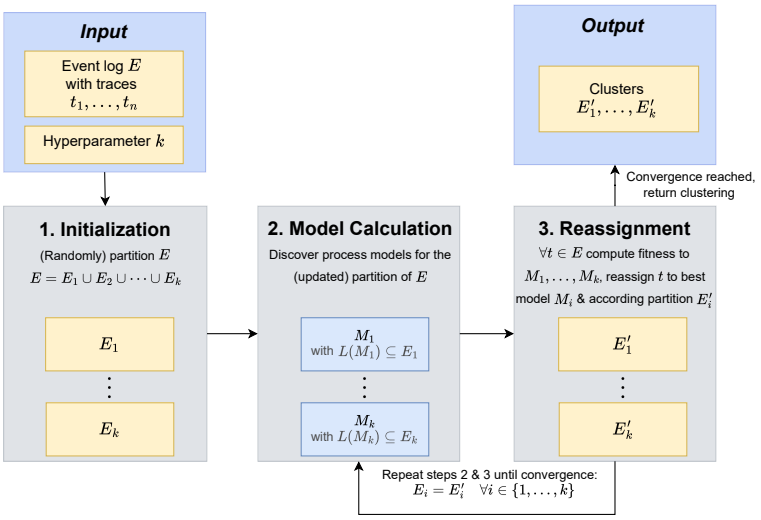

Fig. 1. The workflow of *k-traceoids* comprising initialization, model calculation and reassignment of traces.

convergence, are set. For initialization, each of the n traces are grouped by their according variant and each variant is assigned to one of the k clusters while ensuring that the number of traces across clusters is somewhat balanced.

2. Model calculation. To determine the cluster representation, i.e., the centroid, a model is generated given the set of traces within the cluster. Here, *model* is intended as any possible description of a group of traces, e.g., process model, most frequent variant, super variants among others.

3. Reassignment of traces. The cluster fitness is determined next by calculating each trace's conformance to each of the k models and each trace is reassigned to the cluster with the best fitting model, while ensuring that traces belonging to the same variant stay in the same cluster.

Steps 2 and 3 are repeated iteratively until a convergence condition is met. Convergence is reached if there are no changes in the assignment of traces to the clusters between two consecutive iterations, i.e., all traces remained in the same clusters, or the algorithm reaches the predefined maximum number of iterations. Once convergence is achieved, the final process models for each cluster are obtained along with the corresponding assignment of the clusters.

Our approach is highly general, can be instantiated in various ways and is applicable to any type of trace data.

4 Experimental Setup

The objective of our experiment is to demonstrate that the presented *k-traceoids* framework can identify meaningful patterns that capture distinct behaviors in different clusters while preserving the structure of the traces. The experiments were executed on a *Ubuntu 22.04.5 LTS* KVM with 64 CPUs and 480 GB RAM. For our analysis, we chose the well-known Road Traffic Fine Management (RTFM) Process[1] event log due to its inherent structural patterns and its widespread use within comparable research. The code used in this analysis as well as the results for RTFM are fully available for reproducibility reasons in a public repository[2].

Hyperparameter	Instantiation	Description
k	2 - 10	Desired number of clusters
m	IMF, HM	Model centroid calculation
c	TBR, ALB	Distance trace and model

Table 2. The hyperparameters chosen in this analysis. The *k-traceoids* algorithm is evaluated with each hyperparameter combination for RTFM datatset.

Table 2 lists the hyperparameters chosen in this experiment and their respective value ranges, namely k, the number of clusters that *k-traceoids* should detect, m, the process discovery algorithm used to calculate the cluster centroid (i.e., model), and c, the conformance check approach to compute the fitness between a trace and a model. Given RTMF as the input event log, we execute the *k-traceoids* algorithm with all possible hyperparameter combinations of k, m and c.

Upon inspection, the RTFM log shows four to five major variants, hence, we chose a range between $2 - 10$ for the number of clusters to be detected (k). To ensure reproducibility, the random seed was fixed in all experiments.

