Complex Systems Informatics and Modeling Quarterly (CSIMQ)
eISSN: 2255-9922
Published online by RTU Press, https://csimq-journals.rtu.lv
Article 226, Issue 41, December 2024/January 2025, Pages 55–79
https://doi.org/10.7250/csimq.2024-41.04

Analysis of Algorithms for Detecting Users' Behavioral Models based on Sessions Data

Vitaly Zabiniako, Toms Rožkalns, Erika Nazaruka*, and Jurijs Kornienko

ABC software Ltd., 51a Tallinas St., Riga, LV-1012, Latvia

vitalijs.zabinako@abcsoftware.lv, toms.rozkalns@abcsoftware.lv,
erika.nazaruka@abcsoftware.lv, jurijs.kornijenko@abcsoftware.lv

Abstract. Analysis of user behavior models based on user session data can be conducted using clustering (or community detecting) algorithms that do not require a predefined number of clusters. An unknown number and quality of potential behavioral models, non-efficient utilization of memory and processing units, and a large amount of data are the main difficulties developers could meet. Besides, the suitability of clustering algorithms for the task highly depends on the characteristics of datasets and still requires additional research. The first step and the goal of this research is to analyze the capabilities of the clustering algorithms in order to determine more promising ones and techniques able to minimize computing resources while solving the task. During the research, the properties of algorithms were analyzed through a literature review and also experimentally. It was found that, although the algorithm's suitability is theoretically proved, experiments could show unsatisfactory results. Therefore, a certain trade-off analysis and problem-specific modifications of original processing must be done. As a result, the clickstream clustering method of the Louvain clustering algorithm and the modified Longest Common Subsequence algorithm showed more appropriate results in detecting the well-clustered user behavior models based on user sessions data. Thus, these algorithms are suitable for the further development of the detecting method.

Keywords: User Behavior Model, Session Similarity, Community Detection, Parallel Algorithms, Distributed Algorithms, Scalability.

1 Introduction

Digital transformation in different business fields opens new analytical opportunities. One of the beneficiaries are customer-related processes. Modern information systems store large datasets of

* Corresponding author

Reference: V. Zabiniako, T. Rožkalns, E. Nazaruka, and J. Kornienko, "Analysis of Algorithms for Detecting Users' Behavioral Models based on Sessions Data," Complex Systems Informatics and Modeling Quarterly, CSIMQ, no. 41, pp. 55–79, 2024. Available: https://doi.org/10.7250/csimq.2024-41.04

Additional information. Author ORCID iD: V. Zabiniako – https://orcid.org/0000-0002-1307-1815, E. Nazaruka – https://orcid.org/0000-0002-1731-989X, and J. Kornienko – https://orcid.org/0000-0002-2845-9820. PII S225599222400226X. Received: 16 October 2024. Accepted: 27 December 2024. Available 31 December 2024.

recordings of users' activities during user-system interaction sessions. These datasets may be used for further analysis of users' behavior, for instance, for searching similarities in users' session activities in order to group them and propose a behavioral model for each defined group. Further, these models can be applied for analysis of anomalies in behavior of users belonging to certain groups or to provision better services for a certain group of users.

However, there are several issues related to the processing of this kind of data from a technical point of view. The first general issue is that data in the datasets usually are anonymized and contain only sequences of users' activities performed at some time points. This means that *the number of potential behavioral models is unknown*. The second issue is that the number of logged activities is quite large, and *processing would require high computational resources* that are additional costs for the enterprises. A good illustration is the example considered in this article, where real log data from an existing information system can constitute more than one million users' sessions per day. Therefore, processing these data for the indicated purpose has its specificity.

From the scientific point of view, there is no "silver-bullet" method for processing users' session data in order to determine user behavior models (as explained in Section 3). The existing methods and approaches are very task-specific, and selection of the appropriate techniques for processing is essential. Thus, the research question stated is what method and, correspondingly, implementation algorithms can be helpful in analysis of session activities data and grouping them accordingly to the dynamically determined "user behavioral models", i.e., clustering them taking into account the potential volume of data, unknown number of clusters, and necessity to make the analysis of data "efficient" and to assure that received results are "adequate" to the reality. The term "efficient" here means less processing time within the less expensive computational configurations. The "adequacy" is based on the expert's assessment of quality of determined clusters, i.e., whether they are meaningful or useless.

The goal of the research presented in this paper is to find out what clustering (or community detection) algorithms currently used for processing large, anonymized datasets are more promising for overcoming the indicated issues in satisfying the needs mentioned.

In order to achieve the goal, we have followed the Design Science Research Methodology (DSRM) suggested by Hevner et al. [1] especially for the creation and evaluation of digital artifacts designed to solve real-world problems (details are provided in Section 2 of this article). We have determined three iterations, and the results presented here correspond to the first iteration. By performing the review of existing related work and practical experiments on the selected algorithms and experimental datasets, we have determined that the most promising combination of algorithms is our modification of the *Longest Common Subsequence* (LCS) algorithm for calculating session similarities and the *Louvain* for clustering user sessions by similarity based on common activity sequences. Although the execution time here is 4 times slower than in case of agglomerative clustering, the machine learning (ML) model obtained satisfies the requirements that come from the dataset characteristics and the domain experts.

The article is organized as follows. Section 2 contains explanations of the research framework and procedure. Section 3 represents the literature review results and discussion on the selection of algorithms for experimental activities. Section 4 contains a presentation, analysis, and discussion of experimental results, and Section 5 concludes the paper with the main findings and intended further research steps.

2 Research Framework

2.1 Overall Process and First Iteration

As mentioned in the introduction, the research methodology used in this project is Desing Science, suggested by Hevner et al. [1], which has got quite large recognition in the researchers' communities. We have adopted this methodology for the investigation of the stated problem.

The general framework of the research project is illustrated in Figure 1. The current implementation of grouping users based on their real behavior in the system uses a predefined number of groups (clusters) suggested by domain experts [2]. Therefore, the constructed behavioral models of users are quite general and make analysis of a user behavior inaccurate. Moreover, the current solution is not able to process large sets (approximately one million of new records of user sessions per day) of application-level data in a reasonable manner for small and medium size enterprises (SME). The final target of the research is a Machine Learning (ML) method developed for application-level data analysis in order to determine *real* behavioral models of users based on sequences of their session activities. The models determined must be useful for further analysis of users' behavior. The method developed must be applicable for everyday analysis of new data of approximately one million records of user sessions per day (as mentioned) within the least computational resources possible without losing quality of the result. This method is planned to be used as a part of the monitoring systems of users' interaction with the information system.

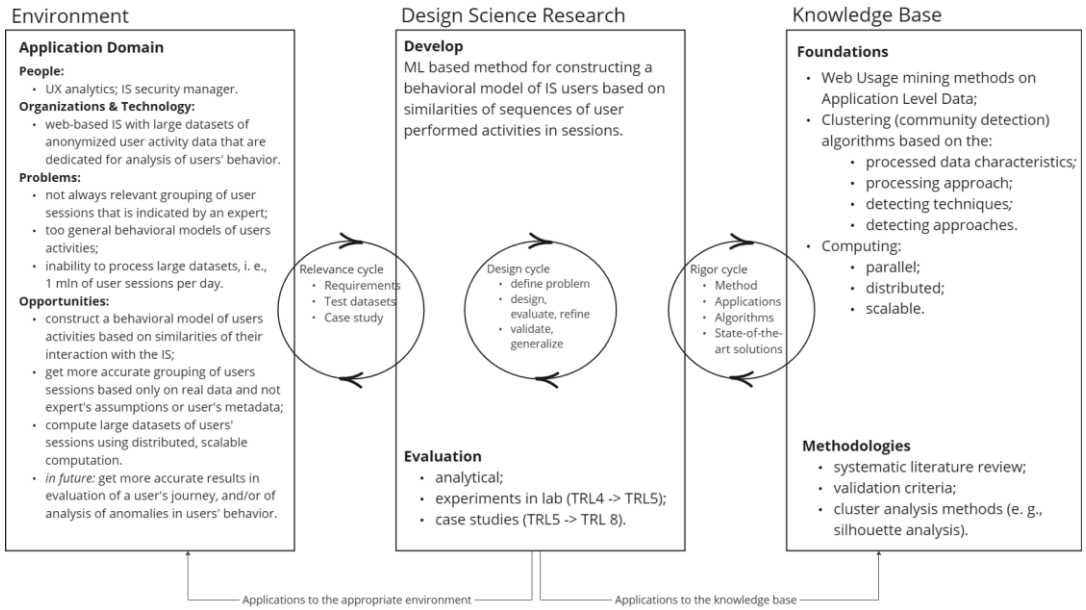


Figure 1. The general framework of the research project (TRL – technology readiness level)

The preliminary analysis of the existing knowledge base resulted in the conclusion that the required method should be a part of the web usage mining methods on application-level data and should leverage existing implementations of clustering methods. The implementations should show satisfactory results in terms of data processing time, quality of communities, and reasonable usage of computational resources. Therefore, scalable, parallel, and distributed computing is at the general research focus, too.

The relevance of the research results is to be evaluated analytically, as well as experimentally in the lab and in case studies achieving in such a way the technology readiness level (TRL) of the method from TRL4 to TRL5 to TRL8. Evaluation results are planned to be achieved as a result of the literature review and analysis of the results obtained experimentally by using the developed validation criteria, e.g., Silhouette[†] analysis for quality of grouping.

As a final result of the research project, we plan to extend the knowledge base with the new ML method for detecting users' behavioral models, knowledge about the application of the existing clustering algorithms in distributed scalable environments, and knowledge about implementations of the state-of-the-art clustering algorithms for distributed and scalable computation.

The entire research process planned consists of three iterations. The objective of the first research iteration (what results are presented here) is to determine integral parts of the method,

[†] https://doi.org/10.1007/978-3-030-52348-0_2

i.e., algorithms that could be beneficial for clustering large, preprocessed datasets efficiently in terms of time, computational resources used, and quality of clusters in case if an expected number of clusters is unknown (Figure 2). The task of the first iteration is to evaluate the existing implementations of the selected algorithms on the *synthetic data* and *anonymized and pseudonymized datasets from non-production environments* in order to combine the most beneficial algorithms within the ML method in the next iterations. The first iteration findings are to be applied in the second iteration to check the algorithm properties in distributed scalable environments on full datasets and to develop a machine learning (ML) based method for efficient clustering of new daily collections of user sessions according to detected behavioral models of IS users. The third iteration is dedicated for development of a prototype that implements the methods tested in the lab and validation of it in the environment similar to the real one. In this article we are focusing only on the results of the first iteration.

