How An Optimized DBSCAN Implementation Reduces Execution Time And Memory Requirements For Large Data Sets

Ole Kristian Ekseth and Svein-Olaf Hvasshovd
Department of Computer Science (IDI)
NTNU
Trondheim, Norway
email: oekseth@gmail.com and sophus@ntnu.no

***Abstract*—In data-analysis the use of approximate cluster algorithms has received broad popularity. A popular cluster algorithm is the DBSCAN cluster algorithm. While a number of software libraries provide support for the latter, they provide poor performance when analysing high dimensional data. In this work we address this issue. We present a novel method and implementation which significantly boosts the performance of DBSCAN. The result is a software which reduce the memory consumption by $10^3 GB$ for large data sets while reducing the execution time by 600x+ (for important similarity metrics). This artilce presents a high-performance appraoch to identify answers to region based similarity queries. While our work is tuned towards the application of DBSCAN, our novel approach for high-performance filtering of pairwise similarity scores may be used in a number of cluster algorithms. Therefore, the proposed method and software manages to address issues which are known to hamper high dimensional data analysis.**

Keywords: ***clustering; similarity metrics; data analysis; performance*.**

I. INTRODUCTION

A center piece in high productive research is the availability of accurate and fast data analysis. The authors of [1] observe how application of established cluster algorithms require accurate knowledge of cluster algorithms and their configuration. The works of [2], [3], [5]–[7] observe that the size in data sets outgrows the speed of computer software. To address the performance issues numerous algorithms are presented every year, eg, with respect to algorithm permutations of k-means [8], SLINK [9], MCL [10], etc. A strategy for performance improvement concerns the application of *K-D tree* [11], which is used to reduce the time cost of pairwise similarity metric computation. For many clustering algorithms the major time consumer is the task to compute pairwise similarity, *e.g.*, through application of the Euclidean pairwise similarity metric. Figure 1 measures the time cost of different strategies for computation of pairwise similarity. The growth curve in time consumption (Figure 1) implies that it is unfeasible to apply "Density-based spatial clustering of applications with noise" (*DBSCAN*) on large data sets, *i.e.*, when using established software.

The use of heuristics, such as *K-D tree*, has many weaknesses. The work of [12] observes that in the analysis of "high-dimensional data, indexing schemes such as k-d tree do not work well" [12]. This is due to difficulties in designing a splitting criteria for rows which correctly captures the geometry of a high dimensional space. To overcome this issue many alternative approaches are described, such as with respect to "k-means mini-batch" [13], "k-means++" [14], and "k-medoid". The results in Figure 4 demonstrate how the *Expectation–Maximization* (EM) based cluster algorithms of "k-means++" [13] and "k-means" [8] identify cluster predictions with poor accuracy for cases where the type of data normalization and similarity metric is inaccurate. (While Figure 4 focuses on prediction accuracy on synthetic data sets, the complete set of measurement results may be generated through the scripts described in sub section V-C.)

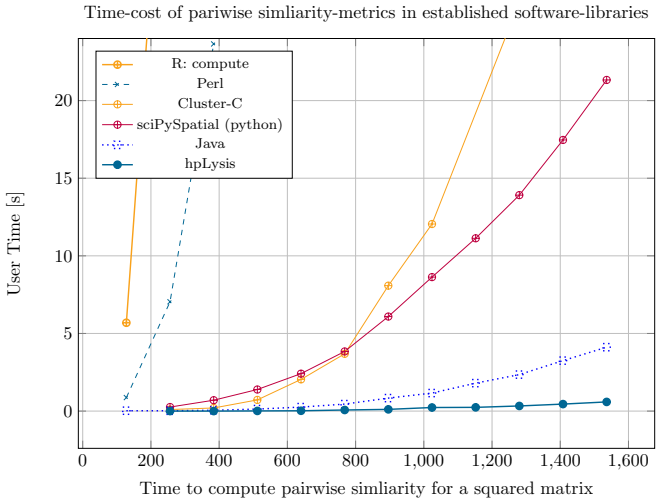

Fig. 1. The time cost of pairwise similarity metrics. The above figure captures the time cost of different strategies for computation of the Euclidean pairwise similarity metric. We observe how an naive python implementation causes a 600x+ execution timeperformance overhead.

