REVIEW PAPER

AN ANALYSIS OF DENSITY BASED CLUSTERING ALGORITHMS
AND THEIR APPLICATION IN GEO-SPATIAL CLUSTERING

Arman Mann
13BCE00**73**
+91 9787119725
armanmann2@gmail.com

Prof. Sharmila Banu
Assistant Professor (Senior)
+91 9942469321
sharmilabanu.k@vit.ac.in

B. Tech.

Computer Science and Engineering

School of Computer Science & Engineering

VIT
UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
VELLORE ▪ CHENNAI

TABLE OF CONTENTS

Serial No.	Title	Page No.
1	INTRODUCTION	1
	1.1 Abstract	1
	1.2 Theoretical Background	1
2	LITERATURE SURVEY	2
	2.1 DBSCAN	2
	2.2 GDBSCAN	3
	2.3 OPTICS	4
3	METHODOLOGY	5
	3.1 Introduction	5
	3.2 The Visualisations	6
	3.3 The Algorithms	
4	RESULTS AND DISCUSSION	
5	CONCLUSION	
6	REFERENCES	

1. INTRODUCTION

1.1. Abstract

With this paper we, firstly, provide an analysis on the execution and implementation of 3 different density-based clustering algorithms by testing them on a temperature dataset with over a quarter million records, and secondly, we discuss the results obtained and new knowledge gained. We shall be using DBSCAN, GDBSCAN and OPTICS to form clusters of cities based on the surface temperature data recorded at weather stations across the world from 1743 to 2013. The information obtained by applying these algorithms on the data will be used to map and visualise the similarities in temperature as well as the similarities in temperature changes at these locations. This is achieved by forming clusters such that cities with temperature conditions that most closely resemble each other are in the same cluster. Using this information, we can also try to predict the future weather conditions at these locations, as well as observe the effect of historic events and past human activity on the worldwide temperature conditions.

1.2. Theoretical Background

Over the last decade, advances in machine learning have tremendously increased the ease with which large amounts of data can be assimilated to obtain key insights and new information from the knowledge hidden deep in the data. Unsupervised learning allows us to find unknown relationships between different data objects and determine the degree of influence certain attributes and properties have on the values of others. In order to draw associations within data to reach a particular predictive result, the most popular form of analysis today is clustering. Clustering has become a prominent topic for research in data analytics due its various and widespread applications. The advent of general and specialised data clustering algorithms in the last few decades and their extensive use in a wide variety of applications, such as image vision, computational mathematics, medicine, biology, mobile networking, and economic, has led their vast adoption. Clustering seeks to recognise homogenous groups of objects based on their attributes and is essential for the task of classification or class identification. Most clustering algorithms find clusters based on distance or separation between the values of the object's attributes. Such methods have difficulty in discovering arbitrarily shaped

clusters. [1] This led to the development of algorithms based on a notion of density within the clusters. These algorithms form density thresholds around each object to distinguish interesting data items from noise, while representing clusters as dense regions of data points separated from each other by regions of extremely low densities. Such algorithms may be applied to filter out noise or form large clusters with arbitrary shapes. Their application to large spatial databases permits generalisation of the spatial characteristics and extension of spatial attributes of the objects to define non-explicit relations within the object's neighbourhood. This requires clustering algorithms that:

1. Require minimal information about the domain to determine input parameters
2. Can discover clusters that have arbitrary shapes
3. Work efficiently on extremely large databases

2. LITERATURE SURVEY

2.1. DBSCAN (Density Based Spatial Clustering of Application with Noise)

Density-based spatial clustering methods use a density threshold value around each object to separate the relevant items from noise. [2] The first algorithm to rely on this density-based idea of clusters is DBSCAN. It does not need to be specified the number of clusters that are to be obtained from the data. To find clusters, DBSCAN, starts at an arbitrary point and thereafter, iterates through the whole dataset, retrieving all points with respect to ε density-reachable from it. It judges the density around the neighbourhood of the points to be adequately dense if the number of objects within a radius of distance ε from it is greater than *MinPts* number of points and grows high density regions into clusters. At the same time it tags each object either as *core objects*, *border objects*, or *noise*. New clusters are formed around newly added core objects. A cluster formed by the single object is extended by adding objects from its ε-neighbourhood. The process is then repeated for all objects in the ε-neighbourhood of the newly added points, and so on. For border points, no points are reachable, and we move on to the next object in the dataset. This point could later be contained in the ε-neighbourhood of a different point and therefore belong to another cluster. Reaching a core object from a different cluster in the neighbourhood of a different core object results in merging of the 2 clusters these objects belong to. The growth of the cluster stops when all the border points of the