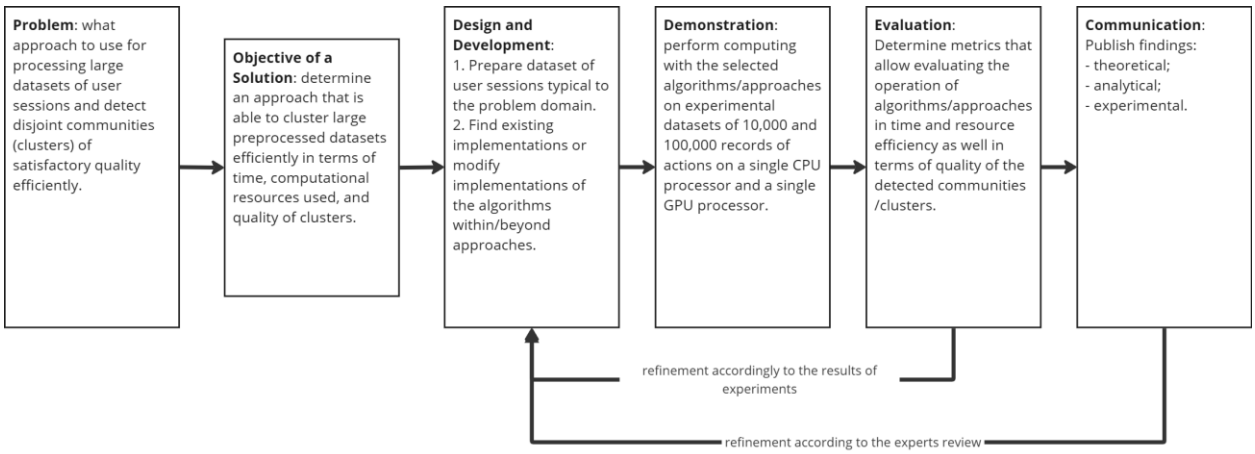


Figure 2. The 1st iteration of the research project

2.2 Dataset Requirements to Selection of Algorithms for Potential Solution

Requirements for the potential solution, i.e., combination of algorithms, follow from the goal and characteristics of the datasets. The dataset we operate with is a set of records of actions performed by an anonymized user during its interaction session (30 minutes) with the system. We are interested in searching for similarities of sequences of user session actions and cluster them to determine user behavioral models. Since the number of clusters is unknown, i.e., we are unaware what is the expected number of well-formed clusters, we are looking for a method or an algorithm that is able to detect a minimal number of clusters of the best possible quality for the given datasets. This problem is similar to the community detection problem in a network, where it is necessary to "identify all communities in a network" [3]. A community "can be defined as a group of similar sets of entities, which are closer to each other in comparison to other entities of the dataset" [4]. In other words, community detection is a process of searching for a set of similar entities [4]. However, a community in our case *does not* include any content-related information, just user activities done within a session. Hereinafter, when we refer to "a cluster" or "a community", we do refer to one and the same concept.

Surveys performed in recent years indicate that a large amount of recent research in the field focuses on querying communities on heterogeneous information networks [3], [5], [6]. However, we treat all the "nodes" as the same type and are not interested in additional semantic information.

The community structure also can be quite complex, being disjoint, overlapping, hierarchical, and local [5]. In the given research, our aim is to detect *disjoint* communities without hierarchical characteristics and localization.

As indicated in [7], community detection can be implemented according to the time slices or incrementally. The idea is that such implementations allow considering the dynamic nature of network data. Although, in our research project, real-world data are increasing per 1 million

sessions per day, and, in reality, we will face the dynamic nature of communities, in the first research iteration, we treat *data as static* for each separate dataset and assume the upper limit as 100,000 user actions.

3 Selection of Algorithms For Detecting Users' Behavioral Models

The first step to be done according to the *design and development phase* (Figure 2) is to get an answer to the question "What methods and algorithms are used for detecting users' behavioral models in other research?". Therefore, the results of the review of state-of-the-art methods and algorithms mentioned as applicable for the same or similar problem are presented in this section.

3.1 Classifications of Clustering Algorithms

In essence, a method for detecting behavioral models based on user session similarities is a task of community detection. Community detection or *network* structure detection is a process of division of nodes "in a network into groups where the nodes in a group are densely connected whereas nodes in different groups are sparsely linked" [6]. In other words, it is mining a network or graph structure. Therefore, this opens wide application of community detection algorithms that may be differently classified.

For instance, the source [6] suggests classification *based on the processed data characteristics*:

- The algorithms *without node semantics or attributes* include hierarchical clustering, modularity optimization, spectral clustering, and statistical inference.
- The *advanced semantic-based* algorithms are multi-objective, matrix factorization, and Bayesian models as well as deep learning.
- The algorithms that *depend on the models* used, namely, learning-based methods, e.g., model-based generative methods and the graph convolutional network that uses a convolutional neural network.

Another classification is suggested in [8], where algorithms are divided by the *processing approach* into *agglomerative* and *local movers*. As it is clarified by the authors, the agglomerative algorithms merge pairs of communities until expected community quality is achieved. In contrast, local movers try to achieve the expected quality by moving a node to the community of a neighbor. Label propagation algorithms belong to the local node movers.

Another algorithm classification that uses the *detecting techniques* as a splitting criterion indicates *divisive*, *agglomerative*, and *optimization-based* [9] classes of algorithms. Thus, divisive algorithms go from a complete network to the smaller communities by sequential division of the network. The agglomerative algorithms consider each node as a unique community at the beginning of the processing and iteratively combine nodes into a larger community. The optimization algorithms use an optimization *function* to find the optimal solution for a pre-specified objective.

Another classification of community detection algorithms *based on the detecting approaches* used indicates three basic approaches [10]:

1) *the top-down approach* that starts from the graph representing the entire network and then applies its division to form communities;
2) *the bottom-up approach* that uses local structures and expands them to form communities;
3) *the data-structure-based approach* that converts the entire network into a data structure and then processes it to detected communities.

As the authors in [10] note, although the top-down approaches are well-adopted for the community detection task, they could produce too high overlapping among communities and delay processing data. In our research, we are not interested in detecting overlapping communities, but quite the opposite – we are interested in detecting *disjoint* communities. In their turn, bottom-up algorithms have a linear complexity (for optimization and label-propagation techniques) [9], [10]

that makes them preferable for community detection, but they could fail in detection of very small (even well-defined) communities due to the expansion method used. For the given problem statement, this weakness could be considered as unessential. The agglomerative algorithms [10] and genetic algorithms [11] are representatives of the data-structure-based approach, and usually a tree form is selected there. The potential problem of these algorithms is high computation costs that usually is solved by carefully implemented parallelism.

The source [4] indicates more detailed classification of the algorithms *based on the detecting approaches* such as: traditional, statistical, node betweenness centrality based, spectral partition, tree structure, hierarchical, graph partition, and cohesive sub-graphs. There are also algorithms that calculate the *edge betweenness centrality* and try to achieve the better values of modularity *Q* [12].

Therefore, after summarizing published classifications (Figure 3) we can conclude, firstly, that at present there is a lack of the only "valid" classification, and, secondly, classification criteria are just properties of existing algorithms which they may share.

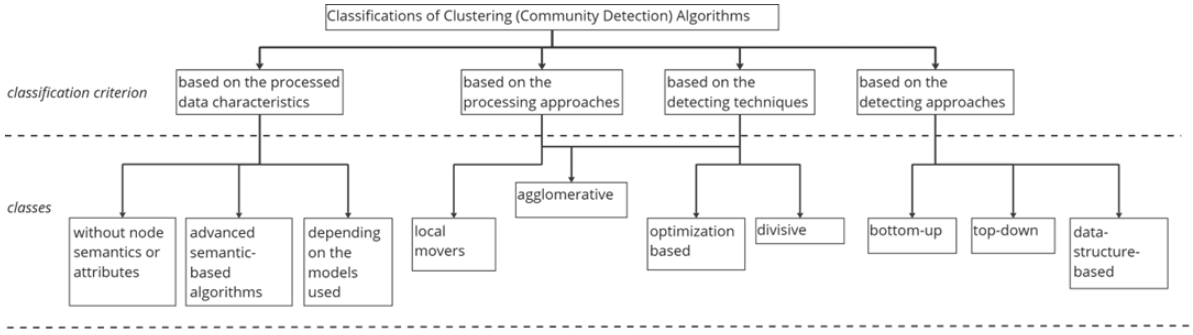


Figure 3. Classification criteria for clustering algorithms

In this research, we do focus on the group of algorithms that *process data without semantics* (or attributes) of nodes and edges, since, first, while analyzing users' session data we do not consider any additional semantics of nodes, and second, learning-based methods are less suitable to our approach due to a lack of heuristics and statistics as well as computational resources for learning models. Then, we would focus on algorithms that calculate *node betweenness centrality*, as it is in the case of agglomerative algorithms, or *similarity of nodes*, as it can be in the case of divisive or optimization-based algorithms. The *edge-betweenness algorithms* are excluded from the research since they consider edge weights to be distances, not connection strengths, which contradicts with the meaning of weights in our datasets. Thus, we do not limit the processing approaches as well as detecting techniques and approaches in the algorithms under consideration if they belong to the algorithms that process data without semantics of attributes.

3.2 Existing Clustering Methods and Algorithms: Results of Related Work Review

By formulating properties to methods/algorithms we are searching for, the following research questions are stated:

- What activities and corresponding algorithms are used in the similar method implementation? Are there publicly available versions of the algorithms used?
- Is this method/algorithm scalable? Is its implementation with parallelization available?
- What experimental results are achieved? Are they useful for the given research?

3.2.1 Literature Search and Selection

The search for the information sources was started from the IEEE Xplore where keywords used are *("All Metadata":"community detect*") AND ("All Metadata":application* data) NOT ("All*

Metadata":overlap)*, and continued with the Google Scholar for discovering additional sources about specific aspects. In this research, we have searched for results published from 2014 up to 2024 in English. The sources found were first filtered by the relevance of their titles and abstracts, and then by the relevancy and quality of the content. Since the goal of this research iteration is to analyze algorithms that detect disjoint communities, we excluded sources focused on overlap, hierarchies, and localization of communities. Besides, we do not consider processing dynamic graphs and networks but put our focus on static ones. The query returned 553 relevant information sources available in IEEE Xplore that include early access articles – 1, books – 2, magazines – 9, journal articles – 171, and conference papers – 48. After the first round of filtering by analyzing the overall relevance of the content, the number of sources decreased to 81, where 1 is a book, 2 – magazine articles, 30 - journal articles, and 48 – conference research papers. Finally, only 15 sources that satisfied the requirements stated in Section 3.1 were selected for further analysis. The initial number of sources was extended by searching in Google Scholar for relevant newer sources or those of providing more details for further analysis.

3.2.2 Methods and Activities for Clustering Data by Node Similarity

The results of searching for facts to answer the question about methods and activities used are illustrated in Table 1. The analysis of the literature showed that the 1st step in methods that use weighted graphs is the calculation of similarity between nodes in the network, i.e., between vertices in the graph. The meaning of *similarity* differs in the approaches since it depends on the nature of the data and the task set. For instance, community discovery based on user behavior similarity may include such sub-activities as pre-processing text data, mining the topic feature according to the Latent Dirichlet allocation (LDA) [13], extracting the user behavior information and the user topic feature words, and increasing the attribute characteristics of the user behavior similarity [14]. Besides, similarity can be used both for determining only the fact that a relation between nodes does exist and also – for determining the weight of this relation [15]. In the latter case, one speaks about *weighted networks (graphs)* where each weight "encrypts" semantic that is very important for grouping. Another activity mentioned as the 1st one after data pre-processing and transformation is the calculation of Jaccard Distances for different types of interactions [16]. Since, in the given research context, we do not have any type of interaction between sessions, this activity, in its straight meaning, is not applicable. However, the Jaccard distances can be calculated for action sequence alignment, where the calculation will be focused not on the interaction but on the same concept of similarity as in A1. Additionally, the final activity mentioned is the exact clustering or community detection that could be performed by applying different modifications of existing clustering techniques.