While iterative cluster algorithms such as "k-means" permutations apply strategies for centrality to capture core traits in networks, there are alternative strategies which are faster and (for some use cases) more accurate, *e.g.*, with respect to the

 ISBN: 978-1-61208-612-5

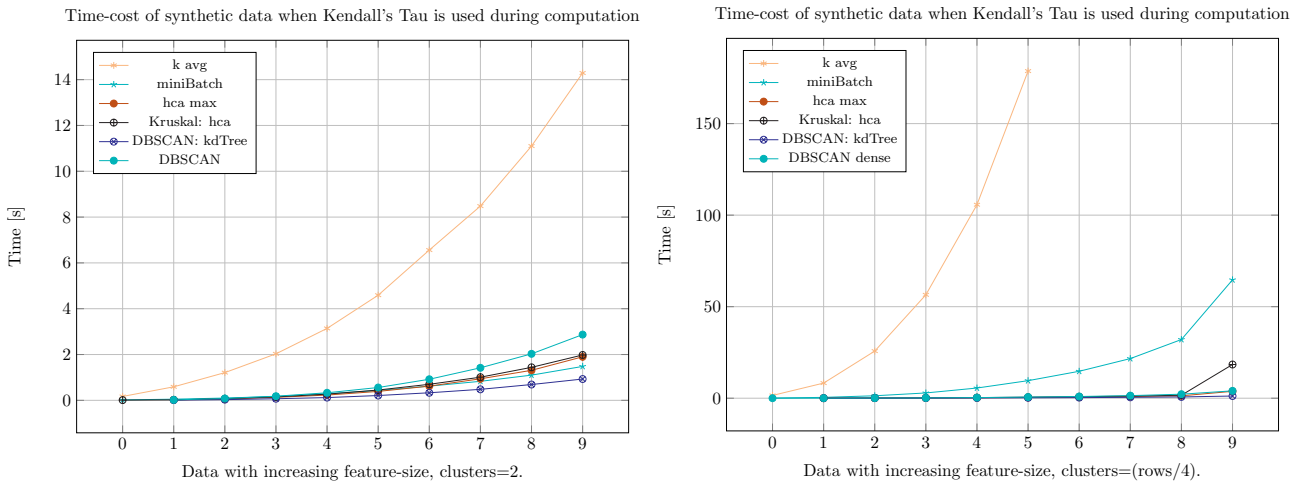

Fig. 2. Execution time between different cluster algorithms. The above figure measures the execution time on synthetic data sets with well defined clusters, thereby avoiding the predictions from being limited to the evaluated real life data sets. From the figure it is clear that the relative performance improvements of *DBSCAN* increases with increased feature size and cluster count.

DBSCAN algorithm [15]. The popularity of *DBSCAN* stems from its low execution time (Figure 2) and accurate predictions (Figure 4). The challenge in application of *DBSCAN* concerns its high execution time for data sets with many features [12]. An approach which addresses the latter may therefore increase the applicability of *DBSCAN*, hence improving a number of efforts in data mining. In brief a major challenge of the *DBSCAN* cluster algorithm concerns its high execution time and inaccurate cluster prediction results.

Application of our high performance machine learning library *hpLysis* [16] offers the ability to address the latter issue. The hpLysis software provides support for 320+ pairwise similarity metrics and 20+ unsupervised cluster algorithms. However, its high memory consumption makes it unfeasible for data analysis on large data sets.

In this work we unify the expressiveness of the hpLysis software with the requirements of large scale data analysis:

1) time and memory: design a software which reduces the execution time by 600x+ (Figure 1) while reduces the memory consumption by a factor of $10^7 x$ (Section III);
2) prediction quality: support accurate similarity metrics, hence improving prediction accuracy by 100x+ (sub section V-B);
3) applicability: a new library for high performance inference of filtered similarity metrics from a dense evaluation of pairwise similarity metrics (sub section V-C).