Process models represent the cluster centroids (m) as they provide a structured representation for a group of traces. Specifically, we employed the Inductive Miner - infrequent (IMF) [16] and the Heuristic Miner (HM) [17] process discovery techniques given their well-known properties and overall acceptance in the literature.

IMF uses Directly-Follows Graphs (DFG) [18] and finds cuts that divide the complete event log into a subset of logs. The same procedure is again applied to the sublogs. These hierarchies of cuts form a process tree which is further translated into a Petri net. IMF is designed to account for infrequent activities in the event log, while procuding sound models. Similarly, the HM algorithm builds a DFG and then counts how often activities are followed by other activities. The key is that only frequent and significant connections are kept, using a

[1] `https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5`
[2] `https://github.com/NeroCorleone/k-traceoids/`

dependency measure that calculates the strength of two consecutive activities. Weak connections, i.e., noisy control-flow relationships, are removed depending on a predefined threshold. The final graph representation is used as the basis to build a process model. Both discovery techniques are state-of-the-art, have been established in many different applications and, thereby, provide a good baseline to be employed by *k-traceoids*.

As distance metric c, we adopted conformance checking as the means to yield fitness of each trace to the discovered process models. Specifically, we applied Token Based Replay (TBR) and Alignment-Based Fitness (ALB) [19]. Both approaches are widely exploited in the literature.

To guarantee convergence, we include a hyperparameter that controls the maximum number of iterations during cluster reassignment, i.e., step three in Figure 1. For this experiment, we chose $max\_iterations = 100$. We further introduced a timeout of 10 minutes for the ALB conformance check, accommodating the fact that calculating the ALB for infrequent and complex traces can take tremendous processing time. In these cases, the distance calculation is interrupted and a fitness value of zero is assigned to a given trace-model pair, ensuring that the algorithm converges in acceptable time.

5 Evaluation

To demonstrate the effectiveness of *k-traceoids*, we first present a qualitative investigation highlighting the main behaviors identified in different clusters. Then, we move into a quantitative analysis to demonstrating cluster convergence and soundness.

5.1 Qualitative Evaluation

We now present a qualitative evaluation for the configuration with $k = 4$, model type *IMF*, and distance measure *ALB* to demonstrate how *k-traceoids* can effectively capture distinct behaviours across separate clusters. Figure 2 illustrates the resulting models, or cluster centroids, that were extracted using *k-traceoids*. The four models differ in size, complexity, and linearity, and group together traces that represent distinct process behaviours. For example, Clusters A and C both allow for loops in the trace, while Cluster B captures long-term dependencies in the processes as well as activity variations.

These distinctions become more apparent when analyzing exemplary traces identified in each cluster, as can be seen in Table 3. In Clusters A and D, several traces repeat the *Payment* activity. These traces are found in the same cluster even though the differ substantially in trace length. However, while Cluster C contains the simple repetition of a single activity (*Payment*), Cluster 0 captures the repetition of two activities (*Payment*, *Add Penalty*). Cluster B, on the other hand, includes traces with long-term dependencies (e.g., *Create Fine*, *Send Fine*, *Inser Fine Notification*) and allows for variations in two specific activities: *Add*

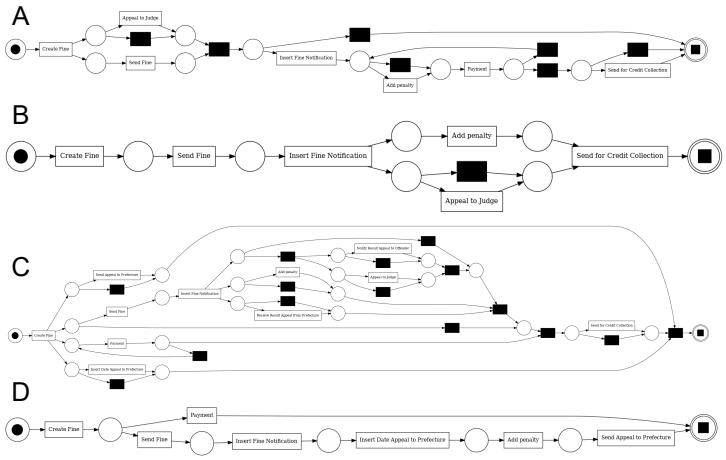

Fig. 2. Clustering outcome for *k-traceoids* configuration with $k = 4$, IMF and ALB. Each clusters groups distinct traces, e.g. cluster A contains loops, while cluster B contains long-term dependencies and variations of two activities.