Table 1. Activities frequently used for community detection

Activity	Depends on activity	Description
A1. Calculate node similarity		Node similarity is calculated after pre-processing user or node data, e.g., it can consider content generated by user [14], connection [14] or social [15] relationship, behavior similarity, common neighbors [17], node attributes [18], clickstream [19], sequence alignment [20] and so on.
A2. Calculate Jaccard Distances for different types of interaction		After the initiation of Jaccard Distance, the weight of edges is calculated for different types of interaction with a set of star graphs and initial values are correspondingly updated [16].
A3. Cluster / detect communities	A1, A2	Communities are detected by using one or another clustering algorithm [14], [15], [16], [17], [18], [19], [20],[16] that gives more satisfactory results. The satisfaction of the results is determined by an expert using closeness to the expected values of selected metrics.

Thus, for the given problem, we can determine two main suitable activities: node similarity calculation and clustering in weighted data structure. Since the result of similarity calculations in most cases is represented as a mathematical weighted graph, we focus on different kinds of graph-based clustering.

3.2.3 Algorithms for Calculating Node Similarity

The formal definition of "similarity" is given by Lin [21] and incorporates three intuitive definitions based on two phenomenon comparison: commonality, differences, and identity. At present, plenty of node similarity determination algorithms and similarity measures exist [22], e.g., algorithms can be based on profile matching, link prediction, sequence alignment, classification, and information retrieval. In networks, similarity metrics can be node-based, neighbor-based, path-based, and random walk based [22]. Since the experimental datasets do not contain structural information about relations with node neighbors or paths among nodes but rather reflect the internal dynamic aspect of each node, we cannot use algorithms that calculate the "external" similarity values between every pair of nodes like, for instance, the SimRank [23]. In the given case, similarity determination is based on node features information that is a certain sequence of user actions within a session.

Node-based similarity algorithms can range from a general, where a classical problem of vector (string) similarity is considered, to a more specific one, where node attributes are used. Since each record in the given datasets is a single user action within a session, then each session can be considered as a node whose internal structure can be expressed as a vector of a sequence of user actions. This means that the task is reduced to calculating similarity of sequences.

Analysis of the publications found (Table 1) concluded with the determination of two methods applied for a similar context. They are the Clickstream clustering [19], [24] and the Agglomerative clustering [20] methods. Algorithms applied for calculating a similarity between action sequences in the Clickstream and the Agglomerative methods are the *Longest Common Subsequence* (LCS) and *Matching ratio calculation,* correspondingly. The LCS is mentioned in the methods for detecting clickstream-based clusters in user behavior datasets. The Matching ratio calculation uses sequence alignment for clustering. The applicability of these methods for the given problem has been compared under certain conditions (Section 4.3).

3.2.4 Algorithms for Clustering or Detecting Communities

The overview of the selected literature sources allows us to identify the most promising clustering algorithms that we decided to use for further experimentation. The algorithms were grouped by the detecting approaches and detecting techniques (Section 3.1) as follows.

Data-structure-based algorithms presented in the literature sources apply mostly agglomerative and optimization detection techniques (described in more detail below):

- *Agglomerative* detecting is represented by the Girvan-Newman algorithm and its derivatives, UBSM algorithm, CNM algorithm, CLU_TBB, and DBSCAN.
- *Optimization* detection is represented by MENSGA, HGT (hybrid genetic algorithm and tabu-search) and Distributed Genetic Algorithm.

Agglomerative detecting *per se* has advantages in calculating speed and the ability to deal with large datasets. Therefore, this group of algorithms raises high interest in similar research work.

Girvan-Newman algorithm (proposed in 2002 and 2004) is based on *node betweenness centrality* [14]. It was the first that introduced the community merging until the module degree Q (the modularity of modules). Modularity is a function to quantify the quality of the detected community structure in binary networks [12], [15]. In the general case, this algorithm calculates *betweenness centrality* and iteratively eliminates edges that "have the highest number of shortest paths between nodes passing through them" [25]. There are publicly available implementations of

this algorithm[‡,§], but we could not find any implementation that uses parallelism. The operation of the algorithm is based on *betweenness centrality* and does not apply *similarity*. For the given research, it would require significant modifications without visible benefits.

The UBSM algorithm [14] is based on *node similarity*. The similarity between nodes is introduced and improved, including user-generated content. The modularity function is redefined. We have not found any publicly available implementations of this algorithm. Besides, the algorithm proposed uses very specific similarity calculation. Thus, it was not further investigated.

A globally greedy hierarchical agglomerative method CNM [8] also uses the concept of modularity. This is a hierarchical algorithm that was validated by a dataset of 400,000 vertices and more than 2.5 million edges. The maximum value of modularity achieved is 0.745 [26] that is considered a satisfactory result. The authors indicate that similar sequential multilevel calculations may combine agglomeration with refinement techniques. However, we have not found any publicly available implementation of this algorithm and excluded it from further investigation.

The CLU_TBB algorithm suggested by Fagginger, Auer, and Bisseling in [27] is an agglomerative modularity maximization algorithm [8] with an implementation for both the GPU (using NVIDIA CUDA) and the CPU (using Intel TBB). The authors note that agglomerative clustering advantages are achieving the high modularity of clustering in a small amount of time. During the authors' experiments, the CLU_TBB showed better results (in terms of speed) than the parallel Louvain [8]. However, another algorithm of a similar type called CEL showed worse results than the parallel Louvain [8] under the same conditions. Therefore, we can conclude that the important point for this type of algorithm is also implementation of parallelism. However, we did not find source code of the CLU_TBB; just pseudocode is presented by the authors.

And the last promising representative of the agglomerative algorithms found is DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [28]. The idea is that the network members are classified to core members of the groups and less active members of the groups represent borders of them. The third layer represents outliers or noise – that is, members without influence in the group. The number of clusters will be equal to the number of core elements. For clustering, DBSCAN applies principles of Girvan-Newman clustering algorithm. Pseudocode for optimized version is presented in [29], but only the source code of the *unoptimized* DBSCAN is available in C[**] and Python[††]. The parallel version of the algorithm also exists [30] and is published[‡‡]. However, the algorithm's available implementations do not apply edge weights while propagating labels and would require additional modifications and research. Therefore, this algorithm is excluded from further experiments.

While searching additional literature, we have found another algorithm of this type whose implementation considers similarity values – the Agglomerative Hierarchical Clustering [20] – that was applied for a similar task, i.e., to clustering based on the sequence alignment. Since this algorithm also considers similarities of nodes, i.e., weights, we have found its implementation and taken it for further research (Section 4, Table 2).

Optimization detection in data-structure-based algorithms is represented by genetic structures. MENSGA [31], HGT (hybrid genetic algorithm and tabu-search) [12], and Distributed Genetic Algorithm [11] are so-called genetic algorithms. The basic idea of genetic algorithms is in the optimization of the modularity function for populations created on the basis of node similarity. The first critical component is a principle used for coding population chromosomes. Within the given problem, discovering this principle for a sequence of user actions could be challenging, as well as applying the suitable mechanism for generating the initial population. Moreover, there are many design parameters such as population size, total number of generations, probabilities of application of crossover, and mutation operators. All these factors increase the complexity of

[‡] https://memgraph.github.io/networkx-guide/algorithms/community-detection/girvan-newman/
[§] https://neo4j.com/docs/graph-data-science/current/algorithms/modularity-optimization/
[**] https://github.com/gyaikhom/dbscan in C
[††] https://github.com/choffstein/dbscan in Python
[‡‡] https://pypi.org/project/dbscan/ (parallel version)

adequately mapping real semantics of data to population chromosomes to the required task. Besides, the presented genetic algorithms show similar or a slightly worse results than the Louvain algorithm [12] and were not verified on large datasets. In turn, the Distributed Genetic Algorithm was developed using the original distributed encoding technique to decrease computation costs. The experiments showed that approximately 30 million triplets (data structure elements) can be clustered by 47 min on 6 machines with an average number of threads of 40 per machine node. The results achieved prove that parallel and distributed computation is possible for genetic algorithms, but the results received are not overwhelmingly better than those shown by other algorithms. Respecting all the mentioned facts, the genetic optimization algorithms are excluded from further investigation.

Bottom-up algorithms for graph-based representations apply the *optimization* techniques and are represented by the Louvain algorithm (a.k.a. Fast Unfolding algorithm) and Weighted Label Propagation Algorithms (WLPA) with the optimization.

The Louvain algorithm is a locally greedy hierarchical (bottom-up) clustering algorithm [8], [14] based on the *modularity* function proposed by Newman as the objective function. The newer algorithm's versions not only use node labeling but also take weights into account. Thus, the Louvain can be useful to graphs with *calculated similarity*. Authors in [8] suggested parallel Louvain Method with Refinement. The algorithm has high computational efficiency and is appropriate for processing large-scale networks [14]. The steps of the algorithm are simple and intuitive, easy to implement [14], that is proved by multiple available implementations in code. Parallel implementations of the algorithm also are available[§§] [8], [32]. Multiple authors [4], [14], [8], and [32] indicate the Louvain algorithm may be most valuable for the researchers working on the community detecting task due to its ability to provide relatively good performance even when the number of nodes and edges increases. Therefore, this algorithm is taken for further research.

The Weighted Label Propagation Algorithms (WLPA) with the optimization [33] implement the idea to group nodes by labeling them similarly around the node with the strongest relations with its adjacent nodes. The optimization should find a balance between performance and accuracy that is not presented in different earlier versions of WLPA [33]. The improvement can be made by setting labels initially in accordance with the node weights. Then, a vertex with a larger weight would tend to have strong connections to propagate labels. The authors have presented just a pseudocode of the optimized version, and the availability of a parallel version is unknown. However, available implementations of Label Propagation Algorithms are not suitable for the given task since does not consider edge weights. Therefore, this type of algorithm was excluded.

Top-down algorithms for graph-based representations are represented by MeTiS, an algorithm that uses *divisive* detecting technique. The MeTiS algorithm is a graph-partitioning algorithm that splits a graph into sub-graphs with a high density of relations inside each subgraph and a low density among subgraphs. Because of this property, it is possible to process subgraphs in parallel. Sequential versions (e.g., [34]) and parallel implementations (e.g., ParMETIS[***]) are available. The ParMETIS extends the functionality of the sequential MeTiS by including routines specific for parallel computations [35] and large-scale numerical simulations [36]. However, the MeTiS and its parallel version require indicating the predefined number of clusters that contradicts the stated requirements. Therefore, this algorithm also was excluded from the further experiments.

4 Experimental Analysis of Selected Methods and Algorithms

4.1 Design of Experiments and Evaluation of Results

Based on the analysis of literature and identified activities (Table 1), as well as limitations to the task – to use unsupervised learning, we have concluded that the experimental part is to consist of

[§§] https://jp-tgdocs.netlify.app/graph-ml/current/community-algorithms/louvain-method-with-parallelism-and-refinement
[***] https://github.com/KarypisLab/ParMETIS

three steps – data pre-processing, pattern discovery, and pattern analysis (Figure 4), accordingly to the web usage mining tasks.

