Performing Hierarchical Clustering on Distance Matrices in OptiML

Lidija Fodor, Danijela Tešendić, Vladimir Kurbalija, and Srdjan Skrbic

***Abstract*—Since it became evident that domain-specific languages are able to follow the trends of language design evolving demands, they became increasingly popular. This is resulting in the need for a convenient host languages, that can satisfy and ease the aspects of DSL development. Scala is highly flexible, virtualised language, serving as a great ground for different DSLs. In this paper, we used some of the languages, developed on top of Scala, namely Lightweight Modular Staging, Delite and OptiML, to start the development of our own data-mining DSL. Our main aim is to develop a highly efficient DSL for some important data-mining algorithms. As a starting point, we implemented hierarchical clustering in OptiML, with Scala code generation. We also performed testing, based on experimental data, gained from a psychological experiment, related to human behavior analysis and artificial agent development. We compared the results of our hierarchical clustering, to results gained from R on the same data set. This algorithm serves as a starting point towards parallel code generation for a set of data-mining algorithms.**

***Index Terms*—Scala, OptiML, clustering, human behaviour, R**

I. Introduction

TIME series analysis represents an essential tool for obtaining important data in business, as well as in science. The conclusions derived from time series can serve as a ground for important business decisions or significant scientific results. For these reasons, it is important to provide convenient tools for conducting the analysis. Numerous environments are provided for this purpose, but the most common limitation appearing is time consumption. Time series datasets can be large and high-dimensional, and the processing of such data can become significantly slow. [12, 32]

The most evolving trend in language development is to focus on Domain Specific Languages (DSLs), as they can be easier to understand and use, and during development, the focus is on the specific domain, instead of wide range of aspects typical for general purpose languages. This way, the end users of DSL can be focused to their work. M. Fowler [16] explained DSLs in more details in his book "Domain Specific Languages".

There are many different principles of DSL design and development, but generally, we can think about DSL in two ways: as internal (embedded) or external DSLs. An external DSL is developed completely from scratch, where all the different aspects of language specification, compiler and libraries construction, development of IDEs, debuggers, profilers, documentation and many others need to be included. This requires a huge effort, and long time to achieve the satisfying level of maturity. Internal DSLs, also called embedded DSL, make the development process easier, shorter and more reliable. An embedded DSL lives inside of an another language, called host language, and relies on it. This is a convenience for the the developers, but introduces the question how to bridge the gap between the DSL constructions on top and the architecture down below, when the host language is present in the middle.

A DSL, developed in Scala-virtualized [1], called Lightweight Modular Staging (LMS) [2] represents a completely innovative way for embedded DSL development, as it relies on multi-stage programming with code generation. On the other hand, there is a DSL developed on top of LMS, called Delite [3], that introduces the principle of generating target architecture specific DSLs. Finally, a DSL for machine learning, OptiML [4], is developed on top of Delite and LMS. OptiML has implemented many useful algorithms, and allows different kinds of manipulation over different built-in data types.

Our initial goal was to develop a representative version of kmeans algorithm [20], that can perform hierarchical clustering, based on time-series distance matrices, using different linkage criteria. We finished the implementation, and performed some tests to prove the correctness of our function. We are currently generating Scala code, but parallel code generation is our priority in the future. Besides that, we will also focus on including another useful data-mining algorithms [7, 13, 14] to the DSL.

The motivation to develop particularly hierarchical clustering was the test data we have, and the importance of deriving meaningful conclusions. The time-series [5, 31] data were generated from a set of experiments, observing a Socially Augmented Microworld (SAM) [6], where an accent is on detecting different behavior of human factors, based on some instructions they got. The experiment is explained in more detail in Section III-D. This paper makes the following contributions:

- It introduces an implementation of hierarchical clustering, based on time series distance matrices. OptiML already has a built in implementation of some important data-mining functions, but they is not suitable for time-series analysis.
- It represents the first step towards the creation of a set of functions for time series analysis, and wider, for different data-mining algorithms.
- It serves as a ground for developing a highly optimized DSL for parallel execution.