The remainder of the paper is organized as follows. Section II briefly surveys related approaches, and Section III describes the approach. Section IV describes an approach to evaluate the influence of the data mining strategy proposed in this paper, a method which is applied in the result Section V. This paper ends with a brief summary of observations in Section VI.

II. Related Work

An evaluation of current approaches for improving prediction accuracy and execution time of *DBSCAN* cluster

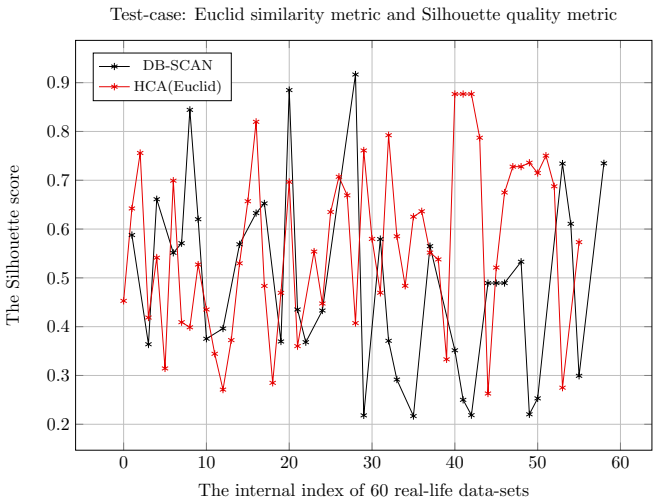

Fig. 3. A quality comparison of "SLINK" [9] and *DBSCAN*. The above figure describes the prediction accuracy of cluster algorithms on 60+ real life data sets found in the *real-kt* data set [16]. In the evaluation of prediction accuracy the "Silhouette Index" [17] is used.

algorithms [15] reveals issues in:

1) prediction quality: while software for *DBSCAN* and pairwise similarity metrics are focused on similarity metrics (sub section V-B), existing approaches are limited to Minkowski based metrics;
2) time and memory: software for data mining does not apply low level performance tuning strategies (Figure 1).

In below we identify key approaches in software and algorithms for *DBSCAN*. The importance of efficiently supporting the 320+ established similarity metrics stems from the purpose of clustering, *i.e.*, to accurately identify groups of entities. An example concerns the comparison of features which use different scales, an use case where Minkowski based metrics

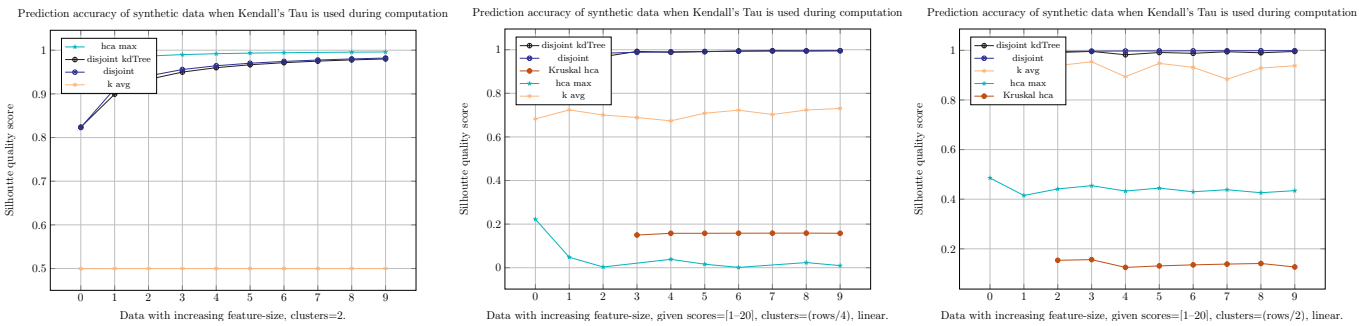

Fig. 4. The quality of cluster algorithms. The above figure evaluates the "Silhouette" [17] prediction accuracy for the cluster algorithms described in Figure 2. Each of the sub figures describe different cluster topologies.

fail.