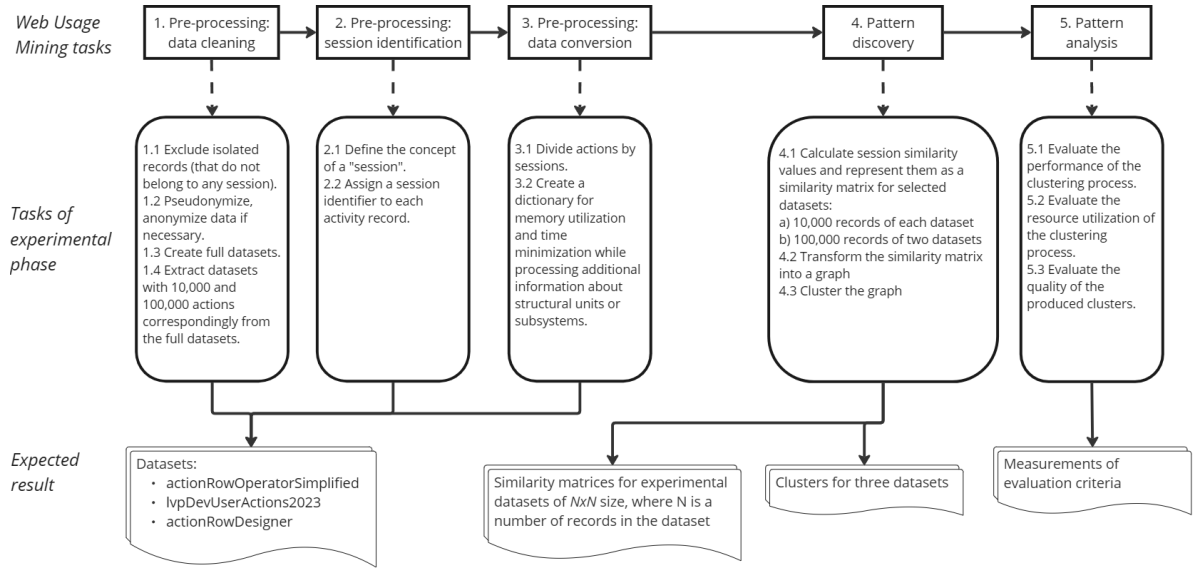


Figure 4. Task flow of experimental phase

Data pre-processing includes cleaning, session identification, and conversion (Section 4.2). The three main activities performed after pre-processing data are a calculation of session similarities, forming the similarity matrix, with its further transformation into a graph in order to select the similarity calculation algorithm (Section 4.3), and clustering the graph with its further evaluation in order to select the more suitable clustering algorithm (section 4.4).

In order to evaluate suitability of the candidate algorithms, we have planned the following set of experiments for two methods mentioned in the sources [14] and [20], i.e., the Clickstream (where MeTiS is substituted with the Louvain) and the Agglomerative sequence alignment, correspondingly:

1. Calculate session similarities and create the corresponding similarity matrix for a *synthetic* dataset of 10,000 user actions using *Longest Common Subsequence* (LCS) and *Matching ratio calculation*. For each algorithm, evaluate the performance without and with performance improvement techniques, if applicable. Measure the execution time in seconds. Transformation of the similarity matric into a graph or input format required to the clustering algorithm is not evaluated since this operation is sequential and cannot drastically affect the performance and resources utilized.
2. Evaluate the performance and quality of clustering for selected algorithms: the Clickstream with the Louvain and the Agglomerative Hierarchical Clustering for datasets of 10,000 and 100,000 actions without time. Dataset names are indicated in Figure 4. Note that the expected result is the minimal number of well-formed clusters. Accordingly, as mentioned in [8], cluster sizes also can be taken for clustering quality evaluation.
3. For the most promising method, consider how the performance and quality of clustering change if time is added to the datasets.

Evaluation of the performance of algorithms is based on the calculation of the time ratio per action for the mentioned datasets. Time ratios, as indicated in [8], [4], and [10], allow making the prediction on execution time required for other sizes of dataset objects or execution threads.

Evaluation of the quality of clustering is not so straightforward. In disjoint community detection, the goal is to group the nodes in the network (or graph) in a way that “maximizes the similarity of the nodes within one cluster while maintaining the dissimilarity between different clusters” [37]. As the authors mention, one of the main challenges is to determine the number of clusters that

achieve this goal beforehand in the case of clustering algorithms with a pre-defined number of clusters. However, it is also true for cases when one needs to determine whether the number of clusters determined by an algorithm is satisfactory. In other words, when it is necessary to see are the clusters determined optimal for the given dataset. Therefore, it is necessary to have proper quantitative measurements for formal evaluation of the result when expert's pre-defined structure (the ground truth) is unavailable.

Clustering Validity Indices (CVIs) can be internal and external [4]. The external indices compare the *a priori* known results with those coming from the model (actual results); in turn, the internal ones are used to measure the quality of actual clusters without having a priori known truth [38], reflecting their compactness, connectedness, and separation. The metadata communities usually are considered as the *ground-truth* for the external indices such as Normalized Mutual Information (NMI), Recall score, Precision score, F1-measure, etc. [39]. Therefore, since the ground truth labels are assumed to be unknown in our case, the evaluation can only be performed using the model results itself, i.e., by the internal measurements. The optimal number of clusters can be determined based on an internal CVI and further analyzed by an expert (knowing the general requirements of the result).

There are many widely used internal indices for searching an optimal number of clusters, for example, Silhouette index, Davies-Bouldin index, Calinski-Harabasz index, Dunn index, RMSSTD index, and so on [38]. The question is how to understand what validation index is most suitable for the given datasets if it is known that they can even contradict each other [40] or show incorrect results [38], [40], [41]. As the authors mentioned in [40], there are several attempts to provide a general framework for selecting one or another index. However, they are not yet completely validated to be considered for industrial research. We analyzed our datasets and concluded that the Silhouette index is appropriate. The main advantage of its using is a low computational complexity and simple interpretation rules [41].

The Silhouette index (Silhouette coefficient) proposed by Rousseeuw [42] is a technique that represents each cluster by a "silhouette", which is based on comparison of clusters compactness and separation. When a cluster contains just one element, the coefficient is assumed to be equal to zero. The general evaluation of the coefficient lies in [-1, 1]. The silhouette of a cluster is a plot of coefficients of its elements ranked in a decreasing order [42]. The result of clustering is evaluated as better, as higher are the coefficients of clusters (or the average silhouette width).

Besides, it is possible to evaluate clustering by relative metrics: a number of generated clusters, a minimal cluster size, a maximal cluster size, an average cluster size, a mean cluster size, as well as a number of clusters with one element (i.e., elements that cannot fit any generated clusters). Besides, the percentage distribution of coverage of clustering size types [43] allows evaluating empirical form for the clustering.

Therefore, metrics to be used for evaluation of clustering algorithms are the following:

- Execution time, in seconds.
- Execution time ratio, in seconds per session.
- Number of generated clusters in units.
- Minimal cluster size, in number of elements.
- Maximal cluster size, in number of elements.
- Average cluster size, in number of elements.
- Mean cluster size, in number of elements.
- Number of clusters with one element in units.
- Distribution of coverage of cluster size types in percents.
- Internal clustering using the Silhouette method in a range [-1, 1].
- Average Silhouette width in percents.
- Average positive Silhouette width in percents.

The last metric applies the modified formula of the average Silhouette width by excluding clusters with negative coefficients.

4.2 Data Pre-processing

Datasets for experiments include completely anonymized and depersonalized data on user actions similar to those from the real application areas of information systems. Data are taken from the auditing records collected in the development/testing environments and do not contain any data that can be used for real user identification. The pre-processing included data cleaning, session identification and data conversion steps (Figure 4). All actions of a user that occur within a half-hour interval are assumed to belong to one and the same session. Another limitation set to the datasets is a time unit between two actions in the session; it is assumed to be a second. In addition to the described pre-processing, the records that do not belong to any session were excluded from the dataset. The pseudonymization of data, where necessary, is done by substituting users' names with randomly selected international female names and by replacing titles of actions with other semantically close words or numerical identifiers (of integer type) from the developed dictionary in order to reduce the required memory resources.

For the experiments, the following pre-processed datasets are used (Table 2):

- **actionRowOperatorsSimplified**: It is a dataset that is collected while performing another project and is completely pseudonymized by name substitutions. The specificity of the data is that they simulate "data entry operators" performed actions in some IS. The number of records representing user actions after pre-processing is close to 3 million.
- **lvpDevUserActions2023**: It is a dataset including user data from the LVP development environment that are completely pseudonymized (can be considered as synthetic) and simulate actions done by real users of portal *latvija.lv*. The number of records representing user actions after pre-processing is less than 1 million. It is worth noting that there is a plan to publish this dataset as open data on *https://data.gov.lv/lv* in the future.
- **actionRowDesigners**: It is a dataset that is collected while performing another project and simulates the use of "AutoCAD" software. Data includes the actions of engineering system designers and the execution time of the actions. This is the smallest experimental dataset of approximately 80 thousand action records.

The examples of JSON (JavaScript Object Notation) records in each dataset are illustrated in Figure 5. As illustrated, action names (*task_name, ActionName*) and user identifiers (*userid, UserId*) are pseudonymized.

Table 2. Datasets characteristics after pre-processing

Nr	Id	Number of records	Number of unique users	Number of sessions	Number of unique actions	Number of sub-systems	Number of structural units
1	actionRowOperatorsSimplified	2,775,493	488	173,855	169	2	45
2	lvpDevUserActions2023	900,545	39,443	110,081	1,318	1	3
3	actionRowDesigners	78,398	23	749	257	-	-

Figure 5. Examples of dataset records

4.3 Experimental Evaluation of Similarity Calculating Algorithms

Since the research task is to determine communities in datasets and it is mostly based on the analysis of interaction density within a group and outside a group, the interaction density can be measured using *similarities* (or *distance between entities*) [4]. As discussed in section 3.2.2, we have selected two algorithms – *Longest Common Subsequence* (LCS) and *Matching ratio calculation* (MRC).

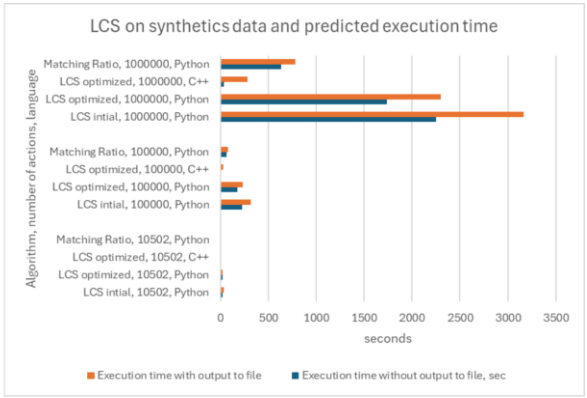
Both algorithms are checked on the synthetic data in order to clarify their performance. The synthetic data were generated and contained only action sequences; overall, 10,502 actions and 1,000 sessions. The LCS algorithm was executed in the environment with this configuration: CPU: Intel Core i7-6500U 2.5 GHz; RAM: 16 GB, Windows 10 x64; Python v. 3.10.6. The MRC was executed in the environment, where CPU: Apple M2; RAM: 16 GB, macOS: Sonoma 14.4.1; Python v. 3.10.12. The results are presented in Figure 6 (a) and explained below.

The first execution of the LCS algorithm implemented in Python without writing results onto the disk was 23.66 seconds or 0.00225 s/action, and with writing onto the disk 33.27 seconds, or 0.00371 sec/action (Figure 6 (a), 'LCS initial, 10502, Python'). In the case of the required 1 million actions, it would constitute almost 37.5 and 52.8 minutes, respectively, which is not satisfactory since the similarity must be re-calculated for every new 1 million sessions every day, where each action record is 255 bytes long. Therefore, the time required would be more than 160 hours. The time devoted to writing is just 28% of the overall time. Besides, memory constraints appeared to be critical. It was impossible to store the full similarity matrix because of the problems with matrix allocation in the memory. Therefore, it was decided to consider the conversion of data where similarity coefficients between sessions were coded by using 1 byte (char). The results are shown in Figure 6 (a), 'LCS optimized, 10502, Python'. The algorithm allows improving the execution time on approximately 20% that still is not satisfactorily.