The rest of the paper is organized as follows: Section II, gives an overview of the related work, while Section III-A gives a short description of the hierarchical clustering algorithm. Section III-B is an overview of OptiML, Section III-C describes the implementation of hierarchical clustering

Manuscript received December 1, 2016; revised December 22, 2016.
Authors are with the Department Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia, e-mail: srdjan.skrbic@dmi.uns.ac.rs

ISBN: 978-988-14048-3-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

algorithm, and Section III-D includes the description of experimental test data. Section III-E tells about the process of testing the implementation. In Section III-F we analyse the results of clustering, in terms of data-mining. In Section IV, we discuss the directions for future work, and finally, Section V gives some final conclusions.

II. Related work

The hierarchical clustering problem is an interesting task in data-mining community, and there is a tendency to perform it faster as much as possible. Müllner [25] proposed a C++ library for hierarchical agglomerative clustering, for R and Python. On the other hand, Murtagh et al. [27] gave an exhaustive analysis of Ward method implementations in hierarchical clustering. This is an important concept, as one of our future goals will be to find the best way to implement the algorithm, preserving the whole functionality. An interesting idea is to perform hierarchical clustering with prototypes, with an accent on the minimal linkage criterion, introduced by Bien at al. [28].

The idea of parallel data-mining algorithms appeared early. Jin and Agrawal [26] introduced a middleware, for fast parallel data-mining application development, including both shared and distributed memory. This middleware is appropriate for a cluster of SMP workstations, and uses the idea that mostly, the observed algorithms have very simple structure.

An equally important aspect of an algorithm is the possibility to exploit the domain knowledge of its use. Wagstaff and Cardie [21] used this idea to improve the accuracy of k-means algorithm. This concept is also worth considering in our future work, by applying domain knowledge to hierarchical clustering.

A lot of papers are related to enhancing the process of data-mining, particularly the clustering algorithms. Kanungo et al. [17] introduced an efficient implementation of k-means, as a representative clustering algorithm. Pham et al. [18] evaluated the process of choosing the right value for the number of clusters, when performing k-means clustering. When talking about performance, Arthur and Vassilivitskii [19] gave an overview of the possible worst k-means running time. This ideas serve as an important ground, as we are also interested in improving the k-means algorithm, as part of our DSL, and they can also be adopted to the hierarchical clustering approach.

When considering the another aspect of this paper, related to the evaluation of time-series data, it represents a topic of interest to a large community of researchers. Zhang et al. [8] used time-series analysis on Web search engine transaction log. The usefulness of time-series data mining in making some business decisions is described in the work of Schubert and Lee [9], where some innovations in business data mining techniques are described. The work of Faloutsos et al. [10] caused a great progress in time-series mining concepts, where Keogh [11] gave an overview of this progress.

Our ideas for further work are partly continuing and using the described concepts. On the other hand, we will focus on extending the present state of data-mining algorithms execution, by introducing the concept of running them on MPI clusters.

III. Hierarchical clustering in OptiML

A. Hierarchical clustering

Hierarchical clustering [24] represents a data-mining algorithm, that creates a hierarchy of clusters. Generally, two types of hierarchical clustering are present: agglomerative and divisive. The divisive strategy is a "top down" approach, where all the observations are assumed to belong to one cluster at the beginning. Later, they are divided to smaller clusters. The agglomerative strategy is a "bottom up" approach, where each observation starts in its own cluster. The clusters are later agglomerated, until the moment when just one cluster is left.

In this paper, we used the agglomerative approach. It can be described as follows: given a set of N observations e.g. time-series, a distance matrix (NxN) is created, based on some similarity method. The distance for each observation is calculated with regard to all the other series,then:

- Performing the initial step: each observation is assumed to belong to its own cluster, the distances between clusters are the distances between the observations. At this point, we have N clusters.
- Finding the most similar clusters and merging them to one cluster.
- Calculating the distance between the newly created cluster and the other clusters. There are different linkage methods present, that define the way to calculate the distance between clusters.
- Repeating the process, until there is just one cluster present.

The linkage methods, implemented in this work are: single, complete, average, mcquitty, median, centroid and ward. For our test cases however, we used single, complete and average linkage:

- single linkage - the distance between two clusters is the shortest distance between two observations in each cluster.
- complete linkage - the distance between two clusters is the longest distance between two observations in each cluster
- average linkage - the distance between two clusters is the average distance between each observation in one cluster to every observation in the other cluster.

We chose these three linkage criteria as the possibly most appropriate, based on previous experience with similar data. At the end, we tested the implementation of hierarchical clustering against these three linkage criteria, compared the results and draw some conclusions discussed later.