A. Prediction quality

DBSCAN is asserted to provide a high degree of prediction quality in minimal execution time [18], [19]. A weakness in review articles such as [18], [22] concerns their lack of discussion with respect to the different *DBSCAN* implementations. In *DBSCAN* there are two different approaches for optimization of algorithm: to either use a *K-D tree* approach [11] in *DBSCAN* [3], [20] or to access the relationships from a matrix of scores, *e.g.*, as seen in the work of [21]. "*K-D tree* nearest neighbour (kdNN) [etc.,] has many characteristics like: simple, fast, and provide completely balanced tree [etc.,] has several significant drawbacks such as: need extreme search, time consuming, and impulsively divide points into two equal parts which may miss out on data structure" [22]. While [19] evaluates the performance of *DBSCAN* [22], they do not consider the implication of replacing *K-D tree* with a dense similarity metric computation, a view which is in contrast to [12].

B. Execution Time and Memory Consumption

For *DBSCAN* software implementations concerned with supporting high dimensional data analysis, the main time cost is associated with the computation of similarity metrics (Figure 1). To address this issue *K-D tree* [11] is used, a strategy which reduces the execution time. While the time complexity of a matrix based *DBSCAN* approach is of $O(n^2 f)$ (where n is the number of vertices in the graph and f is the features), a *K-D tree* approach has a time complexity of $O(n * log(n))$ [11]. The inaccurate cluster prediction accuracy of *K-D tree* application [22] motivates the computation of a *dense* similarity matrix: to apply filters during *run time*, hence reducing the memory consumption from n^2 to $n * k$, where k denotes the maximum number of adjacency vertices to be stored for each vertex. Established software does not provide support for the latter, *e.g.*, with respect to the "scikit-learn" software library [23]. A different aspect concerns the application of parallel hardware to reduce the time cost of computations, eg, through application of GPUs in [21]. In [21] the authors assert that they manage a 100x performance improvement when compared to a CPU based implementation. The measurements presented in sub section V-B identifies how the accurate choice of similarity metrics is critical for prediction accuracy, hence the limited applicability of their approach. A different aspect concerns their comparison basis, where Figure 1 demonstrates how a poor implemented C/C++ implementation results in a 30x+ performance-overhead, while the use of a naive R implementation introduce a 3000x+ performance penalty.

III. Method: Run Time Filtering and Hardware Close Optimization

The optimized implementation of *DBSCAN*, described in this paper, focuses on addressing the performance of dense similarity metric computation. The idea is to update the computation of each of the 320+ pairwise similarity metrics, supported by the *hpLysis* machine learning library [16], through introduction of run time filtering. This Section focuses on how to update the 320+ established metrics without introducing performance penalties to the hardware close optimization strategies applied by the *hpLysis* software: to compute similarity between $O(n^2)$ pairs of measurements without using $10^{12}B = 10^3GB$ for a data set with 10^6 rows.

In order to provide support for computation of dense pairwise similarities without requiring the storage of all pairwise similarities, the *hpLysis* is updated with a new data structure and related logic. The result is a performance optimized approach which store $n * k$ relationships instead of n^2 relationships, where n represents the number of feature rows, while k denotes the user specified number of best performing vertices to remember. In the high performance computation of similarity metrics *hpLysis* makes use of the optimization strategy described in [24].

To address the requirement of result filtering during computation of pairwise similarity metrics, *hpLysis* is updated with new logics to post process each *matrix tile*. A *matrix tile* represents an $matrix[i...(i+b)][m...(m+b)] \in matrix$ where i is a row index in the matrix, m is a column index (in the matrix) and b is the block size. Each of the evaluated *matrix tiles* are merged after the tiles are computed. This strategy translates into a significant reduction in memory consumption without introducing penalty to execution time nor parallelism:

 ISBN: 978-1-61208-612-5

1) arithmetic: the time cost of similarity metrics regards the computation of each *matrix tile*, hence the introduction of post processing steps (separately for each *matrix tile*) does not result in a noteworthy increase in execution time;
2) parallelism: each thread computes separately for a block of row combinations, hence the merging of *matrix tiles* does not include any performance overhead with respect to communication nor memory delays;
3) memory consumption: for a matrix with 10^9 rows and where only the 10 best row similarities are of interest (for each vertex) the proposed approach enables a 10^8x reduction in memory requirement (when compared to the default approach), *i.e.*, from $10^{18}B = 10^9GB$ to $10^{10}B = 10GB$.