The next improvement made was introducing the parallelism (multiple processors in the same machine) into the LCS algorithm and to change the implementation language to faster one, i.e., C++. This implementation showed satisfactory results (Figure 6 (a), 'LCS optimized, 10502, C++'). The execution time without writing onto the disk has been reduced on 98%, but with the writing – on 88%. Moreover, the execution itself constitutes just 12% of the overall time with writing.

Algoritm, Action number, language	Execution time without output to file, sec	Execution time with output to file, sec
LCS intial, 10502, Python	23.66	33.27
LCS optimized, 10502, Python	18.27	24.15
LCS optimized, 10502, C++	0.36	2.94
MRC initial, 10502, Python	6.68	8.23

(a)



(b)

Figure 6. Experiments with initial and improved implementations of the LCS algorithm

The Matching Ratio Calculation algorithm has shown better results in its original version on Python (Figure 6 (a), 'MRC initial, 10502, Python') but could not overcome the parallel LCS in C++. The parallel version of MRC was not found.

Therefore, the predicted time for 1 million synthetic actions would be approximately 4.5 hours for LCS optimized in C++ and 13 hours for the initial MRC.

4.4 Experimental Evaluation of Clustering Algorithms

After calculating the similarity matrix, we have experimented with the selected clustering methods, namely, the Clickstream Clustering (hereinafter, Clickstream) and the Agglomerative Sequence Alignment (hereinafter, AggSA) that use the Louvain and Agglomerative Hierarchical Clustering, correspondingly.

Before the experiment, it was assumed that only similarity values greater than 0.5 would be included in the similarity matrix to compose a graph for clustering. It allows for reducing the execution time significantly. Python programming language was selected as the implementation language due to existing implementations of the algorithms. The differences between the original methods and the experimental ones are explained in Table 3.

Environment configurations for execution on the CPU for algorithms were different:

- for the Clickstream — the CPU: Intel Core i7-6500U 2.5 GHz; RAM: 16 GB, Windows 10 x64; Python v. 3.10.6.
- for the AggSA — CPU: Apple M2; RAM: 16 GB, macOS: Sonoma 14.4.1; Python v. 3.10.12.

During the experiments, the following datasets were used: (1) for both the clickstream and the AggSA — *actionRowDesigners* of 10,000 and 78,398 actions without time, *actionRowOperatorsSimplified* of 10,000 and 100,000 actions without time, *lvpDevUserActions2023* of 10,000 and 100,000 actions without time; (2) for the clickstream only — *actionRowDesigners* of 10,000 actions with time, *actionRowOperatorsSimplified* of 10,000 actions with time, *lvpDevUserActions2023* of 10,000 actions with time in order to include difference in the similarity processing.

Table 3. Differences among original methods and the experimental ones

Method's element	Clickstream clustering original	Agglomerative clustering original	Experimental methods
Concept of a session	input: set of sessions, session: sequence of user actions and time	input: set of sessions, session: sequence of user actions and URL similarity	input: set of sessions, session: sequence of user actions within 30 min
Calculation of the similarity	LCS: session similarity calculation + importance	session similarity calculation, sequence matching and Matching ration calculation	*Clickstream*: the optimized LCS in C++ for session similarity calculation, where original similarity metrics are taken into account for sessions with the time attribute between actions, and the Jaccard index for sessions where time attribute is not considered. *AggSA*: Matching ration calculation in its initial version.
Similarity representation	All similarity values are included	All similarity values are included	Only similarity values greater than a threshold of 0.5 are included.
Clustering algorithm	MeTiS: the clustering algorithm that requires to know the number of clusters	Agglomerative Hierarchical Clustering	*Clickstream*: the method "*louvain_communities*" that implements the Louvain algorithm from the Python library "NetworkX"[†††] *AggSA*: the method "*weighted*" that implements "weighted/WPGMA" (Weighted Pair Group Method with Arithmetic Mean) algorithm from the clustering package *scipy.cluster* of the Python library "SciPy"[‡‡‡].
Clustering output	Clustering model	Clustering model	Clustering model

[†††] https://networkx.org

[‡‡‡] https://docs.scipy.org/doc/scipy/reference/cluster.html

Obtained experimental results (Table 4) and their analysis (Figure 7) illustrate that the execution time ratio increases drastically on CPU when the size of datasets increases, e.g., in case of *lvpDevUserActions2023* the increase is 78.6% – it is 0.0534 seconds per session for 10,000 actions and 1,213 sessions, and 0.2507 seconds per session in case of 100,000 actions and 12,087 sessions.

Table 4. Algorithm execution time for experimental datasets

Method	Dataset	Time	Number of actions	Number of sessions	Execution time on CPU, sec.	Execution time on CPU ratio, sec/session	Execution time on GPU, sec.	Execution time on GPU ratio, sec/session
AggSA	DES	no	10000	238	2	0.0084	n/a	n/a
Clickstream	DES	no	10000	176	15.4	0.0875	8.3	0.0472
Clickstream	DES	yes	10000	176	34.2	0.1943	9.2	0.0523
AggSA	OP	no	10000	634	6.4	0.0101	n/a	n/a
Clickstream	OP	no	10000	749	48.2	0.0644	14.3	0.0191
Clickstream	OP	yes	10000	749	154	0.2056	14.6	0.0195
AggSA	*DEV*	no	10000	1101	10.1	0.0092	n/a	n/a
Clickstream	*DEV*	no	10000	1213	64.8	0.0534	25.3	0.0209
Clickstream	*DEV*	yes	10000	1213	218	0.1797	24.8	0.0204
AggSA	DES	no	78398	749	50	0.0668	n/a	n/a
Clickstream	DES	no	78398	749	312	0.4166	15.2	0.0203
AggSA	OP	no	100000	6533	477	0.0730	n/a	n/a
Clickstream	OP	no	100000	7565	2538	0.3355	98.8	0.0131
AggSA	*DEV*	no	100000	11942	954.8	0.0800	n/a	n/a
Clickstream	*DEV*	no	100000	12087	3030	0.2507	138.7	0.0115

Note: DES – actionRowDesigners, OP – actionRowOperatorsSimplified, DEV - lvpDevUserActions2023

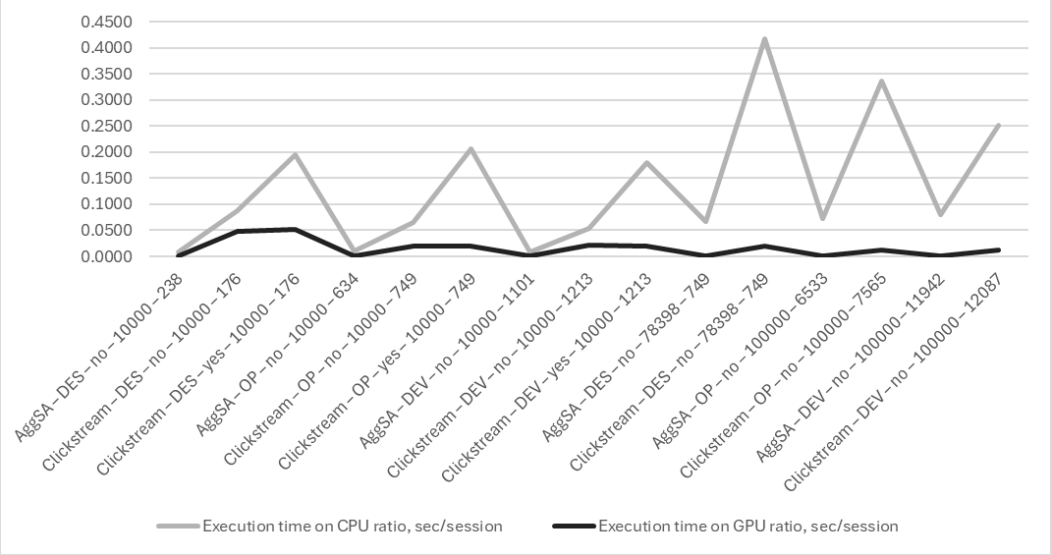


Figure 7. Execution time ratio for algorithms

Adding time units to the actions increased the execution time ratio by 3 times. This, tendency is the same for all experimental datasets executed by the Clickstream on the CPU.

Therefore, it was decided to check how clustering can be performed on the GPU. The experimental environment for execution – the GPU: RTX 4060, CPU: AMD Ryzen 5 5600x 6- core processor x 12; RAM: 32 GB, Ubuntu Linux 22.04, Python v. 3.10.12. Table 3 and Figure 7 illustrate that this tendency can be broken by executing the Clickstream method on the GPU, where for the same datasets, the ratio increase is just 2-10% for actions with time units depending on the

dataset structure, and it is less for the real-world data. Moreover, the overall execution time decreases significantly from 46% up to 90% for larger datasets.

In the case of the AggSA, it showed the best results in execution time on the CPU in comparison with the clickstream that corresponds to the notes from the related research papers.

Table 3 shows insignificant differences in session numbers in the datasets because of the different approaches used for session identification. However, we should note that it is not significant for validity of the achieved results.

The next measurements are related to the cluster sizes (Table 5) and their analysis brings us to the following considerations:

- The AggSA method gives the largest number of clusters in all cases, while the Clickstream produces the minimal number of clusters for datasets without time units in actions. At the same time, the number of clusters drastically increases when the time units are added.
- The clustering result has no direct relation to the number of sessions in the dataset.
- The Clickstream tries to avoid forming clusters of one object, if possible, while the AggSA tends to create many small clusters (Figure 8).

Table 5. Relative Cluster Metrics for Experimental Datasets

Method	Dataset	Time	Number of actions	Number of Sessions	Cluster size						
					Number	Min. size	Max. size	Avg. size	Standard deviation	Mean size	Number with one element
AggSA	DES	no	10000	238	34	1	19	7	4	7	3
Clickstream	DES	no	10000	176	5	17	64	35	21	20	0
Clickstream	DES	yes	10000	176	20	1	60	9	17	1	16
AggSA	OP	no	10000	634	129	1	38	5	6	2	58
Clickstream	OP	no	10000	749	4	2	278	187	111	234	0
Clickstream	OP	yes	10000	749	9	1	225	83	88	36	4
AggSA	*DEV*	no	10000	1101	95	1	191	12	20	8	5
Clickstream	*DEV*	no	10000	1213	11	1	422	110	145	1	6
Clickstream	*DEV*	yes	10000	1213	253	1	337	5	33	1	249
AggSA	DES	no	78398	749	108	1	30	7	6	6	25
Clickstream	DES	no	78398	749	4	147	242	187	35	180	0
AggSA	OP	no	100000	6533	1322	1	138	5	10	1	826
Clickstream	OP	no	100000	7565	6	1	3220	1261	1418	598	1
AggSA	*DEV*	no	100000	11942	922	1	1850	13	66	5	313
Clickstream	*DEV*	no	100000	12087	8	1	3977	1511	1621	851	2

Note: DES – actionRowDesigners, OP – actionRowOperatorsSimplified, DEV - lvpDevUserActions2023

The next metric used in the analysis of cluster sizes was the percentage distribution of coverage of cluster size types (Figure 9) which illustrates that the AggSA produces the largest number of clusters trying to keep their sizes more or less even but with the tendency to split the remaining objects among multiple groups. This tendency increases along with the increasing of the dataset size. The Clickstream method executes quite the opposite – it produces a small number of clusters of larger sizes.