B. OptiML

OptiML is a DSL itself, meant for machine learning purposes. It has built in complex data types, including vectors and matrices, and also optimized operations on them. This representations are very convenient for representing the data for data-mining algorithms. OptiML relies on code generation, provided by LMS, and is also able to generate parallel code, as it is tightly connected to Delite.

Scala-virtualized is a language, highly applicable to DSL development. It includes a minimal set of extensions to regular Scala, using the principles of language virtualization.

ISBN: 978-988-14048-3-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

These concepts enable the creation of efficient, embedded DSLs. The main idea is that each language construction is defined as a method call, so the DSL developer can reuse it, and redefine its behavior.

LMS is a framework built in Scala-virtualized. It is meant for DSL development, using staging and code generation approaches. It is lightweight, as it is in a form of a library. LMS represents an innovation over earlier approaches, as it does not use quotations to distinguish between binding times, but instead uses types. It introduces staged types, marked with Rep, that wrap the basic type, and hence change the binding time of a value. The approach of generative programming with staging is an important aspect to bridge the gap between performances and high-level DSL code, resulting in specific DSLs, with decent execution times, caused by eliminating abstractions during staging.

Delite is a framework and runtime for building embedded DSL, that are suitable for execution on heterogeneous hardware. Currently, Delite can generate Scala, C++, CUDA and OpenCL code. Delite is embedded in LMS, and supports primitives for parallel operations, for higher level operations definitions.

We used OptiML to implement the algorithm for clustering, while generating Scala code. We tested this version of our DSL, to prove its correctness.

C. Implementing hierarchical clustering in OptiML

For this paper, we used the well known form of hierarchical clustering algorithm. However, many other interesting opportunities exist, and it will be one of the challenges to explore in the future.

We implemented the hierarchical clustering algorithm in OptiML, that consists of a rich architecture of components, described in the previous section. OptiML's built-in data types are extremely convenient for representing the data for the clustering. We used a DenseMatrix instance to represent a distance matrix, read from a file. Further, the nearest neighbour indices and distances are stored in DenseVector instances. The distance matrices were calculated earlier, using the FAP system, described in [15]. These matrices are stored in csv files. OptiML has the capability to create a distance matrix for two vectors or matrices, by means of a predefined function. However, this function supports a few basic distance measures [12], not including the ones we are interested in. For this reason, we are using the already available data, stored in mentioned csv files. Similarly, OptiML includes a function for nearest neighbour calculation, but it works on whole rows in a matrix, calculating the distance of a row, related to other rows of a matrix. It uses the function for distance calculation, based on a predefined distance metric abs. In our case, we need a calculation of nearest neighbours, based on the distance matrix, and we included this concept to the algorithm.The process of clustering is performed, according to the description of hierarchical clustering algorithm, using the wide range of available, optimized functions on DenseMatrix and DenseVector.

The process of staging, ensures that the initial code is being transformed to an intermediate representation. Optimizations are applied at different levels. At the end, we can execute the generated Scala code. We plan to spread this idea, with an aim to generate parallel code. The first step will be to explore the possibility of OptiML to generate CUDA code. Our next step will be to generate and execute MPI code, in order to enable fast and reliable parallel execution of the algorithm on large data sets.

TABLE I: A part of CSV file with DTW distance matrix, for path series for "learning track"

Exper1navigat1	Exper1navigat2	Exper2navigat1
0	18008.35	2759.605
18008.35	0	7192.057
2759.605	7192.057	0
5916.756	3472.793	3530.143
6758.235	4829.235	3382.454

D. The experimental data

The test data are collected from a set of experiments, performed during Human Factors Research at Humboldt University Berlin. The experimental lab system includes two components: a microworld, and a supervisor. The microworld here is called Socially Augmented Microworld (SAM), as it maps a real world environment, including also a social, human component. More precisely, two participants are included in each experiment and their assignment is to perform a cooperative tracking task. The participants need to manipulate an object along a track, showed on a monitor in front of them, using a joystick. The cooperative tracking is achieved by the fact that each joystick contributes to 50 % of motion. A very important aspect is that the participants got different instructions for performing the navigation, but they are not aware of this. One of them is instructed to focus on speed, and the other to perform accurately. This way, the participant focus on different aspects. The process of tracking is also supervised by an operator, who is responsible for encouraging the performances of the navigators.

