Contraction Clustering (RASTER)

A Big Data Algorithm for Density-Based Clustering in Constant Memory and Linear Time

Gregor Ulm, Emil Gustavsson, and Mats Jirstrand

Fraunhofer-Chalmers Research Centre for Industrial Mathematics,
Chalmers Science Park, SE-412 88 Gothenburg, Sweden
{gregor.ulm, emil.gustavsson, mats.jirstrand}@fcc.chalmers.se
http://fcc.chalmers.se

Abstract. Clustering is an essential data mining tool for analyzing and grouping similar objects. In big data applications, however, many clustering methods are infeasible due to their memory requirements or runtime complexity. Contraction Clustering (RASTER) is a linear-time algorithm for identifying density-based clusters. Its coefficient is negligible as it depends neither on input size nor the number of clusters. Its memory requirements are constant. Consequently, RASTER is suitable for big data applications where the size of the data may be huge. It consists of two steps: (1) a contraction step which projects objects onto *tiles* and (2) an agglomeration step which groups tiles into clusters. Our algorithm is extremely fast. In single-threaded execution on a contemporary workstation, it clusters ten million points in less than 20 seconds — when using a slow interpreted programming language like Python. Furthermore, RASTER is easily parallelizable.

Keywords: algorithms, big data, machine learning, unsupervised learning, clustering

1 Introduction

The goal of clustering is to aggregate similar objects into groups in which objects exhibit similar characteristics. When attempting to cluster very large amounts of data, i.e. data in excess of 10^{12} elements [10], two limitations of many well-known clustering algorithms become apparent. First, they operate under the premise that all available data fits into memory, which does not necessarily hold in a big data context. Second, their time complexity is unfavorable. For instance, one of the most used clustering algorithms is DBSCAN [8]. It runs in $\mathcal{O}(n \log n)$ at best, where n stands for the number of objects. There are standard implementations that use a distance matrix, but its space requirements of $\mathcal{O}(n^2)$ make big data applications infeasible. In addition, the logarithmic factor is problematic in a big data context. Linear-time clustering methods have been described, for instance in surveys by Kumar et al. [12,13], but those have large coefficients, which makes them inapplicable to big data clustering.

In this paper, we introduce CONTRACTION CLUSTERING (RASTER).[1] In the taxonomy presented by Fahad et al. [9], it is a grid-based clustering algorithm. RASTER has been designed for clustering big data. It scales linearly in the size of its input. In addition, it is able to handle cases where the available data do not fit into memory as its memory requirements are constant. Its two distinctive phases are parallelizable; the most computationally intensive first one trivially so, while the second one can be expressed in the divide-and-conquer paradigm of algorithm design. In a review paper, Shirkhorshidi et al. [19] state that distributed clustering algorithms are needed for efficiently clustering big data. However, our algorithm is a counter example to that claim as it exhibits excellent performance metrics even in single-threaded execution on a single machine.

The novelty of RASTER is that it is a straight-forward, easy to implement, and extremely fast big data clustering algorithm. It requires only one pass through the input data and does not need to retain its input. In addition, key operations like projecting to tiles and neighborhood lookups are performed in constant time. Another benefit of our approach is that the parameters required are intuitive, and that there is no need to compute distances between points. Furthermore, it is easily parallelizable.

The remainder of this paper is organized as follows. Section 2 contains the problem description, followed by a detailed description of RASTER in Section 3. In Section 4 we discuss results. Related work is covered in Section 5. Lastly, we outline possible future work in Section 6.

2 Problem Description

In this section we give a brief overview of the clustering problem (2.1), describe the motivating use case behind RASTER (2.2), and highlight limitations of common clustering methods (2.3).

2.1 The Clustering Problem

Clustering is a standard approach in machine learning for grouping similar items, with the goal of dividing a dataset into subsets that share certain features. It is an example of unsupervised learning, which implies that there are many valid ways of clustering data points. Elements belonging to a cluster are normally more similar to other elements in it than to elements in any other cluster. If an element does not belong to a cluster, it is classified as noise. An element normally only belongs to at most one cluster. However, fuzzy clustering methods [17, 22] can identify non-disjoint clusters, meaning that elements may be part of multiple overlapping clusters. Yet, this is not an area we are concerned with in this paper.