cluster have been visited and there are no more reachable items. [3] In this case, an unprocessed point is visited, allowing for the formation of more clusters. The clusters created by DBSCAN depend only on the parameters, ε and *MinPts*. DBSCAN thus relies on the our ability to select an optimum set of parameters.

DBSCAN clusters follow the following rules:

1. An object can only belong to a cluster if and only if it lies within the ε-neighbourhood of some core object in the cluster.
2. Core objects within the ε-neighbourhood of other core objects must belong to the same clusters.
3. A non-core object within the ε-neighbourhood of some core objects must belong to the same cluster as at least one of those core objects.
4. A non-core object, which does not lie within the ε-neighbourhood of any core object, is labeled as noise.

2.2. GDBSCAN (Generalised Density Based Spatial Clustering of Application with Noise)

GDBSCAN can cluster point objects as well as spatially extended objects according to both, their spatial and their non-spatial attributes. [4] The DBSCAN algorithm introduced above is a specialisation of GDBSCAN, which generalises DBSCAN in two ways. Any notion of neighbourhood of an object may be used as long as the definition of the neighbourhood is based on a binary predicate, which is symmetric and reflexive. [5] Rather than just adding the objects in the neighbourhood of the object, other characteristics such as the non-spatial attributes, like median temperature of a city, may be used for defining the cardinality of that object's neighbourhood neighbourhood. To find clusters, GDBSCAN starts with an arbitrary point and retrieves all other points reachable from that point with respect to its neighbourhood, *NPred,* and the minimum weighted cardinality, *MinWeight*. The *MinWeight* parameter for a group K of objects is true in case $wCard(K) \geq MinCard$, where *wCard* is a pointer to a function returning the weighted cardinality of the dataset. If the object is a core object, the procedure yields a cluster with respect to *NPred* and *MinWeight*. Otherwise, no objects are reachable from it and the object is considered to be noise. Here, noise is the set of objects in the dataset not part of any of the clusters. The same procedure is repeated for each remaining unclassified object. Clusters are labeled using an ordered and countable data type, such as integers, and each object is marked with

a *cluster_id* such that 'UNCLASSIFIED < NOISE < other *cluster_id*s'. Calling of the neighbourhood function returns the *NPred*-neighbourhood of an object in the dataset as a list of objects. Objects which are labeled as NOISE may belong to a cluster if they're reachable from some other object later. If two clusters are formed very close to each other, there may be some objects that belong to them both. If these objects are border objects in both the clusters, they will only be a part of the cluster found first. Apart from such instances, GDBSCAN produces results independent of the order of visiting objects in the dataset. [4]

2.3. OPTICS (Ordering Points To Identify the Clustering Structure)

Although DBSCAN can discover of arbitrarily shaped clusters in noisy data, it heavily depends on the values of its two input parameters. To discover acceptable clusters, the algorithm may have to be run multiple times with different parameters for each run. To overcome this issue OPTICS was proposed. It is possible to extend the DBSCAN algorithm such that several distance parameters are processed at the same time. [6] This enables us to find density-based clusters with unique densities, all at the same time. For consistent results, we have to follow a specific sequence while processing the objects during cluster expansion. We must choose an object that is reachable for the lowest ε value to make certain that the higher density clusters are completed first. The OPTICS algorithm is thus like an extension of the DBSCAN algorithm for all the distance parameters smaller than a reachability distance, ε. Instead of producing clustering results for one pair of parameter values, to represent its clustering structure, it creates an augmented cluster ordering of the data points. While the algorithm does not explicitly assign objects to clusters, it does store the order in which the objects were processed and the information which would be used by DBSCAN to assign cluster memberships. This consists of the core-distance and a reachability-distance for each point.