Therefore, the proposed approach enables accurate and efficient evaluation of large scale data sets. A complexity in the *matrix tile* merge step concerns the use cases where a *ranked step* is to be applied, *i.e.*, where each row is to hold the best performing relationships. In the merging of *matrix tiles* each row is evaluated separately. For the *ranked step* sorting is required, for which the "quick-sort" algorithm [25] is used. Hence, the time cost of a *ranked step* in matrix filtering is $O(n * log(n))$, where n is the number of vertices. The latter time cost is insignificant when compared to the overall time cost of dense similarity computation of $O(n^2f)$, where f is the number of column features. Therefore, the proposed merging procedure will not increase the overall execution time of similarity metric computation.

The *parallel computation* support is enabled through a new compile time added parameter to the *hpLysis* software. For the parallel computation strategy to produce correct results, a requirement has to ensure that the row identities of a given matrix block is separately handled by each computer thread (*i.e.*, not shared by multiple threads). In our implementation of the parallel thread scheduler, the scheduler delegates tile matrix blocks $matrix[i....m][0...|cols|]$ to a given computer thread, hence the parallel scheduler avoids the need for a *writing lock* when updating the sparse result containers.

IV. Experimental Setup

This article seeks to improve data mining through provision for a fast *DBSCAN* implementation in support of 320+ pairwise similarity metrics.

A. Execution Time and Prediction accuracy

To evaluate differences in execution time the following strategies are applied:

1) similarity metrics: through application of established strategies to capture the isolated time cost of similarity metric computations;
2) cluster algorithms: evaluate the 20+ cluster algorithms supported in the *hpLysis* software, thereby identifying the benefits of the proposed approach in application of data mining;
3) data topology: how data sets capturing different topological use cases are influenced by the approach, for which the prediction accuracy is evaluated.

In order to capture the benefit of the large number of published software we relate software implementation to execution time, results which for brevity are summarized in Section V.

B. Prediction quality: Unsupervised parameter estimation to avoid bias in comparison of prediction accuracy

A challenge in the evaluation of large data sets concerns the difficulty in manually identifying correct parameter configurations. The severity is further increased with respect to the importance of choosing a dedicated similarity metric, and the types data optimization strategies to apply. "The performance of the JarvisPatrick algorithm and DBSCAN depend on two parameters: neighborhood size in terms of distance, and the minimum number of points in a neighborhood for its inclusion in a cluster" [26]. In order to compare unsupervised cluster algorithms on large data ensembles without introducing bias, the following requirements need to be satisfied:

1) similarity metrics: the *hpLysis* library is used to evaluate the prediction influence of similarity metrics and strategies for data normalization;
2) algorithm configurations: iteratively explore/evaluate numerous cluster parameters for *DBSCAN* and "k-means";
3) prediction quality: different metrics for external cluster validation is used, *e.g.*, with respect to "SSE" [8], "Dunn's Index" [27], "Silhouette" [17], and "VRC" [28], etc.

The above list identify strategies which addresses issues known to introduce bias in evaluation approaches for cluster algorithms. In the identification of 'k' clusters in EM based cluster algorithms, *e.g.*, k-means, "SLINK" [9] is used as an initial pre step. The "SLINK" result is then partitioned into disjoint clusters for different 'k' counts, and where the 'k' with the best cluster validity score (*e.g.*, measured through "Silhouette" [17]) is selected.

V. Result: Empirical evaluation

The proposed strategy to optimize *DBSCAN* manages to address issues in execution time and prediction accuracy of data mining efforts. The figures included in this paper exemplifies how the following requirements are handled in our approach:

1) time and memory: the design of a new approach for run time filtering during dense similarity metric computation (Section III) decrease the performance discrepancy between *off the shelf computer hardware* versus *cluster algorithms*;
2) prediction quality: how application of the 320+ supported similarity metrics address issues in prediction accuracy of *DBSCAN*.