[1] The chosen shorthand may not be immediately obvious: RASTER operates on an implied grid. Resulting clusters can be made to look similar to the dot matrix structure of a *raster graphics* image. Furthermore, the name *RASTER* is an agglomerated contraction of the words *contraction* and *clustering*.

Instead, we focus on problems that are in principle solvable by common clustering methods such as DBSCAN [8] or k-means clustering [16].

2.2 Motivating Use Case

Large-scale processing of telemetry data in the form of GPS coordinates is required for the analysis of vehicle transportation networks. A central problem in that domain is the identification of *hubs*, which are locations within a road network at which vehicles stop so that a particular action can be performed, e.g. warehouses, delivery points, or bus stops. Conceptually, a hub is a node in such a network, and a node is the center of a cluster.

GPS coordinates are provided as coordinate pairs. Their precision is defined by the number of place values after the decimal point. For instance, the coordinate pair (40.748441, -73.985664) specifies the location of the Empire State Building in New York City with a precision of 11.1 centimeters. A seventh decimal place value would specify a given location with a precision of 1.1 centimeters, while truncating to five decimal place values would lower precision to 1.1 meters. High-precision GPS measurements require special equipment, while consumer-grade GPS is accurate to within about ten meters under open sky [7]. Thus, for the purpose of clustering, lower-precision GPS coordinates could be used, without losing a significant amount of information. This is the key insight that led to the discovery of RASTER.

2.3 Limitations of Common Clustering Methods

In this subsection we briefly describe why two standard clustering algorithms, DBSCAN and k-means clustering, are not suitable for our big data clustering use case.

DBSCAN identifies density-based clusters. Its time complexity is $\mathcal{O}(n \log n)$ in the best case. This depends on whether a query identifying the neighbors of a particular data point can be performed in $\mathcal{O}(\log n)$. In a pathological case, or in the absence of a fast lookup query, its time complexity is $\mathcal{O}(n^2)$. DBSCAN is comparatively fast. However, when working with many billions or even trillions of data points, clustering becomes infeasible due to the logarithmic factor, provided all data even fits into memory.[2]

In k-means clustering, the number of clusters k, has to be known in advance. The goal is to determine k partitions of the input data. Two aspects make k-means clustering less suitable for our use case. First, when dealing with big data, estimating a reasonable k is non-trivial. Second, its time complexity is unfavorable. An exact solution requires $\mathcal{O}(n^{dk+1})$, where d is the number of dimensions [11]. Lloyd's algorithm [15], which uses heuristics, is likewise not applicable as its time complexity is $\mathcal{O}(dnki)$, where i is the number of iterations

[2] On a contemporary workstation with 16 GB RAM, the `scikit-learn` implementation of DBSCAN cannot even handle one million data points.

until convergence is reached. There have been recent improvements [2, 4], but their time complexity still seems unfavorable for huge data sets.[3]

3 RASTER

This section contains a thorough presentation of RASTER, consisting of a high-level description of the algorithm (3.1), a discussion of the concept of *tiles* and their role in creating clusters (3.2), a detailed description of the algorithm, including its time complexity (3.3), an outline of one possible parallelization strategy (3.4), brief remarks on using data of higher dimensionality (3.5), and a potential weakness of our algorithm, including a workaround (3.6).

3.1 High-Level Description

The goal of RASTER is to reduce a very large number n of 2-dimensional points to a more manageable number of points that specify the *approximate area* of clusters in the input data, without retaining its input data. Fig. 1 provides a visualization. Our algorithm uses an implicit 2-dimensional grid of a coarser resolution than the input data. Each square of this grid is referred to as a *tile*; each point point is mapped to exactly one tile. A tile containing at least a user-specified threshold number t of observations is labeled as a *significant tile*. Afterwards, *RASTER clusters* are constructed from adjacent significant tiles.