- The *core-distance* of an object *p* is simply the smallest distance ε' between *p* and an object in its ε-neighbourhood such that *p* would be a core object with respect to ε' if this neighbour is contained in its ε-neighbourhood. Otherwise, the core-distance is UNDEFINED. [6]
- The *reachability-distance* of an object *p* with respect to another object *o* is the smallest distance such that *p* is directly density-reachable from *o* if *o* is a core object. In this case, the reachability-

distance cannot be smaller than the core-distance of *o* because for smaller distances no object is directly density-reachable from *o*. Otherwise, if *o* is not a core object, even at the generating distance ε, the reachability-distance of *p* with respect to *o* is UNDEFINED. The reachability-distance of an object *p* depends on the core object with respect to which it is calculated. [6]

Thus, OPTICS resolves one of DBSCAN's biggest weaknesses, which is, the ability of accurately detecting clusters in data of varying densities by linearly ordering the points of the dataset such that the spatially closest points become neighbours in the ordering sequence.

4. METHODOLOGY

4.1. Introduction

Berkley Earth provides spatial data records of surface temperatures measured between 1743 and 2013, logged at weather and climate stations in cities around the world. This dataset may be downloaded from the data repository at Berkeley Earth [A]. To keep the data from getting too esoteric, only the monthly mean temperature for each city will be used. The dataset, along with the average temperature value and the name of the city also contains an uncertainty measure for each recorded value as well as the latitude - longitude co-ordinates for the city, which will be used to plot the cities on a map and visualise the changes in temperature observed at their locations. We work with Python, utilising the immense computational power the language provides through its third–party packages to facilitate simple and efficient density based clustering on the global temperature dataset.

First, we ready the dataset for clustering. As the dataset contains separate records for each monthly mean temperature value for every city, we start by calculating the average temperature for each year for a city and grouping together the entire temperature dataset with respect to the cities. This results in a data-frame containing a single record for each city with the mean yearly temperatures listed in the columns. We also create a separate list, which stores all of the city names to enable us to keep track of the city’s data in the data-frame. Before clustering, we may also limit the data to contain records from just a sample range of years between 1743 and 2013 or from just a subset of the cities in the database.

4.2. The Visualisations

We visualise the temperature changes in cities over time with 2 animations that were created using Basemap and Matplotlib with Python. The first visualisation plots the cities being observed on a cylindrical projection of the world map. The second plots temperatures for chosen cities as a function of time, with the measured average yearly temperature on the Y-axis and the current year ascending on the X-axis. As the animations move forward in time, the sizes of the city markers change with respect to the deviation of the observed temperature that year from the median temperature for the city. At the same time, the colours of the markers change from blue, for record low temperatures, to red, for record highs, for each city. Since the data contains records starting in the year 1743, we can visualise changes in temperature conditions in any of the cities in our dataset since 1743 all the way up to 2013.