Penalty and *Appeal to Judge*. Cluster D primarily captures small, linear process executions.

Cluster	**Example Traces**
A	*Create Fine, Send Fine, Insert Fine Notification,* ***Payment, Add penalty, Payment****.* *Create Fine - Send Fine - Insert Fine Notification - Add penalty -* ***Payment (3x)*** *- Send for Credit Collection.* *Create Fine - Send Fine - Insert Fine Notification - Add penalty -* ***Payment (10x)*** *- Send for Credit Collection.*
B	*Create Fine - Send Fine - Insert Fine Notification -* ***Appeal to Judge - Add penalty*** *- Send for Credit Collection.* *Create Fine - Send Fine - Insert Fine Notification -* ***Add penalty - Appeal to Judge*** *- Send for Credit Collection.*
C	*Create Fine - Send Fine -* ***Payment****.* *Create Fine - Send Fine - Insert Fine Notification - Appeal to Judge - Add penalty -* ***Payment (15x)****.*
D	*Create Fine - Payment.* *Create Fine - Send Fine - Insert Fine Notification - Insert Date Appeal to Prefecture - Add penalty - Send Appeal to Prefecture.*

Table 3. Exemplary traces for *k-traceoids* outcome with configuration with $k = 4$, IMF and ALB. Cluster A and D capture repetition of the same activities even with substantial difference in trace length. Cluster C preserves variations of two activities within one cluster.

This qualitative analysis focuses on one specific configuration demonstrating the effectiveness of *k-traceoids*. Due to space constraints, we refer to the accompanying GitHub repository for further evaluation results.

5.2 Quantitative Evaluation

Here, we present quantitative metrics evaluating the clustering outcomes for the hyperparameter combinations listed in Table 2. It is important to note that runs using model HM with distance measure ALB did not yield complete results due to multiple errors encountered during execution. These issues appear to stem from properties of such algorithms that do not match when combined and require further investigation.

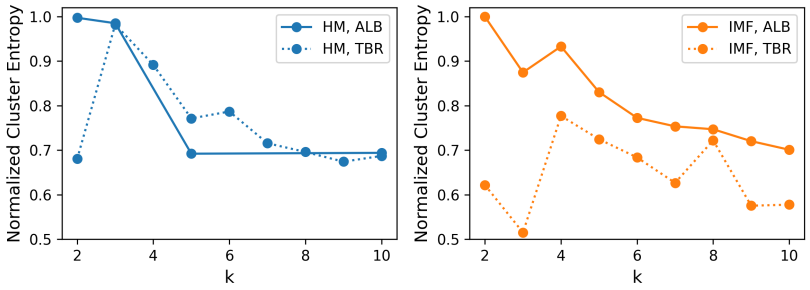

Fig. 3. Normalized Shannon entropy of cluster assignments for different k. Higher values indicate balanced clusters. Entropy tends to decrease with increasing k, particularly for IMF with TBR, suggesting higher cluster imbalance.

Figure 3 presents the Shannon entropy and normalized to the maximum possible value for each k to indicate cluster balance. High entropy values indicate an even distribution of traces across clusters and therefore suggest a balanced clustering, as each cluster is approximately equally likely. In contrast, low values approaching zero reflect a skewed distribution of traces, indicating imbalanced clusters. For most hyperparameter combinations, the clusters are balanced as entropy is often above 0.8. For models discovered by HM, there is no substantial difference in entropy values between the ALB and TBR distance measures, although some data points are missing for the ALB execution. In contrast, for IMF, TBR generally leads to lower entropy values compared to ALB. Notably, for TBR with k as 3, 9 and 10, the entropy is among the lowest observed, suggesting that such combination tends to group the majority of variants together while leaving infrequent variants in smaller groups. This downward trend in entropy with increasing k could be expected. As the number of clusters increases, the constraint that all traces of a given variant must be assigned to the same cluster can lead to uneven cluster sizes. Consequently, the likelihood that some clusters contain significantly more traces than others increases, reducing entropy.