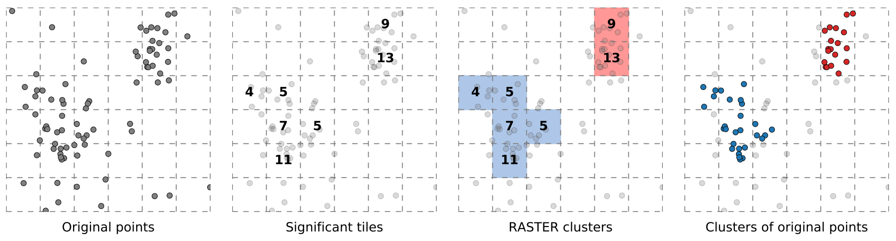

Fig. 1: A visualization of RASTER

3.2 Tiles and RASTER Clusters

A key component of RASTER is the deliberate reduction of the precision of its input data. In general, this operation is a projection of points to tiles, which could, for instance, be achieved by truncating or rounding. The goal is the identification of RASTER clusters, which is attained via two distinct and consecutive steps: *contraction* and *agglomeration*. The contraction step first determines the number of observations per tile, and then discards all non-significant tiles. The agglomeration step constructs RASTER clusters out of adjacent significant

[3] We have identified tens of thousands of clusters with RASTER in a huge real-world data set, which shows that k-means clustering would have been *highly* unsuitable.

tiles. To illustrate the idea of a *tile*, consider a grid consisting of squares of a fixed side length. A square may contain several coordinate points. Reducing the precision by one decimal digit means removing the last digit of a fixed-precision coordinate. For instance, with a chosen decimal precision of 2, the coordinates $(1.005, 1.000)$, $(1.009, 1.002)$, and $(1.008, 1.006)$ are all truncated (*contracted*) to the tile identified by the corner point $(1.00, 1.00)$. Thus, $(1.00, 1.00)$ is a tile with three associated points. This tile would be classified as significant when using a threshold value of $t \geq 3$ and disregarded otherwise.

In the subsequent *agglomeration* step, RASTER clusters are constructed, which consist of significant tiles that are at most a Manhattan distance of δ steps apart. A value of $\delta = 1$ means that significant tiles need to be directly adjacent; with a larger δ, a cluster could contain significant tiles without direct neighbors. Finally, constructing a RASTER cluster is an iterative process that agglomerates significant tiles, taking δ into account.

3.3 The Algorithm

In this subsection we present some explanations that accompany the RASTER pseudocode in Fig. 2.[4] The algorithm consists of three sequential loops. The first two *for*-loops constitute the *contraction* step. The subsequent *while*-loop constitutes the *agglomeration* step.

Mapping to a tile consists of associating a data point p to a value representing a tile. The case of 2-dimensional values was illustrated in the previous subsection. RASTER does not exhaustively check every possible tile value, but instead only retains tiles that were encountered while processing data. Due to the efficiency of hash tables, the first *for*-loop runs in $\mathcal{O}(n)$, where n is the number of points. The mapping function is performed in $\mathcal{O}(1)$, which is also the time complexity of the various hash table operations we use. After the first *for*-loop all points are mapped to a tile. The second *for*-loop traverses all keys of the hash table *tiles*. Only significant tiles are retained. The intermediate result of this loop is a hash table of significant tiles and their respective number of observations. Deleting an entry is an $\mathcal{O}(1)$ operation. At most, and only in the pathological case where there is exactly one observation per tile, there are n tiles. In any case, it holds that $m \leq n$, where m is the number of tiles. Thus, this step of the algorithm is performed in $\mathcal{O}(m) \leq \mathcal{O}(n)$.