The same tendency is very well observable in cluster silhouettes in Appendix A and Appendix B. Besides, the silhouettes show the internal quality of the model produced that is quite satisfactorily in case of the Clickstream method, independently of the dataset sizes. The structure of the model produced by the AggSA contains multiple small size well-formed clusters and almost

the third part of the bad-formed clusters, and this tendency increases along with the increasing the size of the datasets.

Besides, the average (original and positive) silhouette widths (Table 6) clearly indicate that there is less stability in the quality of the AggSA than the Clickstream; nevertheless, the AggSA may also form the best silhouettes. However, the average silhouette widths do not consider a number of clusters.

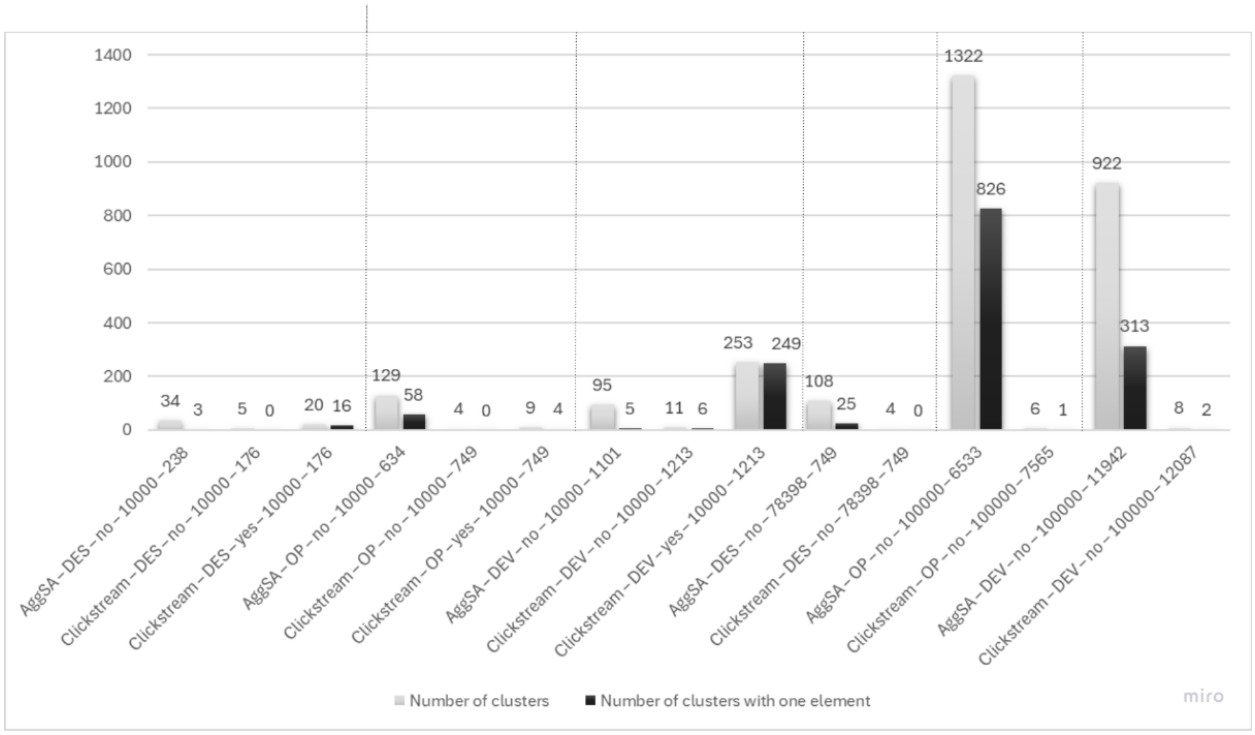


Figure 8. Number of clusters against number of clusters with one element for algorithms

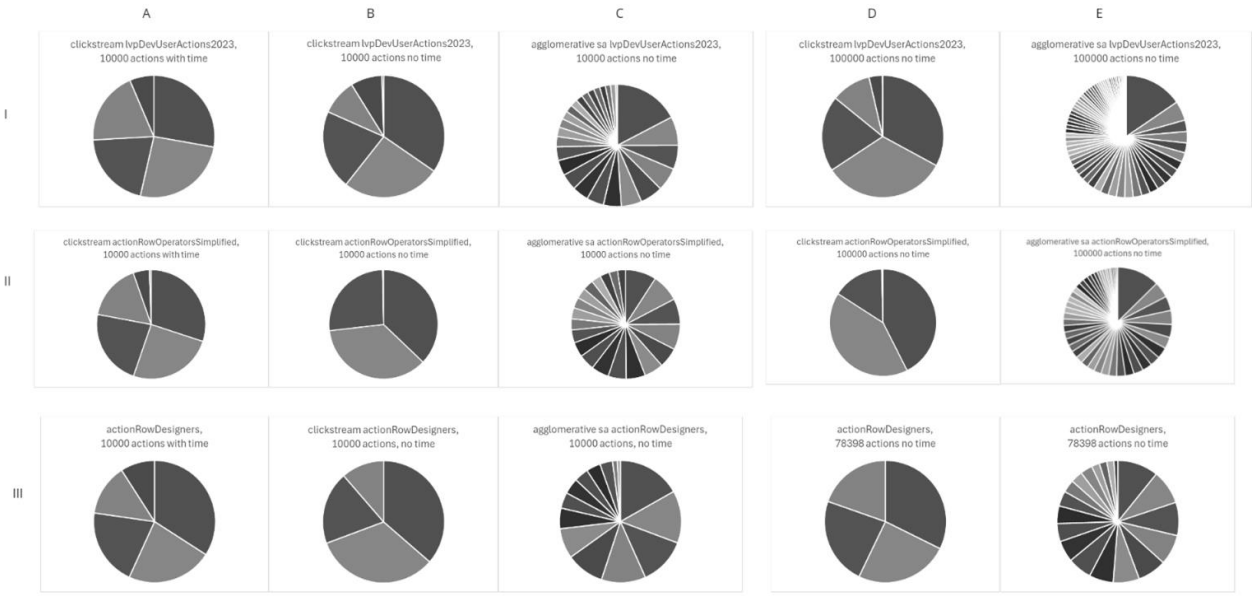


Figure 9. Percentage distribution of coverage of cluster size types

Table 6. Relative cluster metrics for experimental datasets

Experiment	Number of clusters	Number of clusters with one element	Average silhouette width (for the entire dataset) %	Average positive silhouette width (for the entire dataset)%
AggSA – OP – no – 10000 – 634	129	58	-54.1	0.0056
AggSA – DEV – no – 10000 – 1101	95	5	-41.53	0.0065
AggSA – DES – no – 10000 – 238	34	3	-16.59	0.54
AggSA – DES – no – 78398 – 749	108	25	-8.54	12.18
Clickstream – DES – yes – 10000 – 176	20	16	1.06	1.42
Clickstream – DEV – yes – 10000 – 1213	253	249	3.19	3.38
Clickstream – DES – no – 10000 – 176	5	0	3.55	4.04
Clickstream – OP – yes – 10000 – 749	9	4	3.57	3.74
Clickstream – DES – no – 78398 – 749	4	0	3.77	3.86
Clickstream – OP – no – 10000 – 749	4	0	7.61	8.64
Clickstream – DEV – no – 10000 – 1213	11	6	7.88	8.3
Clickstream – DEV – no – 100000 – 12087	8	2	8.08	8.87
AggSA – OP – no – 100000 – 6533	1322	826	9.85	16.97
Clickstream – OP – no – 100000 – 7565	6	1	12.08	13.35
AggSA – DEV – no – 100000 – 11942	922	313	32.57	36.79

4.5 Analysis of Results and Discussion

The main target of the research project is to be able to monitor changes in user behavior in the IS on the daily basis. This means the final target of the project is to be able to analyze a kind of *dynamic datasets* that, in general, may include time-series of static graphs reflecting entity interactions derived from daily, weekly, or monthly collections as mentioned in [4]. However, the first task of the project is to find out the more appropriate algorithms for the ML method. Therefore, the given research considers *static graph clustering*, i.e., ML models produced for the unchanging datasets of user sessions in the system.

The characteristics of the datasets formed a set of requirements to the algorithms: (a) be able to calculate the similarity matrix of sessions based on the sequence of actions in each session, (b) be able to cluster the obtained weighted graph with similarities between vertices (representing the sessions), and (c) get the model of the minimal (as possible) number of clusters of well-formed structure.

The LCS algorithm and the Matching Ratio Calculation were selected as candidate algorithms for calculating similarities. During experiments (Section 4.3) it was found that the LCS requires several optimization techniques: parallelism of executing and implementation in non-interpreted programming language as well as storing the matrix in a more compact format. After experimenting with the optimized versions of the LCS, satisfactory results were achieved. Thus, the optimized LCS implementation in C++ can be used to achieve the project goal. The Matching Ratio Calculation showed better results without optimization but worse if compared to the optimized LCS in C++.

Clustering required a larger number of experiments since two candidate algorithms were selected, namely, the Agglomerative Hierarchical Clustering with weights as a part of the AggSA method and the Louvain algorithm as a part of the Clickstream method. The results of experiments (Section 4.4) allow for the analysis of the model from four viewpoints: the execution time, the cluster sizes, the distribution of cluster sizes, and the cluster silhouettes. The agglomerative method

showed better execution time than the Louvain algorithm executed as on CPU as on GPU. This corresponds to the theoretical findings in Section 3.2. However, other metrics showed that the Louvain is more suitable for clustering datasets of the given type. It produces a smaller number of well-formed clusters that the agglomerative approach that has a well-visible tendency to producing a huge number of clusters of small sizes with the decreasing distribution of sizes to the end.

Validity of the clustering results corresponds to the results published in [39] where clustering by different methods for the real-word datasets is observed. The results in [39] show the similar picture as obtained during the experiments. Thus, the Louvain algorithm showed higher separability, embeddedness, and cluster modularity than other evaluated algorithms on the real-world data. However, on the synthetics data the agglomeration had shown the better results. The same is true for the considered datasets that contain real-world pseudonymized and anonymized data, the Louvain algorithm produces the more satisfying clustering model.

Therefore, the Louvain algorithm together with the optimized LCS shows more appropriate results for the project and will be used in the next iterations as a core part of the ML method and validated during the second research iteration on the datasets of 1 million user actions and the full datasets.

5 Conclusion

Detection of user behavior models based on user action sequences in sessions is a complex task for anonymized and pseudonymized real-world datasets. The first difficulty is that the potential number and structure of the models is unknown. The second is the size of the datasets that increases each day for at least one million sessions.

In order to develop the ML method for detecting these models, we have planned to analyze which action sequence similarity evaluation algorithms and clustering algorithms can provide visible benefits in terms of smaller execution time and more qualitative structure of the model and clusters.

The analysis of the literature showed that there is no any recommended method or algorithm for this task. Besides, plenty of methods, as well as similarity and cluster detecting algorithms, are used. After analysis of the existing methods, two activities of the ML method (beyond the pre-processing of data) were detected: calculation of node similarity and detection of clusters. Since selection of the similarity algorithm is based on the semantics of datasets, two methods that uses similarity calculations were found in the literature: one that leverages the LCS method, and the other one – the Matching Ratio Calculation.

After analyzing classifications of clustering algorithms, we have concluded that agglomerative clustering and bottom-up optimization techniques are the best candidates. Searching for the available promising implementations, we have selected the Agglomerative Hierarchical Clustering with weights and the Louvain algorithm for execution on the CPU.