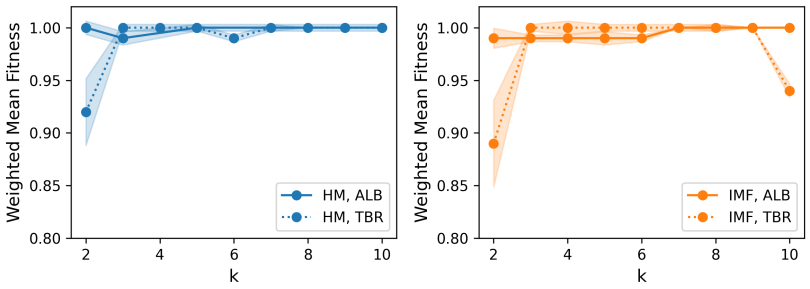

Fig. 4. Mean fitness per k for selected discovery and conformance algorithms weighted by the amount of traces per cluster.

Figure 4 shows the fitness of traces relative to the discovered process models for different k values. Fitness measures how well a trace aligns with model. The mean fitness is calculated for each model, weighted by the number of traces in each cluster. Overall, fitness values are very high across the board, suggesting that the traces are well-represented by their corresponding models. In many cases, fitness scores are exactly 1, indicating meaningful clusters where similar traces are grouped together.

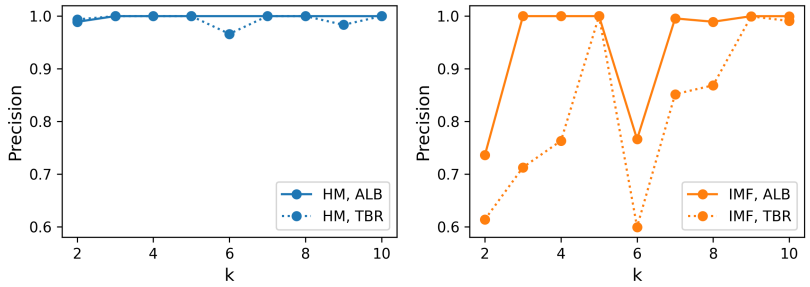

Fig. 5. Precision values per k for selected discovery and conformance algorithms.

Figure 5 presents the precision of the discovered models. Precision is a measure of specificity and evaluates how much unseen behavior a model allows beyond what is present in the event log. A precision value of 1 indicates that the model permits only the behavior observed in the log, meaning the model is highly specific and prone to overfitting. In contrast, a precision value close to 0 indicates a too general model that allows to broad variations, thereby losing interpretability or meaning. Results from Figure 5 demonstrate that HM consistently achieves a precision of 1 across all k values, suggesting potential overfitting. While a perfect precision score might suggest model overfitting in other settings, it is a less relevant concern for clustering traces in process mining, where all data points are available and there is no need to generalize to new

data points. In this context, a perfect precision might hint that *k-traceoid's* is able to find distinct clusters. IMF combined with TBR shows executions with more balanced precision values indicating a better generalization.

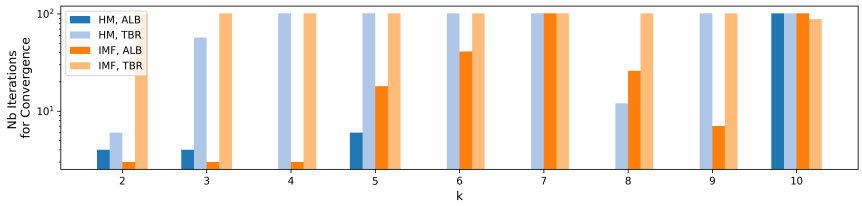

Fig. 6. Number of iterations for convergence per k for selected discovery and conformance algorithms.