The subsequent *while*-loop performs the *agglomeration* step, which constructs clusters from significant tiles. The pseudocode does not specify the definition of clusters, but implies that a cluster is either a set of significant tiles, or a set of tuples, where each tuple consists of the coordinates of a significant tile and the total number of observations. There are at most n tiles. In order to determine the tiles a cluster consists of, take one tile from the set *tiles* and recursively determine all neighboring tiles in $\mathcal{O}(n)$ in a depth-first manner. This is conceptually similar to the well-known flood fill algorithm. Neighborhood lookup with a hash table

[4] Reference implementations in several programming languages are available at `https://gitlab.com/fraunhofer_chalmers_centre/contraction_clustering_raster`.

is in $\mathcal{O}(1)$, as the locations of all its neighboring tiles are known. For instance, when performing agglomerations with $\delta = 1$, the neighbors of any coordinate pair (x, y) are the squares to its left, right, top, and bottom in the grid. Thus, the third loop runs likewise in $\mathcal{O}(m) \leq \mathcal{O}(n)$. Each of the three loops runs in $\mathcal{O}(n)$ in the worst case, leading to a total time complexity of $\mathcal{O}(n)$.

The result is a set C of sets, where each $c \in C$ is a cluster of a finite amount of points that each uniquely identify a significant tile. An illustration of how such a cluster may look like is given in Fig. 1, which shows the count of observations per tile and, considering a threshold value of $t = 4$, highlights those that are classified as significant tiles and agglomerated into clusters.

```
input: data points, threshold t,
    distance δ
output: collection of clusters clusters

                                  ▷ initialization
initialize hash table tiles, set clusters
                                  ▷ contraction
for point in points do
    map point to corresponding tile
    if tile $\notin$ tiles then
        add tile with value 1 to tiles
    else
        increment value for tile by 1
    end if
end for
for tile in tiles do
    if value for tile $< t$ then
        remove tile from tiles
    end if
end for
                                  ▷ agglomeration
while tiles $\neq \emptyset$ do
    select arbitrary tile from tiles
    determine cluster c containing tile,
        considering δ
    remove tiles in c from tiles
    add c to clusters
end while
```

Fig. 2: RASTER

According to our specification, the result is a set of sets, where each set is a cluster that is specified only by its constituent significant tiles. It is trivial to modify RASTER to retain either all points of a cluster, or the count of observations per tile. Indeed, one key element of our algorithm is that information regarding data points per tile is only maintained as an aggregate by keeping track of their sum. This is a useful approach when clustering very large amounts of data. However, if data fits into memory, one could as well retain all points per tile or all unique points per tile.

3.4 Parallel RASTER

RASTER is easily parallelizable. An obvious target is mapping to tiles, which is *embarrassingly parallel* in nature as no projection depends on the result of any other projection. Thus, this step could be part of a separate parallel *for*-loop. Updating the hash table *tiles* could be sped up with a concurrent hash table, which affects the first two loops of the algorithm. The agglomeration step could be parallelized via the divide-and-conquer paradigm of algorithm design by subdividing the input into partitions of equal size and recursively processing them. Fig. 3 shows how partitions can be processed in parallel. However, adjacent RASTER clusters c_1 and c_2 that cross the boundaries of a partition need to be joined. A cluster is considered complete once none of its neighbors is a significant tile and it does not touch any partition boundaries.

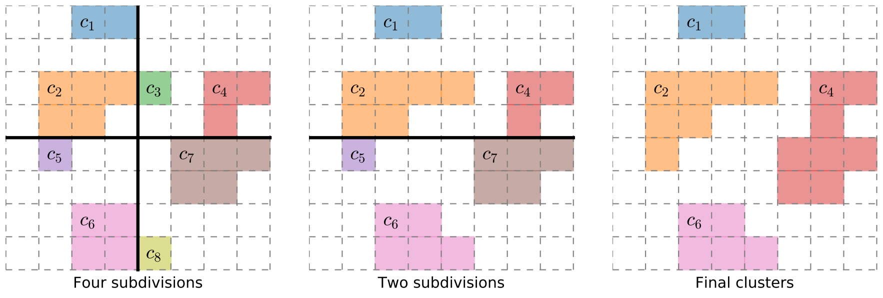

Fig. 3: The agglomeration step as a divide-and-conquer algorithm

3.5 Generalizing to Higher Dimensions

While we have focused on 2-dimensional geospatial data, it is possible to generalize RASTER to d dimensions. Generalizing from $\mathbb{R}^2$, irrespective of dimension, a similar case can be constructed for $\mathbb{R}^n$. For the former, the reduction for each decimal value is 10^2 per tile. For the latter, it is 10^n. In the case of $\delta = 1$, the number of neighbors to consider per tile is $2d$.