The state of SAM is logged at the interval of 39 seconds. The system records the data about the navigation performed by the participants. For this purpose, the individual performance of each navigator was observed first, in order to track the cooperative mode later. This kind of data can serve as a great ground for deriving conclusions about different social and psychological aspects of personalities. The patterns, extracted from these kind of data, could later be used for artificial agents creation.

In total, 26 test sessions were performed, each with assigned team of two navigators. Each session consisted of 11 trials, where the four of them are related to solo mode, where each of the navigators is performing on two different tracks, namely learning track and test track. Three trials are related to cooperative mode, and the last four to cooperative mode with a supervisor. For this paper, we consider only the first four steps.

The log files are in form of csv files, and contain all the necessary information, including the total time elapsed from the beginning of the step, x and y coordinates of the object on the track, the total distance crossed from the beginning of the step, the distance crossed in the last measuring and so on. Based on the log files, three different kinds of time-series were extracted, using the FAP system: path, acceleration and

ISBN: 978-988-14048-3-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

deviation time-series. The path series represents the crossed distance data, the acceleration series stores the information about the acceleration or deceleration at each point related to the previous point, and the deviation series represents the deviation from the calculated ideal motion on the track.

These time-series were used to run similarity measures, by the FAP system. Three similarity measures were selected: DTW (Dynamic Time Warping) [23], EDR (Edit Distance on Real sequence) [22] and ERP (Edit distance with Real Penalty) [29]. They represented the base for creating the distance matrices. The reason for choosing these similarity measures lies in their nature. The DTW similarity measure can be successfully applied to such data, where the time-series are not synchronized at the time axis. DTW enables distorting one or both series, that do not have to be of the same length. This was the reason why DTW looked promising for this domain. When talking about EDR, it should be mentioned that it represents a really robust approach, when talking about data containing some distortions. EDR solves this problems, using the principles of comparing two strings, or calculating the similarities between moving objects. On the other hand, ERP is interesting to us, as it combines some properties of Lp norms with local time shifting.

At the end, 18 distance matrices were generated, for the three mentioned similarity measures, 2 types of tracks and 3 types of time-series. Table 1 shows a part of one resulting distance matrix, where the whole matrices are of size 52x52, since there were 26 session with two navigators. These matrices represent the input data for testing the hierarchical clustering algorithm. Additionally, we will make some conclusions, based on the results of clustering.

E. Testing the hierarchical clustering algorithm

In order to test the correctness of resulting clustering, we compared the results, with the results obtained from GNU R [30] for the same distance matrices. The algorithm was tested for all 18 test files, with three different linkage criteria: single, complete and average. This means that 54 test were performed in total. We created matrices of agglomerations as the results of hierarchical clustering, as R also uses them to represent the hierarchical clustering results in form of a dendrogram. We decided to use R for comparison, as it is freely available and easy to use.

The resulting matrices were compared to the matrices from R. It turned out that they were identical. Also, to make the comparison easier and to visualize the results, we generated dendrograms for both our and R's results. We put our results of clustering to an R script file, and created the dendrograms using R. Then, we performed clustering completely in R, based on the distance matrices.

Figure 1 illustrates examples of obtained dendrograms. The plotting itself is still not supported in our DSL, but is planned for future work. Therefore, we used R's plot function to show the results, as explained earlier. As we checked the results for all 54 series, we concluded that our implementation of hierarchical clustering works correctly. Based on the resulting dendrograms, we can make some conclusions about the suitability of particular kind of series, similarity and linkage method.

F. Analysing the results of hierarchical clustering

After we implemented the algorithm for hierarchical clustering, we applied it on our data, gained from a series of experiments. This application was useful to prove the algorithm is correct, but more importantly, we can actually make some conclusions based on the resulting dendrograms.

Firstly, we need to define what is expected. At the description of the experiments, we pointed out that each experimental step included two navigators, instructed differently. In each team, the first navigator was instructed to concentrate on speed, where the second navigator was instructed to focus on accuracy. As a result, we would expect this trend shown on dendrograms to certain extent. When talking about path time-series, the accurate navigators should traveled a shorter distance, than the fast ones, but of course, this depends on personal skills also. The acceleration series should show the same trend. On the other hand, the deviation series are created under the assumption that a faster navigator does not care so much to stay on track as the accurate one, so the faster navigator should have larger values for deviation.

When we look at the dendrograms, we can conclude that the most appropriate similarity measure for all the three types of time-series is ERP, as it gives the closest results, according to expectations. Lets look at three different linkage criteria for the same ERP distance matrix, and compare them. Figure 3 shows complete, single and average linkage for ERP deviation series on test path.