Figure 6 shows the number of iterations required for convergence, with maximum iterations at 100. In general, configurations involving IMF + TBR (light orange) and HM + TBR (light blue) required a high number of iterations and often reached the maximum limit. IMF + ALB presents more interesting behavior, converging more quickly for $k = 2$ and $k = 3$, and also with efficient convergence rates for higher values such as $k = 5, 6, 8, 9$ which may represent a good balance between convergence speed and model quality. However, further investigation is needed to understand why convergence is particularly fast at lower k values in this case.

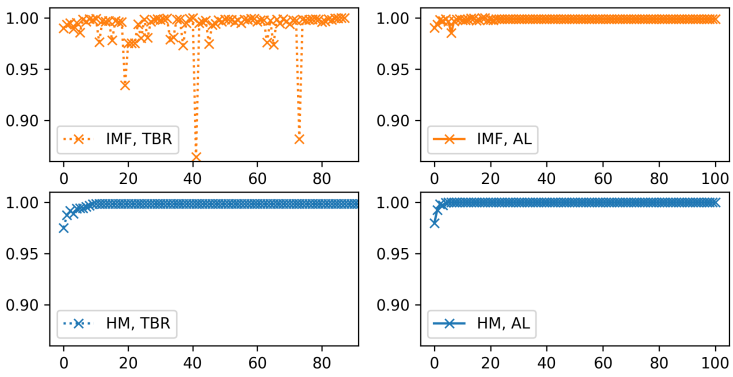

Fig. 7. Cluster agreement between two iterations for $k = 10$

We further investigated the case of $k = 10$, where most configurations converged at the maximum iteration count. We therefore analyzed the cluster stability, i.e. the percentage of traces reassigned between consecutive iterations, see Figure 7. IMF and TBR were the least stable configuration, which is expected

since TBR is known to be less reliable than alignment-based conformance. In contrast, other configurations reached around 98% stability after just 20 iterations. Given the stability of the clustering outcomes, the convergence criterion could be relaxed to stop when a certain percentage of traces remain stable across a certain number of iterations.

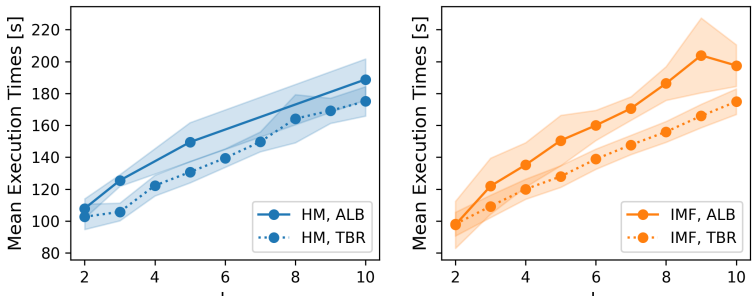

Fig. 8. Mean execution time for each configuration.

Figure 8 shows the mean execution times for all configurations. As expected, execution times increase with larger values of k, since each iteration requires discovering more process models and performing additional alignment checks. While the growth in processing time appears to be roughly linear in k, we refrain from making conclusive statements, as the evaluation is limited to a single dataset. To make reliable claims about execution times, further experiments on larger and more heterogeneous datasets are necessary.

6 Conclusion

The complexity and variability inherent in real-world event data challenge traditional process mining techniques, often resulting in overly complex and less interpretable models. To address this, we introduced *k-traceoids*, a novel trace clustering algorithm that preserves the structural characteristics of event data while avoiding the pitfalls of vector-based encodings. Both qualitative and quantitative evaluation demonstrated the framework's ability to effectively identify meaningful clusters in real-world event data. We have contributed a simple and yet effective framework that is proven to yield suitable results due to the inherent guarantees of the k-means algorithm. *k-traceoids* offers high flexibility in the choice of distance functions and model centroids and can therefore easily be adapted to larger and more diverse real-world datasets.