3.6 Minimum Cluster Size in Disadvantageous Grid Layouts

We consider truncation of a fixed number of decimal digits to be the standard behavior of RASTER. As long as mapping to tiles is performed in a consistent manner, any mapping can be chosen. Yet, for any possible mapping a corner case can be found that illustrates that a significant tile may not be found. Thus, the identification of significant tiles may depend on the chosen grid.

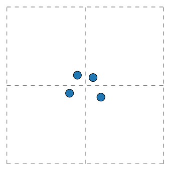

Fig. 4: Any RASTER grid is vulnerable

Assume a threshold of $t = 4$ for a significant tile, and a tile with four points. If all points were located in the same tile of a grid, a significant tile would be detected. However, those four points could also be spread over adjacent tiles, as illustrated by Fig. 4. One could of course shift the grid by choosing a different projection, but an adversary could easily place all points on different tiles in the new grid. In order to alleviate this problem, a threshold $u < t$ for the number of observations in a tile needs to be picked. To be on the safe side, a value of $u = \frac{t}{4}$ is recommended. Alternatively, an additional step could be added to make RASTER more precise. With a complexity of $\mathcal{O}(m)$, where m is the number of tiles, one could determine whether a group of four tiles contains at least t points.[5]

[5] In case it is not obvious why this is in linear time: For each row in a grid, take the current and next row into account. Start, for instance, with the tile in the top left

4 Results

In this section we present results of executing RASTER. A visualization of the use case this algorithm was designed for (4.1) is followed by results with sample datasets for density-based clustering algorithms (4.2). Afterwards, we discuss empirical runtime measurements, including a comparison with DBSCAN (4.3).

The default implementation of RASTER does not retain its input data, but instead keeps track of the number of observations per tile. That number is dropped when agglomerating significant tiles to clusters. This leads to clusters of significant tiles, which would not have been ideal for visualizations. Therefore, the results in this section were achieved with a trivially modified version of our algorithm that retains all points per significant tile. For $\delta = 2$, we made a minor modification to RASTER to take tiles forming a square around any coordinate into account, instead of the Manhattan distance. Lastly, as an added post-processing step, we disregarded clusters containing less than certain threshold number of data points, which is specified in each case below.

In the figures in this section, the entirety of the input data is visualized as translucent gray points. Clusters in the colors red and blue were plotted on top of those gray points. Thus, all remaining non-red and non-blue points are noise.

4.1 Ideal Data

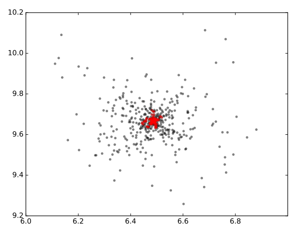

Fig. 5: RASTER applied to ideal data

RASTER was designed for the identification of centers of groups of points which are placed increasingly tightly the closer they are to the center of a cluster. In Fig. 5, we use an artificial data set, where a roughly Gaussian distribution of points is spread around a center point. This is similar to patterns encountered in the motivating use case we described in Section 2.2. Points plotted in red constitute the resulting RASTER cluster. In order to generate a larger cluster, we chose a precision of 2 decimal values when mapping to tiles and the parameters $t = 2$, $\delta = 2$. We disregarded clusters that did not contain at least 10 elements.