The single linkage criterion seems to be the worst, as it does not show the existence of two types of navigators. Instead, it creates clusters for pairs of navigators on low level. Sometimes, these clusters include differently instructed navigators. The best outcome would be if we could cut the dendrogram on the level where there are two clusters, and get accurate navigators as dominant in one cluster, and fast navigators in the other. This is not the case here, as it seems that separation to two clusters gives one cluster with just one navigator inside, and one with all the others.

If we look at the average linkage criterion, the cutting on the level of two clusters will give the same result as with single linkage. But at lower levels, there are more meaningful information than for single linkage. Some of the clusters show the domination of one type of navigator. However, we still cannot recognize the existence of two types of trends.

The complete linkage method seems the most promising. The situation with two topmost clusters is the same as with the other criteria. It seems that this navigator was performing extremely different than the others, so he cannot be connected with them. We can still try to find two clusters at the second level, where we omit this specific navigator. Here, the dendrogram is truly divided into two parts. The clusters are not of equal sizes, and we cannot clearly distinguish the types of navigators, but it can be concluded that in the cluster on the right side, the accurate navigators are more common. For this reason, the complete linkage criterion is the most promising for this kind of use, and it will be used for further work, in combination with the ERP similarity measure.

Further work will be related to testing this tools with similar problem sets. The conclusions can be significant for artificial agents development. The second direction will go towards performance improvement for this and similar algorithms.

ISBN: 978-988-14048-3-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

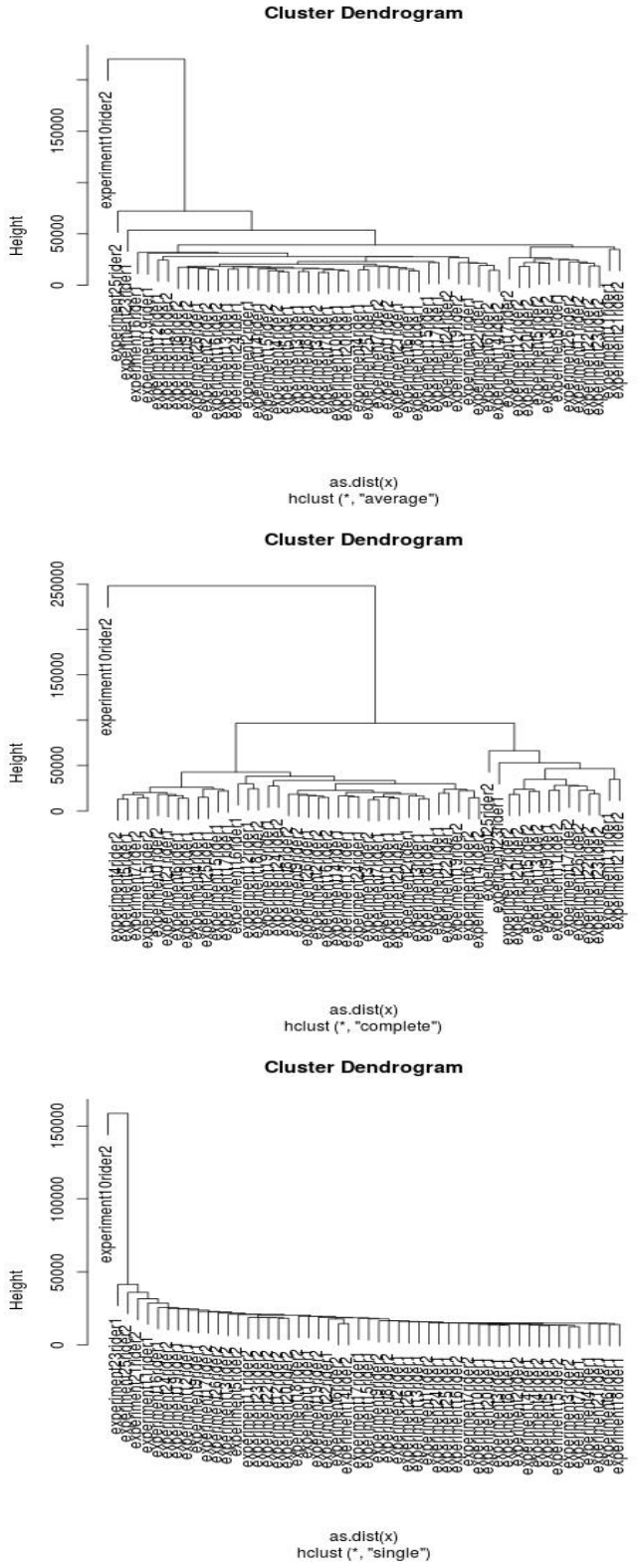

Fig. 1. Dendrograms for average, complete and single linkage, for ERP deviation series for test path.