 ISBN: 978-1-61208-612-5

A. Time cost and Memory consumption

To evaluate the influence of different implementation strategies and data topologies, we measure the difference in execution time between alternative strategies. Figure 1 captures the implication of different approaches for implementing similarity metrics. To investigate the applicability of *DBSCAN* in data analysis 20+ cluster algorithms [16] are applied to 100+ real life data sets and 320+ pairwise similarity metrics, where a subset of the measurements are included in Figure 2. The improvement in both prediction accuracy and execution time, as identified in this paper, is due to the low level implementation strategies which we apply. Section III describes how the proposed software implementation manages to reduce the memory requirements from n^2 to $n*k$, where n is the number of input rows, and k is the number of best scoring pairs to use. Therefore, the software described in this paper enables an n/k reduction in memory requirements. For $k = 2$ our approach implies a reduction factor of $10^9/10^2 = 10^7$ with respect to the memory requirement.

Popular/established/optimized software implementations provide support for only a limited number of similarity metrics, *e.g.*, where "sci-kit-learn" [23] is designed only for Minkowski based metrics. Hence, users are required to write their own support for similarity metrics, which for "R" introduce a 3000x+ performance penalty (Figure 1). The measurements presented in this paper demonstrates how the combination of high performance *DBSCAN* implementation and accurate similarity metrics reduces both the execution time and increases the prediction accuracy (Figure 3). While Figure 3 presents a subset of the measurements on the *real-kt* data set [16] data set, Figure 4 measures the prediction accuracy on synthetic data sets with well defined clusters, measurements which agree in the prediction accuracy of *DBSCAN*.

B. Prediction difference: DBSCAN compared to other unsupervised cluster algorithms

The related work Section II exemplifies how the execution time of *DBSCAN* is highly reliant on the topology of micro benchmark data, an observation verified in Figure 3. The findings presented in this paper represents a small subset of the observations. An example concerns the influence of pairwise similarity metrics in data analysis. The measurement scripts in sub-section V-C identifies how accurate choice of pairwise similarity metrics improves prediction accuracy of more than 100x. Hence, the accurate choice of similarity metrics relates directly to an increase in the prediction accuracy. To ensure representativeness of the described prediction differences, both measures for *internal metrics* (*e.g.*, "Silhouette") and *external metrics* (eg,"Rand's Index" [29] and "Adjusted Rand's Index (ARI)" [30]) have been evaluated. In brief they provide consistent prediction results.

When discussing prediction accuracy this paper focuses on *internal metrics*, an approach which is designed to avoid certain algorithms from gaining a favorable bias during the construction of micro benchmarks. To exemplify the latter, an evaluation of the well known IRIS flower data set (with three established cluster partitions) with the MINE similarity metric [31] reveals how there are only two clusters present, while Euclidean clearly identifies 2.3 clusters in the IRIS flower data set. While the latter result may indicate that Euclidean provides a better closeness to the gold standard, an investigation into the feature similarities reveal patterns in the features which clearly supports the MINE assumption (of two clusters in the IRIS data set), *i.e.*, for which we have observed how the application of *external metrics* may present results which diverge from visual perspectives of cluster similarity.

C. Reproducibility

The proposed software is implemented in the *hpLysis* software, and made accessible through:

1) *DBSCAN*: integrated in the *hpLysis_api.h*;
2) sparse similarity: accessible from the *hp_api_sim_dense.h* API, an interface for inferring sparse sets of similarity metrics from a dense evaluation of pairwise similarity metrics;
3) terminal interface: *x_hp_sim_dense* enables computation of sparse similarity and sparse *DBSCAN*,

The above interface provides access the high performance optimization strategy described in Section III. The implementation level details of the result merging is found in the *insertBlockOf_scores__sorted__kt_list_2d(..)* function (*kt_list_2d.h*, where latter file is found in the *hpLysis* repository).