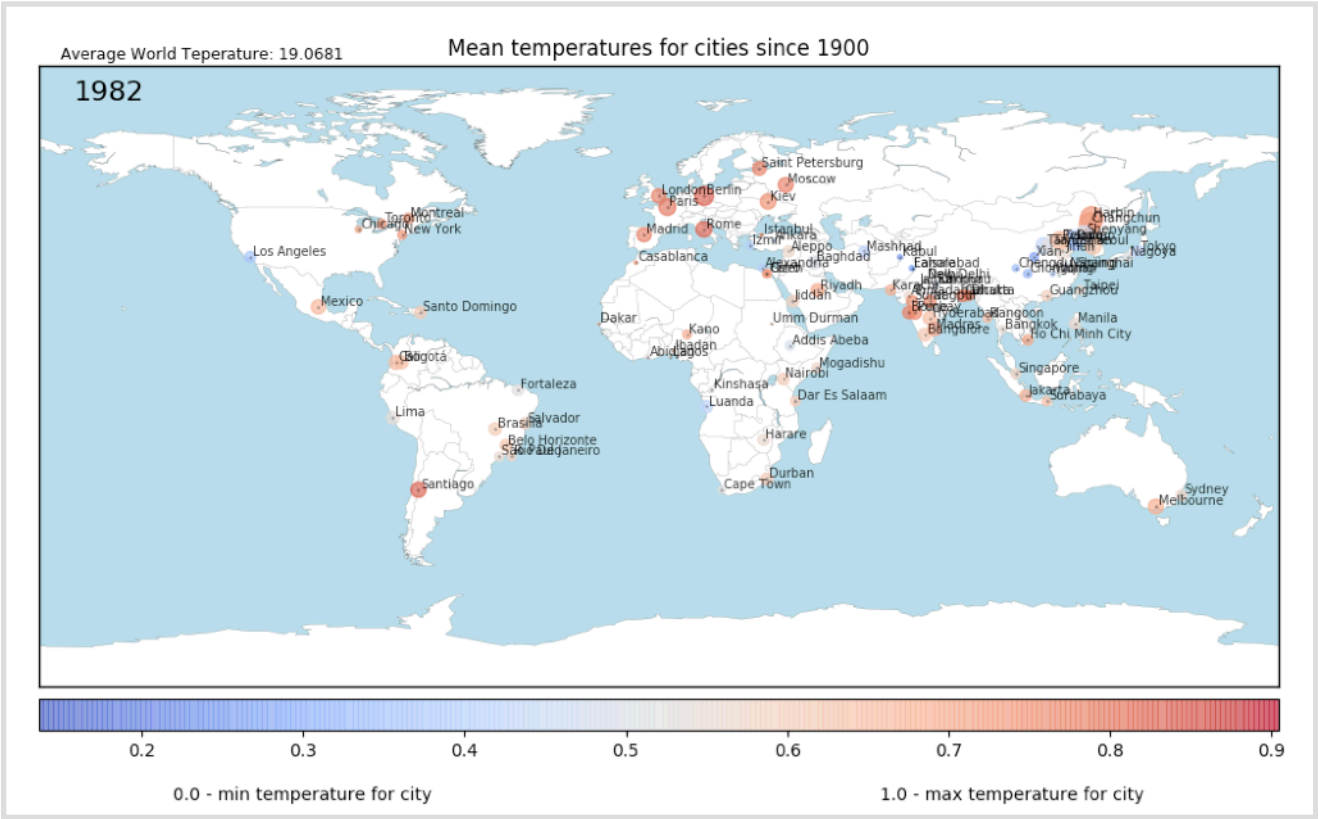

Fig 1.