IV. Future work

Our main goal is to develop a small and efficient DSL for a set of widely used data-mining algorithms. This work is the first step towards this idea. We have two challenges to master. The fist one is to expand our DSL with other needed algorithms for data analysis. This should include in the first place, logistic regression and k-means clustering. The k-means algorithm is already implemented and is currently under testing.

The second request is to make the DSL as more as possible efficient. There is a lot of tools present, that can perform data analysis on different manners. However, one of the most common problems is related to large data sets, that can results with a few days long execution of the analysis. Our idea is to connect the trend of parallel program execution with data-mining algorithms. We plan to achieve this, by generating firstly CUDA code, that represents an already available option in OptiML. The second step will be to transform the program to be applicable for executing on MPI clusters. This way, we could shorten the execution time of data-mining algorithms drastically, and provide a transparent way achieving parallelization with MPI.

V. Conclusion

In this paper, we introduced the idea to develop a DSL in OptiML, meant for data-mining algorithms. We provided an initial form with the hierarchical clustering algorithm implemented. We also described a real case study used for testing and the obtained results. Finally, we revealed the idea to include MPI code creation, in collaboration with the described tools.

Acknowledgment

This research was supported by the Ministry of Education, Science, and Technological Development of the Republic of Serbia under project ON174023 and SCOPES project No. IZ74Z0-160453. The authors would like to thank also to DAAD (German Academic Exchange Service) and to colleagues from Humboldt University, Berlin, for making available dataset used in this paper. Special thanks are due to Vojin Jovanović for his valuable feedback and suggestions.

References

[1] A. Moors, T. Rompf, P. Haller, M. Odersky. Scala-virtualized. In Proceedings of the ACM SIGPLAN 2012 workshop on Partial evaluation and program manipulation, 2012, pp. 117-120

[2] T. Rompf, M. Odersky. Lightweight modular staging: A pragmatic approach to runtime code geenration and compiled DSLs. In Proceedings of the ninth international conference on Generative programming and component engineering. GPCE '10, 2010

[3] A. K. Sujeeth, K. J. Brown, H. Lee, T. Rompf, H. Chafi, M. Odersky, K. Olukotun. Delite: A Compiler Architecture for Performance-Oriented Embedded Domain-Specific Languages, ACM Transactions on Embedded Computing Systems (TECS), vol 13, no 4s, article no. 134, 2014, doi: 10.1145/2584665

[4] A. K. Sujeeth, H. Lee, K. J. Brown, T. Rompf, H. Chafi, M. Wu, A. R. Atreya, M. Odersky, K. Olukotun.

OptiML: An Implicitly Parallel Domain-Specific Language for Machine Learning. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 609-616

[5] V. Kurbalija, H-D. Burkhard, M. Ivanovic, C. Mayer, J. Nachtwei, L. Fodor. Time-series mining in a psychological domain. In proceedings of the Fifth Balkan Conference in Informatics, 2012, pp. 58-63

[6] H-D. Burkhard, L. Jahn, S. Kain, C. Meyer, J.Muetterlein, J. Nachtwei, N. Niestroj, S. Rougk and M. Schneider. Artificial Subjects in the Psychological Experiment "Socially Augmented Microworld (SAM)". In Proceedings of International Workshop CS&P 2011, pp. 54-66

[7] J. Han and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, CA, 2005

[8] Y. Zhang, B. J. Jansen and A. Spink. Time-series analysis of a Web search engine transaction log. Information Processing and Management, vol 45, no 2, 2009, pp. 230-245

[9] S. Schubert and T. Lee. Time Series Data Mining with SAS®Enterprise Miner. In Proceedings of SAS Global Forum 2011 conference, 2011

[10] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In Proceedings of the 1994 ACM SIGMOD international conference on Management of data, 1994, pp. 419-429