4.2 Sample Datasets

Even though RASTER was specifically designed for the pattern shown in Fig. 5, it also performs well as a general-purpose density-based clustering algorithm. We illustrate this by showing the results of applying RASTER to two standard `scikit-learn` data sets: `noisy_circles` and `noisy_moons`, which both contain 1500 points. For the experiments below, we tried several parameters. We did not preprocess the input data, for instance by rescaling coordinates.

corner and take its right neighbor as well as the two tiles adjacent in the next row into account.

In the `noisy_circles` data set shown in Fig. 6, RASTER identifies two distinct clusters, but ignores part of the outer circle. The chosen parameters were $t = 3$ and $\delta = 1$. The chosen precision for mapping to tiles was one decimal value. Resulting clusters had to contain more than 25 data points. In the `noisy_moons` data set, density-based clusters are likewise reliably identified. Fig. 7 shows output similar to `noisy_circles`, as the lower arc in the image is not fully included in the identified cluster. The chosen parameters were $t = 7$ and $\delta = 2$, as well as a precision of 1 for mapping to tiles. Clusters needed to have a size greater than 15. With minor parameter tweaking of t the shape of the lower arc could be clearly identified, as shown in Fig. 8, but at the cost of adding a minimal amount of noise to the cluster containing the upper arc.

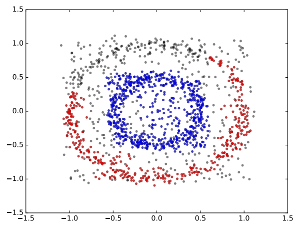

Fig. 6: RASTER applied to `noisy circles`

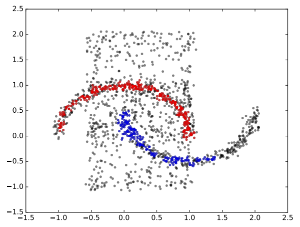

Fig. 7: RASTER applied to `noisy moons` with threshold $t = 7$

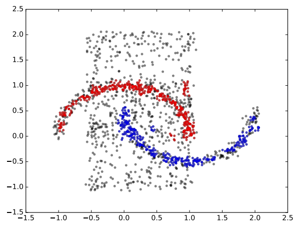

Fig. 8: RASTER applied to `noisy moons` with threshold $t = 6$

4.3 Empirical Runtime

RASTER is a fast clustering algorithm, running in linear time. In order to quantify this, we performed several measurements for two variations of our algorithm, i.e. one that retains all unique points and one that retains a count of the number of observations per significant tile. The experiments were run on an Oracle VirtualBox virtual machine, hosting Ubuntu Linux 16.04. We allocated 16 GB RAM. The CPU was an Intel i7-6700K clocked at 4.0 GHz. The implementation was executed by a Python 3.5 interpreter. We generated ideal data as described in Section 4.1. The chosen parameters were $t = 5$ and $\delta = 1$. No post-processing to filter out clusters below a certain minimum size was performed. The resulting clusters were rather dense. The reason behind that choice was to cause a higher workload for the agglomeration step. When processing real-world data with many small and dense clusters, agglomeration would merely add a negligible cost, due to the fact that clusters are very dense but also quite small.

The difference between both variations of RASTER is modest. To highlight one of the measurements illustrated by Fig. 9: ten million points are processed in 17.8 versus 23.1 seconds, while the slower runtime was achieved when retaining all data points. For an input size of 10^8 points, memory was insufficient for the version of RASTER that retains its input. Memory requirements when aggregating data are bounded by the number of tiles in a grid and the size of the used data types for storing the count of observations per tile, which are both finite with GPS data. Thus, memory requirements are constant. We also compared RASTER, in the variation that retains data points, with the DBSCAN implementation of `scikit-learn` 0.18.1, cf. Fig. 10. As parameters, we set $\epsilon = 0.3$ and the minimum sample size to 10. The DBSCAN experiment ended prematurely as it ran out of memory with one million points.

5 Related Work

An early approach to grid-based spatial data mining was STING [20]. A key difference of that algorithm is that it performs statistical queries, using distributions of attribute values.