To enable validation and extension of the performance measurements included in this paper, a number of new use cases have been included into *hpLysis*, such as:

1) effect of pairwise similarity metrics: *tut_sim_15_manMany_Euclid__selectBestInsideMetricComp.c*;
2) accuracy of cluster algorithms: *tut_clust_dynamicK_7_find_nCluster_multipleData_allSim_xmtPreFiltering.c*;
3) execution time measurements: *tut_time_1_data_syntetic_wellDefinedClusters.c* and *tut_time_2_clustAlg_syntAndReal.c*.

The above measurement scripts makes use of the synthetic data sets constructed from *hp_clusterShapes.h* in combination with the data ensembles of *real-kt*, *kt-mine*, *shapes* (located in the hpLysis repository).

VI. Conclusion and Future Work

This paper has presented a new approach to compute *DBSCAN* on large data sets, both with respect to sparse data sets, dense matrices, and dense matrices with missing values/scores. The results which are presented demonstrate how the application and support of 320+ similarity metrics enables significant performance boosts to cluster algorithms. Through application of a approach for run time filtered computation of dense similarity matrix, this paper manages to address issues known to hamper fast and accurate analysis of many dimensional data sets, as described in Section III. While our proposed approach manages to reduce execution time by a factor of 600x+ (Figure 1), the memory consumption is reduced by a factor of $10^7 x$.

This paper demonstrates how the application of high performance implementation strategies enables a boost in prediction

accuracy and execution time. The combination of results from Figure 1 and Figure 2 reveals how the application of accurate similarity metrics combined with low level implementation may significantly improve prediction accuracy of data mining. In the measurement Section V we have demonstrated how accurate parameter identification of *DBSCAN* settings and similarity metrics enables *DBSCAN* to predict clusters which a high degree of prediction accuracy, *i.e.*, when compared to default approaches. Therefore, *DBSCAN* offers the potential for accurate identification of clusters.

In this work we have addressed a shortfall with respect to established approaches. To enable accurate computation of similarity without being constrained by memory thresholds and unfeasible execution time of current approaches. Therefore, we assert that the software and methodology for accurate and high performance data mining, presented in this paper, may improve a number of established approaches in data analysis.

A. Future Work

A challenge in application of *DBSCAN* concerns the need to identify accurate configuration thresholds. We plan to address the weakness of automated parameter identification in *DBSCAN*, *i.e.*, to provide a methodology and software for accurate and fast evaluation of threshold parameters.

ACKNOWLEDGEMENTS

The authors would like to thank MD K.I. Ekseth at UIO, Dr. O.V. Solberg at SINTEF, Dr. S.A. Aase at GE Healthcare, MD B.H. Helleberg at NTNU–medical, Dr. Y. Dahl, Dr. T. Aalberg, Dr. J.C. Meyer, and K.T. Dragland at NTNU, and the High Performance Computing Group at NTNU for their support.

REFERENCES

[1] Peel, L., Larremore, D.B., Clauset, A.: The ground truth about metadata and community detection in networks. Science Advances **3**(5), 1602548 (2017)

[2] Parsons, L., Haque, E., Liu, H.: Subspace clustering for high dimensional data: a review. Acm Sigkdd Explorations Newsletter **6**(1), 90–105 (2004)

[3] Narayanan, R., Ozisikyilmaz, B., Zambreno, J., Memik, G., Choudhary, A.: Minebench: A benchmark suite for data mining workloads. In: Workload Characterization, 2006 IEEE International Symposium On, pp. 182–188 (2006). IEEE

[4] Consortium, U., *et al.*: Uniprot: the universal protein knowledgebase. Nucleic acids research **45**(D1), 158–169 (2017)

[5] Mekkat, V., Natarajan, R., Hsu, W.-C., Zhai, A.: Performance characterization of data mining benchmarks. In: Proceedings of the 2010 Workshop on Interaction Between Compilers and Computer Architecture, p. 11 (2010). ACM

[6] Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W., et al.: The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley (2006)

[7] Lu, S.-L., Karnik, T., Srinivasa, G., Chao, K.-Y., Carmean, D., Held, J.: Scaling the x201C memory wall x201D designer track. In: Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference On, pp. 271–272 (2012)