[11] E. J. Keogh. A Decade of Progress in Indexing and Mining Large Time-series Databases. In Proceedings of the 32nd international conference on Very large data bases, 2006, pp. 1268-1268

[12] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, E. Keogh. Querying and Mining of Time-series Data: Experimental Comparison of Representations and Distance Measures. In Proceedings of the VLDB Endowment, vol 1, no 2, 2008, pp. 1542-1552

[13] C. A. Ratanamahatana, J. Lin, D. Gunopulos, E. Keogh, M. Vlachos and G. Das. Data mining and knowledge discovery handbook 2010, Part 6, pp. 1049-1077, DOI=10.1007/978-0-387-09823-4_56

[14] D. Pfoser, Y. Tao, K. Mouratidis, M. A. Nascimento, M. Mokbel, S. Shekhar and Y. A. N. Huang. Advances in Spatial and Temporal Databases (SSTD), 12th International Symposium, 2011

[15] V. Kurbalija, M. Radovanović, Z. Geler and M. Ivanović. A framework for time-series analysis. In Proceedings of the 14th international conference on Artificial intelligence: methodology, systems, and applications (AIMSA'10), 2010, pp. 42-51

[16] M. Fowler. Domain specific languages. Addison-Wesley Professional, 2010

[17] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A. Y. Wu. An Efficient k-Means Clustering Algorithm: Analysis and Implementation, IEEE transactions on pattern analysis and machine intelligence, vol. 24, no. 7, 2002, pp. 881-892

[18] D. T. Pham, S. S. Dimov, C. D. Nguyen. Selection of K in K-means clustering. In Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, vol. 219, no. 1, 2005, pp. 103-119.

[19] D. Arthur, S. Vassilivitskii. How slow is the k-Means Method? In Proceedings of the twenty-second annual symposium on Computational geometry, 2006, pp. 144-153

[20] G. Hamerly and C. Elkan. Learning the k in k-means. Advances in neural information processing systems 16, 2003, pp. 281-288

[21] K. Wagstaff, C. Cardie. Constrained K-means clustering with background knowledge. In Proceedings of the Eighteenth International Conference on Machine Learning, 2001, pp. 577-584

[22] L. Chen, M. T. Ozsu, V. Oria. Robust and Fast Similarity Search for Moving Object Trajectories. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data, 2005, pp. 491-502

[23] D. J. Berndt, J. Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In Proceedings of KDD Workshop, 1994, pp. 359-370

[24] A.K. Jain, M.N. Murty, P.J. Flynn. Data clustering: a review. ACM Computing Surveys (CSUR), vol. 31, no. 3, 1999, pp. 264-323, doi:10.1145/331499.331504

[25] D. Müllner. Fastcluster: Fast hierarchical, agglomerative clustering routines for R and Python, Journal of Statistical Software, vol. 53, no. 9, 2013, pp. 1-18

[26] R. Jin, G. Agrawal. A Middleware for Developing Parallel Data Mining Applications. In Proceedings of the first SIAM conference on Data Mining, 2001, pp. 1-18

[27] F.Murtagh, P. Legendre. Ward's Hierarchical Clustering Method:Clustering Criterion and Agglomerative Algorithm. Journal of classification, vol. 31, no. 3, 2014, pp. 274-295, doi: 10.1007/s00357-014-9161-z

[28] J. Bien, R. Tibshirani. Hierarchical Clustering With Prototypes via Minimax Linkage. Journal of the American statistical Association, vol. 106, no. 495, 2011, pp. 1075-1084

[29] L. Chen and R. T. Ng. On the marriage of lp-norms and edit distance. In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. VLDB Endowment, 2004, pp. 792-803

[30] R Development Core Team. The R language definition. R Foundation for Statistical Computing, http://cran.r-project.org/doc/manuals/R-lang.html

[31] V. Kurbalija, C. von Bernstorff, J. Nachtwei, M. Ivanović, HD. Burkhard. Matching observed with empirical reality-What you see is what you get? Fundamenta Informaticae, IOS Press, vol. 129, no. 1-2, 2014, pp. 133-147, doi: 10.3233/FI-2014-965

[32] V. Kurbalija, M. Radovanović, Z. Geler, M. Ivanović. The influence of global constraints on similarity measures for time-series databases. Knowledge-Based Systems, vol. 56, 2014, pp. 49-67, doi: 10.1016/j.knosys.2013.10.021

ISBN: 978-988-14048-3-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)