Calculation of the similarities faced the issue of runtime memory shortage and unsatisfactorily long execution time. After applying the optimization techniques (parallelism and implementation in C++), the LCS showed acceptable results. The Matching Ratio Calculation in its original version showed acceptable results but with longer execution time than the optimized LCS.

Execution of clustering algorithms proved the theoretical findings that the agglomerative approach runs noticeably faster than the Louvain even executed on the GPU. However, the model produced by the agglomerative algorithm does not satisfy the expectations. The result of clustering suffers from excessive separation and bad-formed structure of clusters. The Louvain, in turn, has shown a satisfactory execution time ratio and well-formed structures of clusters, as well as the acceptable number and size of them. The results obtained correspond to the theoretical findings that the Louvain works better than the agglomerative approach on real-world datasets.

Therefore, the first research iteration is finalized, and the ML method for detecting user behavior models shall leverage the optimized LCS algorithm in C++ for the activity of the similarity calculation and the Louvain implementation for the clustering.

The next step of the project is to perform research on the applicability of the current versions of algorithms to the expected sizes of datasets, i.e., for 1 million actions and for the full datasets to prove that the shift from TRL4 to TRL5 is possible with the expected computational resources.

Acknowledgments

The research leading to these results has received funding from the research project "Competence Centre of Information and Communication Technologies" of EU Structural funds, contract No. 5.1.1.2.i.0/1/22/A/CFLA/008 signed between IT Competence Centre and Central Finance and Contracting Agency. The research title is "Constructing User Behavior Models Using Web Data Generalization Methods from User Sessions". The project is co-financed by the Recovery Fund of the Action Program "Latvian Recovery and Resilience Mechanism Plan 5.1.r. 5.1.1.r. of the reform and investment direction "Increasing productivity through increasing the amount of investment in R&D" reforms "Management of innovations and motivation of private R&D investments" 5.1.1.2.i. investment "Support instrument for the development of innovation clusters" implementation rules within the competence centers" framework.

References

[1] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004. Available: https://doi.org/10.2307/25148625

[2] P. Garkalns, O. Nikiforova, V. Zabiniako, and J. Kornienko, "Analysis of the Behavior of Company Employees as Users of Information Systems or Tools, Based on Employees Clustering with K-means Algorithm," in *2023 IEEE 64th International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS)*, IEEE, 2023, pp. 1–7. Available: https://doi.org/10.1109/ITMS59786.2023.10317652

[3] X. Huang, L. V. S. Lakshmanan, and J. Xu, "Community Search over Big Graphs: Models, Algorithms, and Opportunities," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, IEEE, 2017, pp. 1451–1454. Available: https://doi.org/10.1109/ICDE.2017.211

[4] J. Mothe, K. Mkhitaryan, and M. Haroutunian, "Community detection: Comparison of state of the art algorithms," in *2017 Computer Science and Information Technologies (CSIT)*, IEEE, 2017, pp. 125–129. Available: https://doi.org/10.1109/CSITechnol.2017.8312155

[5] A. Karatas and S. Sahin, "Application Areas of Community Detection: A Review," in *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, IEEE, 2018, pp. 65–70. Available: https://doi.org/10.1109/IBIGDELFT.2018.8625349

[6] D. Jin et al., "A Survey of Community Detection Approaches: From Statistical Modeling to Deep Learning," *IEEE Trans Knowl Data Eng*, vol. 35, no. 2, pp. 1149–1170, Feb. 2023, Available: https://doi.org/10.1109/TKDE.2021.3104155

[7] Z. Wu, J. Chen, and Y. Zhang, "An Incremental Community Detection Method in Social Big Data," in *Proceedings – 5th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, BDCAT 2018*, IEEE, 2018, pp. 136–141. Available: https://doi.org/10.1109/BDCAT.2018.00024

[8] C. L. Staudt and H. Meyerhenke, "Engineering Parallel Algorithms for Community Detection in Massive Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 171–184, Jan. 2016. Available: https://doi.org/10.1109/TPDS.2015.2390633

[9] N. R. Smith, P. N. Zivich, L. M. Frerichs, J. Moody, and A. E. Aiello, "A Guide for Choosing Community Detection Algorithms in Social Network Studies: The Question Alignment Approach," *American Journal of Preventive Medicine*, vol. 59, no. 4, pp. 597–605, 2020. Available: https://doi.org/10.1016/j.amepre.2020.04.015

[10] S. Souravlas, A. Sifaleras, and S. Katsavounis, "A Parallel Algorithm for Community Detection in Social Networks, Based on Path Analysis and Threaded Binary Trees," *IEEE Access*, vol. 7, pp. 20499–20519, 2019. Available: https://doi.org/10.1109/ACCESS.2019.2897783

[11] M. H. Hajeer and D. Dasgupta, “Distributed genetic algorithm to big data clustering,” in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2016, pp. 1–9. Available: https://doi.org/10.1109/SSCI.2016.7849864

[12] S. Cheikh, B. Sara, and Z. Sara, “A Hybrid Heuristic Community Detection Approach,” in *2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, IEEE, 2020, pp. 1–7. Available: https://doi.org/10.1109/INISTA49547.2020.9194648

[13] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet Allocation,” *Journal of Machine Learning Research*, no. 3, pp. 993–1022, 2003. Available: https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf. Accessed on Oct. 02, 2024.

[14] H. Wang, L. Pan, Z. Dong, W. Wang, D. Li, and J. Duan, “Community Discovery Algorithm Based on User Behavior Similarity,” in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, IEEE, 2019, pp. 1160–1165. Available: https://doi.org/10.1109/ITNEC.2019.8729363

[15] Z. Lu, X. Sun, Y. Wen, G. Cao, and T. La Porta, “Algorithms and Applications for Community Detection in Weighted Networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 11, pp. 2916–2926, 2015. Available: https://doi.org/10.1109/TPDS.2014.2370031

[16] N. Vo, K. Lee, and T. Tran, “MRAttractor: Detecting communities from large-scale graphs,” in *2017 IEEE International Conference on Big Data (Big Data)*, IEEE, 2017, pp. 797–806. Available: https://doi.org/10.1109/BigData.2017.8257995

[17] S. Chattopadhyay, T. Basu, A. K. Das, K. Ghosh, and C. A. Murthy, “A similarity based generalized modularity measure towards effective community discovery in complex networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 527, p. 121338, 2019. Available: https://doi.org/10.1016/j.physa.2019.121338

[18] Y. Zhao, H. Xu, and C. Zhou, “Nonnegative Matrix Factorization Algorithm with Two Attribute Matrices for Community Detection,” in *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, IEEE, Jun. 2020, pp. 87–90. Available: https://doi.org/10.1109/ICAICA50127.2020.9182466

[19] G. Wang, X. Zhang, S. Tang, C. Wilson, H. Zheng, and B. Y. Zhao, “Clickstream User Behavior Models,” *ACM Transactions on the Web*, vol. 11, no. 4, pp. 1–37, Nov. 2017, Available: https://doi.org/10.1145/3068332

[20] W. Wang and O. R. Zaiane, “Clustering Web sessions by sequence alignment,” in *Proceedings. 13th International Workshop on Database and Expert Systems Applications*, IEEE, 2002, pp. 394–398. Available: https://doi.org/10.1109/DEXA.2002.1045928

[21] D. Lin, “An Information-Theoretic Definition of Similarity,” in *Proceedings of the Fifteenth International Conference on Machine Learning*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 296–304.

[22] S. Tariq, M. Saleem, and M. Shahbaz, “User Similarity Determination in Social Networks,” *Technologies (Basel)*, 2019. Available: https://api.semanticscholar.org/CorpusID:145866830. Accessed on Oct. 03, 2024.

[23] M. Pirouz, “Optimized Ranking-Based Community Detection,” in *Proceedings – 2021 International Conference on Computational Science and Computational Intelligence, CSCI 2021*, IEEE, 2021, pp. 1406–1409. Available: https://doi.org/10.1109/CSCI54926.2021.00281

[24] A. Banerjee and J. Ghosh, “Clickstream clustering using weighted longest common subsequences,” in *Proceedings of the webmining workshop at the 1st SIAM conference on data mining*, vol. 143, Citeseer, 2001, p. 144.

[25] NetworkX developers, “Memgraph’s Guide for NetworkX library,” Girvan-Newman algorithm. Available: https://memgraph.github.io/networkx-guide/algorithms/community-detection/girvan-newman. Accessed on Aug. 29, 2024.

[26] A. Clauset, M. E. J. Newman, and C. Moore, “Finding community structure in very large networks,” *Physical Review E*, vol. 70, no. 6, p. 066111, Dec. 2004, Available: https://doi.org/10.1103/PhysRevE.70.066111

[27] B. Fagginger Auer and R. Bisseling, “Graph coarsening and clustering on the GPU,” in *Graph Partitioning and Graph Clustering*, vol. 588, 2013, pp. 223–240. Available: https://doi.org/10.1090/conm/588/11706

[28] Y. M. ElBarawy, R. F. Mohamed, and N. I. Ghali, “Improving social network community detection using DBSCAN algorithm,” in *2014 World Symposium on Computer Applications & Research (WSCAR)*, IEEE, 2014, pp. 1–6. Available: https://doi.org/10.1109/WSCAR.2014.6916792

[29] H. Aftab, J. Shuja, W. Alasmary, and E. Alanazi, “Hybrid DBSCAN based Community Detection for Edge Caching in Social Media Applications,” in *2021 International Wireless Communications and Mobile Computing, IWCMC 2021*, IEEE, 2021, pp. 2038–2043. Available: https://doi.org/10.1109/IWCMC51323.2021.9498609

[30] Y. Wang, Y. Gu, and J. Shun, “Theoretically-Efficient and Practical Parallel DBSCAN,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ACM, 2020, pp. 2555–2571. Available: https://doi.org/10.1145/3318464.3380582

[31] Y. Li, G. Liu, and S. Lao, “A genetic algorithm for community detection in complex networks,” *Journal of Central South University*, vol. 20, no. 5, pp. 1269–1276, May 2013. Available: https://doi.org/10.1007/s11771-013-1611-y

[32] X. Que, F. Checconi, F. Petrini, and J. A. Gunnels, “Scalable Community Detection with the Louvain Algorithm,” in *Proceedings – 2015 IEEE 29th International Parallel and Distributed Processing Symposium, IPDPS 2015*, IEEE, 2015, pp. 28–37. Available: https://doi.org/10.1109/IPDPS.2015.59

[33] Y. Zheng, Y. Zhu, S. Song, P. Xiong, Z. Cao, and J. Hou, “Improved Weighted Label Propagation Algorithm in Social Network Computing,” in *Proceedings – 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications and 12th IEEE International Conference on Big Data Science and Engineering, Trustcom/BigDataSE 2018*, IEEE, 2018, pp. 1799–1803. Available: https://doi.org/10.1109/TrustCom/BigDataSE.2018.00271

[34] Los Alamos National Laboratory, “Metis,” METIS Interface to LaGriT.

[35] University of Minnesota, “ParMETIS,” ParMETIS – Mesh Graph Partitioning Algorithm. Available: https://license.umn.edu/product/parmetis---mesh-graph-partitioning-algorithm. Accessed on Sep. 25, 2024.

[36] C. Gross and D. Colbry, “ParMETIS – MSU HPCC User Documentation,” ParMETIS. Available: https://docs.icer.msu.edu/available_software/detail/ParMETIS/. Accessed on Sep. 25, 2024.