References

1. Veiga, G.M., Ferreira, D.R.: Understanding spaghetti models with sequence clustering for prom. In Rinderle-Ma, S., Sadiq, S., Leymann, F., eds.: Business Process

Management Workshops. Volume 43 of LNBIP., Springer (2009)
2. Weerdt, J.D., vanden Broucke, S.K.L.M., Vanthienen, J., Baesens, B.: Active trace clustering for improved process discovery. IEEE Trans. Knowl. Data Eng. (2013)
3. Hompes, B., Buijs, J., van der Aalst, W., Dixit, P., Buurman, J.: Discovering deviating cases and process variants using trace clustering. In: 27th Benelux Conference on Artificial Intelligence. (2015)
4. Tavares, G.M., Barbon Junior, S., Damiani, E., Ceravolo, P.: Selecting optimal trace clustering pipelines with meta-learning. In Xavier-Junior, J.C., Rios, R.A., eds.: Intelligent Systems, Cham, Springer (2022)
5. Tavares, G.M., Oyamada, R.S., Barbon, S., Ceravolo, P.: Trace encoding in process mining: A survey and benchmarking. Eng. Appl. Artif. Intell. **126** (2023)
6. Rullo, A., Alam, F., Serra, E.: Trace encoding techniques for multi-perspective process mining: A comparative study. WIREs Data. Mining. Knowl. Discov. (2025)
7. Jin, X., Han, J. In: K-Means Clustering. Springer (2010)
8. Koninck, P.D., vanden Broucke, S., Weerdt, J.D.: act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes. In Weske, M., Montali, M., Weber, I., vom Brocke, J., eds.: Business Process Management Proceedings. Volume 11080 of LNCS., Springer (2018)
9. Zandkarimi, F., Rehse, J., Soudmand, P., Hoehle, H.: A generic framework for trace clustering in process mining. In van Dongen, B.F., Montali, M., Wynn, M.T., eds.: 2nd International Conference on Process Mining, IEEE (2020)
10. Peeperkorn, J., Smedt, J.D., Weerdt, J.D.: Model-driven stochastic trace clustering. CoRR **abs/2506.23776** (2025)
11. Koninck, P.D., Weerdt, J.D.: Scalable mixed-paradigm trace clustering using super-instances. In: International Conference on Process Mining, IEEE (2019)
12. Bose, R.P.J.C., van der Aalst, W.M.P.: Trace clustering based on conserved patterns: Towards achieving better process models. In Rinderle-Ma, S., Sadiq, S., Leymann, F., eds.: Business Process Management Workshops. Volume 43 of LNBIP., Springer (2009)
13. Back, C.O., Simonsen, J.G.: Comparing trace similarity metrics across logs and evaluation measures. In Indulska, M., Reinhartz-Berger, I., Cetina, C., Pastor, O., eds.: International Conference on Advanced Information Systems Engineering. Volume 13901 of LNCS., Springer (2023)
14. Francescomarino, C.D., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Trans. Serv. Comput. **12**(6) (2019) 896–909
15. Seeliger, A., Nolle, T., Mühlhäuser, M.: Finding structure in the unstructured: Hybrid feature set clustering for process discovery. In Weske, M., Montali, M., Weber, I., vom Brocke, J., eds.: International Conference on Business Process Management. Volume 11080 of LNCS., Springer (2018)
16. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable Process Discovery with Guarantees. In Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., Ma, Q., eds.: Enterprise, Business-Process and Information Systems Modeling, Cham, Springer International Publishing (2015) 85–101
17. Weijters, A., Ribeiro, J.: Flexible Heuristics Miner (FHM). In: 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM). (April 2011) 310–317
18. Leemans, S.J., Poppe, E., Wynn, M.T.: Directly Follows-Based Process Mining: Exploration & a Case Study. In: International Conference on Process Mining (ICPM). (2019)
19. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1) (March 2008) 64–95