[8] Lloyd, S.: Least squares quantization in pcm. IEEE transactions on information theory **28**(2), 129–137 (1982)

[9] Sibson, R.: Slink: an optimally efficient algorithm for the single-link cluster method. The computer journal **16**(1), 30–34 (1973)

[10] Enright, A.J., Van Dongen, S., Ouzounis, C.A.: An efficient algorithm for large-scale detection of protein families. Nucleic Acids Res **40**, 1575–1584 (2002)

[11] Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software (TOMS) **3**(3), 209–226 (1977)

[12] Hamerly, G.: Making k-means even faster. In: Proceedings of the 2010 SIAM International Conference on Data Mining, pp. 130–140 (2010). SIAM

[13] Sculley, D.: Web-scale k-means clustering. In: Proceedings of the 19th International Conference on World Wide Web, pp. 1177–1178 (2010). ACM

[14] Arthur, D., Vassilvitskii, S.: k-means++: The advantages of careful seeding. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1027–1035 (2007). Society for Industrial and Applied Mathematics

[15] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., *et al.*: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd, vol. 96, pp. 226–231 (1996)

[16] Ole Kristian Ekseth: hpLysis: a high-performance software-library for big-data machine-learning. https://bitbucket.org/oekseth/hplysis-cluster-analysis-software/. Online; accessed 06. June 2017

[17] Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics **20**, 53–65 (1987)

[18] Shah, G.H., Bhensdadia, C., Ganatra, A.P.: An empirical evaluation of density-based clustering techniques. International Journal of Soft Computing and Engineering (IJSCE) ISSN **22312307**, 216–223 (2012)

[19] Kockara, S., Mete, M., Chen, B., Aydin, K.: Analysis of density based and fuzzy c-means clustering methods on lesion border extraction in dermoscopy images. BMC bioinformatics **11**(6), 26 (2010)

[20] Michael Hahsler: A C-package-library for efficient DB-SCAN computation. https://github.com/mhahsler/dbscan. Online; accessed 06. June 2017

[21] Andrade, G., Ramos, G., Madeira, D., Sachetto, R., Ferreira, R., Rocha, L.: G-dbscan: A gpu accelerated algorithm for density-based clustering. Procedia Computer Science **18**, 369–378 (2013)

[22] Otair, M.: Approximate k-nearest neighbour based spatial clustering using kd tree. International Journal of Database Management Systems **5**(1), 97 (2013)

[23] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., *et al.*: Scikit-learn: Machine learning in python. Journal of Machine Learning Research **12**(Oct), 2825–2830 (2011)

[24] Drepper, U.: What every programmer should know about memory. Red Hat, Inc **11**, 2007 (2007)

[25] Hoare, C.A.: Quicksort. The Computer Journal **5**(1), 10–16 (1962)

[26] Jain, A.K.: Data clustering: 50 years beyond k-means. Pattern recognition letters **31**(8), 651–666 (2010)

[27] Dunn, J.C.: Well-separated clusters and optimal fuzzy partitions. Journal of cybernetics **4**(1), 95–104 (1974)

[28] Caliński, T., Harabasz, J.: A dendrite method for cluster analysis. Communications in Statistics-theory and Methods **3**(1), 1–27 (1974)

[29] Rand, W.M.: Objective criteria for the evaluation of clustering methods. Journal of the American Statistical association **66**(336), 846–850 (1971)

[30] Yeung, K.Y., Ruzzo, W.L.: Details of the adjusted rand index and clustering algorithms, supplement to the paper an empirical study on principal component analysis for clustering gene expression data. Bioinformatics **17**(9), 763–774 (2001)

[31] Reshef, D.N., Reshef, Y.A., Finucane, H.K., Grossman, S.R., McVean, G., Turnbaugh, P.J., Lander, E.S., Mitzenmacher, M., Sabeti, P.C.: Detecting novel associations in large data sets. science **334**(6062), 1518–1524 (2011)

 ISBN: 978-1-61208-612-5