[37] M. Shamsi and S. Beheshti, “Centrality based number of cluster estimation in graph clustering,” in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing – Proceedings*, IEEE, 2021, pp. 3645–3649. Available: https://doi.org/10.1109/ICASSP39728.2021.9413940

[38] K. Wang, B. Wang, and L. Peng, “CVAP: Validation for Cluster Analyses,” *Data Science Journal*, vol. 8, pp. 88–93, 2009, Available: https://doi.org/10.2481/dsj.007-020

[39] V.-L. Dao, C. Bothorel, and P. Lenca, “Community detection methods can discover better structural clusters than ground-truth communities,” in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, ACM, 2017, pp. 395–400. Available: https://doi.org/10.1145/3110025.3110053

[40] M. Jain, M. Jain, T. AlSkaif, and S. Dev, “Which internal validation indices to use while clustering electric load demand profiles?,” *Sustainable Energy, Grids and Networks*, vol. 32, p. 100849, 2022. Available: https://doi.org/10.1016/j.segan.2022.100849

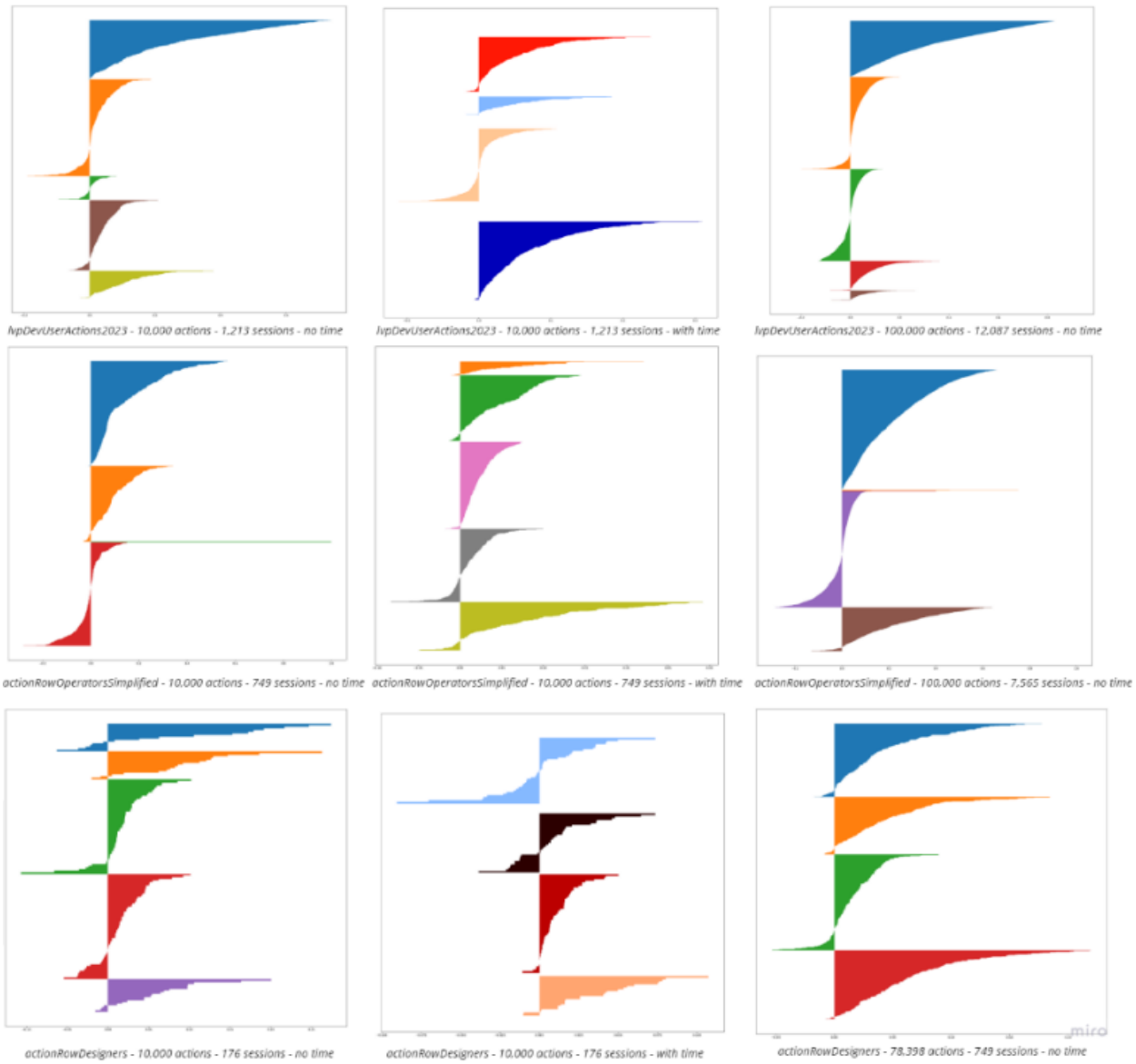
[41] A. Dudek, “Silhouette Index as Clustering Evaluation Tool,” 2020, pp. 19–33. Available: https://doi.org/10.1007/978-3-030-52348-0_2

[42] P. J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,” *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987. Available: https://doi.org/10.1016/0377-0427(87)90125-7

[43] G. D. Quinn, G. H. Bishop, and R. J. Harrison, “On the cluster size distribution for critical percolation,” *Journal of Physics A: Mathematical and General*, vol. 9, no. 1, pp. L9–L14, 1976. Available: https://doi.org/10.1088/0305-4470/9/1/003

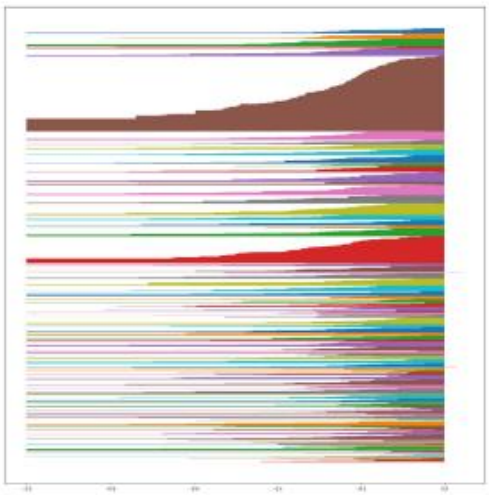
Appendix A

Silhouettes of clusters obtained by Clickstream

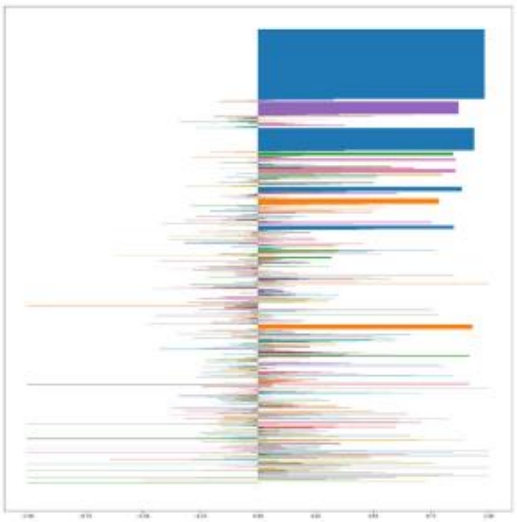


Appendix B

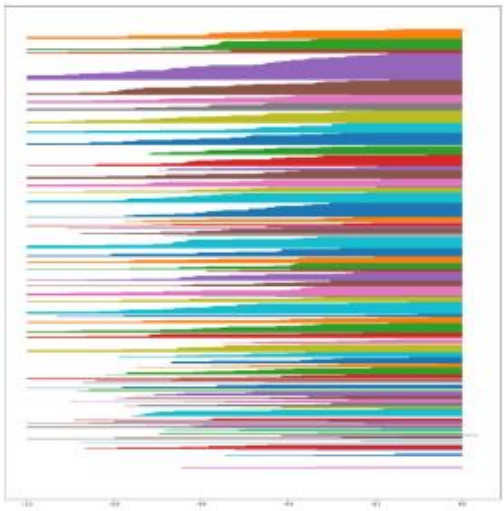
Silhouettes of clusters obtained by AggSA



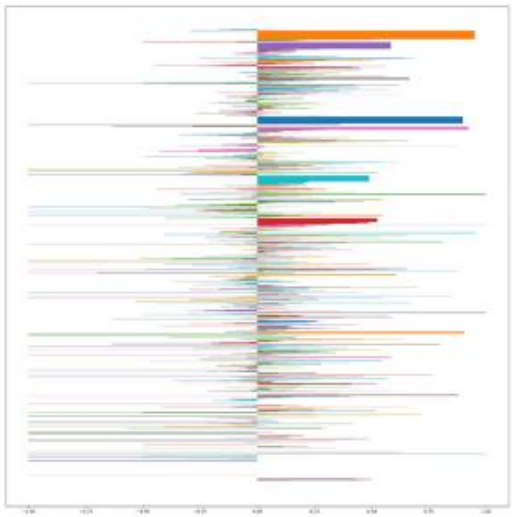
lvpDevUserActions2023 - 10,000 actions - 1,101 sessions - no time



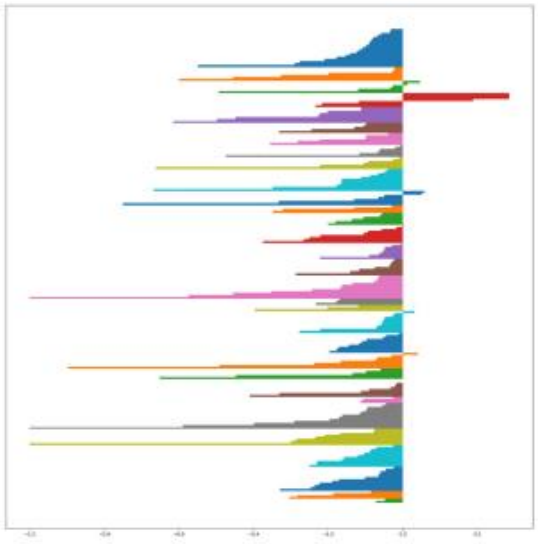
lvpDevUserActions2023 - 100,000 actions - 11,942 sessions - no time



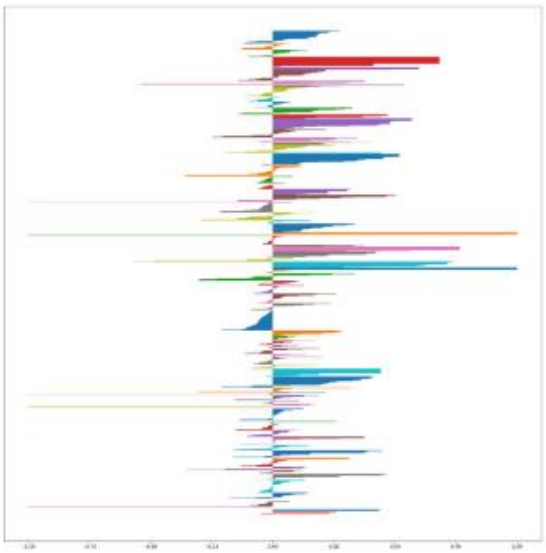
actionRowOperatorsSimplifie - 10,000 actions - 634 sessions - no time



actionRowOperatorsSimplifie - 100,000 actions - 6,533 sessions - no time



actionRowDesigners - 10,000 actions - 238 sessions - no time



actionRowDesigners - 78,398 actions - 749 sessions - no time