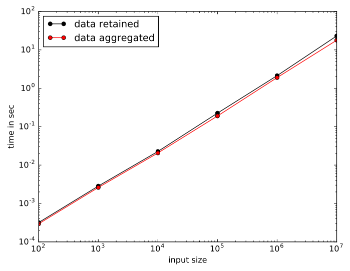

Fig. 9: Two variations of RASTER

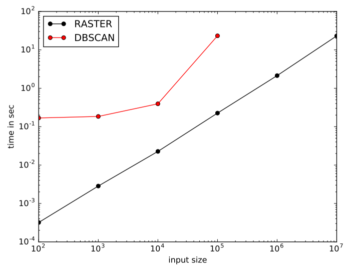

Fig. 10: RASTER vs DBSCAN

WaveCluster [18] shares some similarities with RASTER. It can reduce the resolution of the input, which leads to output that is visually similar to RASTER clusters that are defined by its significant tiles. WaveCluster runs in linear time. However, because the computation of wavelets is costlier than the operations RASTER performs, its empirical runtime is presumably worse.

There are also conceptual similarities between a sub-method of CLIQUE [1] and RASTER. Yet, differences are that the former is mainly concerned with subspace clustering of high-dimensional data. Further, RASTER is more flexible with regards to neighborhood lookup. It is also a faster operation with our algorithm. A minor difference is that CLIQUE uses a density-ratio threshold. Lastly, the empirical runtime of RASTER can be assumed to be lower, largely due to the very cheap cost of projection onto tiles and neighborhood lookup.

The idea of counting observations within a grid has been explored by Baker and Valleron [3]. They presented a solution to a problem in spatial epidemiology whose initial step seems similar to the contraction step performed by RASTER.

The approach taken by GRPDBSCAN, discovered by Darong and Peng [6], seems to have aspects in common with the RASTER variant that retains all input data. Their description is not fleshed out, however, so it is not clear how similar their algorithm is to ours.

Lastly, there is some similarity between our algorithm and blob detection in image analysis [5]. A direct application of that method to finding clusters would arguably require very dense cluster centers, which could be achieved by mapping points to fewer tiles. Also, blob detection is computationally more expensive than RASTER.

6 Future Work

There are several ways to build upon RASTER. One obvious direction would be arbitrary resolutions in the reduction step via rounding. Currently, reductions truncate by one decimal place value. Smaller reduction steps are easy to implement. However, RASTER is more general and could use any projection, so it may be worth exploring different areas of application.

Xiaoyun et al. introduced GMDBSCAN [21], a DBSCAN-variant that is able to detect clusters of different densities. The inability to detect clusters of different densities is a weakness of DBSCAN that is shared by RASTER. For more general purpose-applications, it may be worth investigating a similar approach for RASTER as well. A starting point is an adaptive distance parameter for significant tiles. While this paper only considered fixed values of 1 and 2 for δ, one could certainly consider arbitrary values instead.

RASTER does not distinguish between significant tiles. Yet, one could think of cases in which some of those tiles contain a very large number of observations, while others barely reach the specified threshold value. Thus, one could consider an adaptive approach to RASTER-clustering, for instance by subdividing such tiles into smaller segments, with the goal of determining more accurate cluster shapes. This idea is related to adaptive mesh refinement, suggested by Liao et. al [14]. A related idea is to change the behavior of RASTER when detecting a large number of adjacent tiles that have not been classified as significant. This may prompt a coarsening of the grid size for that part of the input space.

For practical use, it may be worthwhile to add a contextual relaxation value ϵ for the threshold value of significant tiles. For instance, in the vicinity of several significant tiles, a neighboring tile with $t - \epsilon$ observations may be considered part of the agglomeration, in particular if it has multiple significant tiles as neighbors.

Acknowledgments

This research was supported by the project *Fleet telematics big data analytics for vehicle usage modeling and analysis (FUMA)* in the funding program *FFI: Strategic Vehicle Research and Innovation (DNR 2016-02207)*, which is administered by VINNOVA, the Swedish Government Agency for Innovation Systems.

References

1. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: International Conference on Management of Data. vol. 27, pp. 94–105. ACM (1998)
2. Bachem, O., Lucic, M., Hassani, H., Krause, A.: Fast and provably good seedings for k-means. In: Advances in Neural Information Processing Systems. pp. 55–63 (2016)
3. Baker, D.M., Valleron, A.J.: An open source software for fast grid-based datamining in spatial epidemiology (fgbase). International journal of health geographics 13(1), 46 (2014)

4. Capó, M., Pérez, A., Lozano, J.A.: An efficient approximation to the k-means clustering for massive data. Knowledge-Based Systems 117, 56–69 (2017)
5. Danker, A.J., Rosenfeld, A.: Blob detection by relaxation. IEEE Transactions on Pattern Analysis and Machine Intelligence (1), 79–92 (1981)
6. Darong, H., Peng, W.: Grid-based dbscan algorithm with referential parameters. Physics Procedia 24, 1166–1170 (2012)
7. van Diggelen, F., Enge, P.: The world's first gps mooc and worldwide laboratory using smartphones. In: Proceedings of the 28th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2015). pp. 361–369. ION (2015)
8. 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)
9. Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Khalil, I., Zomaya, A.Y., Foufou, S., Bouras, A.: A survey of clustering algorithms for big data: Taxonomy and empirical analysis. IEEE transactions on emerging topics in computing 2(3), 267–279 (2014)
10. Hathaway, R.J., Bezdek, J.C.: Extending fuzzy and probabilistic clustering to very large data sets. Computational Statistics & Data Analysis 51(1), 215–234 (2006)
11. Inaba, M., Katoh, N., Imai, H.: Applications of weighted voronoi diagrams and randomization to variance-based k-clustering: (extended abstract). In: Proceedings of the Tenth Annual Symposium on Computational Geometry. pp. 332–339. SCG '94, ACM, New York, NY, USA (1994)
12. Kumar, A., Sabharwal, Y., Sen, S.: Linear time algorithms for clustering problems in any dimensions. In: International Colloquium on Automata, Languages, and Programming. pp. 1374–1385. Springer (2005)
13. Kumar, A., Sabharwal, Y., Sen, S.: Linear-time approximation schemes for clustering problems in any dimensions. Journal of the ACM (JACM) 57(2), 5 (2010)
14. Liao, W.k., Liu, Y., Choudhary, A.: A grid-based clustering algorithm using adaptive mesh refinement. In: 7th Workshop on Mining Scientific and Engineering Datasets of SIAM International Conference on Data Mining. pp. 61–69 (2004)
15. Lloyd, S.: Least squares quantization in pcm. IEEE transactions on information theory 28(2), 129–137 (1982)
16. MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. vol. 1, pp. 281–297. Oakland, CA, USA. (1967)
17. Pham, D.L.: Spatial models for fuzzy clustering. Computer vision and image understanding 84(2), 285–297 (2001)
18. Sheikholeslami, G., Chatterjee, S., Zhang, A.: Wavecluster: A multi-resolution clustering approach for very large spatial databases. In: VLDB. vol. 98, pp. 428–439 (1998)
19. Shirkhorshidi, A.S., Aghabozorgi, S., Wah, T.Y., Herawan, T.: Big data clustering: a review. In: International Conference on Computational Science and Its Applications. pp. 707–720. Springer (2014)
20. Wang, W., Yang, J., Muntz, R., et al.: Sting: A statistical information grid approach to spatial data mining. In: VLDB. vol. 97, pp. 186–195 (1997)
21. Xiaoyun, C., Yufang, M., Yan, Z., Ping, W.: Gmdbscan: multi-density dbscan cluster based on grid. In: e-Business Engineering, 2008. ICEBE'08. IEEE International Conference on. pp. 780–783. IEEE (2008)
22. Yang, M.S.: A survey of fuzzy clustering. Mathematical and Computer modelling 18(11), 1–16 (1993)