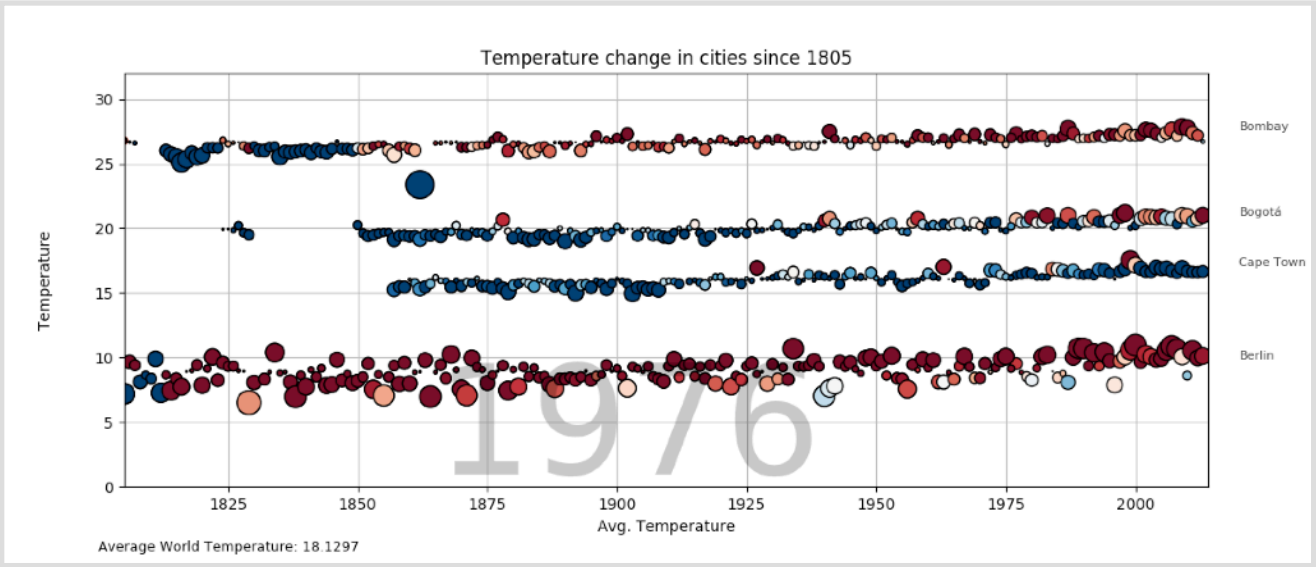

Fig 2.

The Algorithms

- DBSCAN

We define a class of city objects to represent each city in the dataset. Objects of this class have, as attributes, city name, a list of average yearly temperatures, and the mean, maximum and minimum yearly temperatures observed in the city. At the start each object is assigned a cluster_id = UNASSIGNED, since the objects are yet to be clustered. The class also has two member functions. The first is executed to change the cluster id of the class objects and is called by the Expand_Cluster() method when assigning cluster memberships. The second is called to display the results of executing the clustering algorithm. At the same time, the display function also prints out the clusters on the user's screens along with the member's descriptions. Apart from DBSCAN() and Expand_Cluster(), we have also defined two helper methods. These methods are used by the two functions above, supporting them in 2 distinct tasks. The first helper function is the distance function. It takes 2 objects as its arguments, and based on the values in list of temperatures for each object, returns the euclidean distance between them. The second helper function is used to find the ε-neighbourhood for an object. It takes the object and a threshold distance, ε as arguments and returns a list of all neighbours reachable from the object based on ε and the distance() function as described above. The algorithm once executed successfully, in its entirety, returns clusters that can be iterated through or visualised using the display member function, on a graph in unique sizes and colours.

- GDBSCAN

Each city in the dataset is represented as an object with the following attributes: city name, a list of mean yearly temperatures, and the overall mean, max and min annual temperatures for the city. Before clustering with GDBSCAN, every object is assigned a *cluster_id*=UNCLASSIFIED. The class here too contains two member functions. The first changes the cluster id of the object passed to it and the second is to display the clustering results in a plot as well as list out the members of each cluster. GDBSCAN takes a list of city objects of the cities to be clustered as its primary parameter. Along with the data, we pass it pointers to the *NPred* and *wCard* functions as well as a minimum cardinality value, *MinCard*. *wCard()* returns a weighted cardinality value for each object passed to it. *NPred()* is used to check if some condition, such as a distance condition, between 2 objects is satisfied or not. Both these functions can be dependent on non-spatial attributes of the objects. GDBSCAN, on completing execution, returns clustered objects that can be plotted on a graph as shown below. The getNeighbours() function returns the neighbours for the object passed to the it with respect to the conditions defined in NPred. This function is called in the Expand_Cluster method to get the neighbouring points for an object when expanding cluster.

- OPTICS

The OPTICS algorithm takes a set of city objects with attributes city name, mean yearly temperatures, overall mean, max and min yearly temperatures and an id to check if the object has been processed or is yet to be visited. At the beginning, a reachability distance of -1 is assigned to each object as well as to the id attribute to signify that the object is still not processed, as none of the objects have been visited yet. The class of city of objects contains two member functions, to calculate the distance between two objects based on their mean temperature values, and to plot & display the results obtained from the algorithm, respectively. For OPTICS, we plot both a reachability plot as well as an attribute plot to visualise the cluster ordering and member values. It is clear that a low reachability distance for a member object signifies a small distance from a cluster's core objects and hence increases the chances of the object belonging to that cluster. OPTICS too has helper functions. The first returns the ε-neighbourhood for a chosen object with respect to the distance function mentioned

above. The second checks if an object is a core object or not and in case it is, returns the core distance (i.e. the minimum distance from the object to a neighbour in its ε-neighbourhood) for it. Our implementation of the OPTICS algorithm returns two lists, one for the clusters and the second for all objects marked as noise. Each is separately iterated through, and using the display member function, we visualise the cluster ordering for the dataset along with their reachability distances to justify an object's membership to a cluster.

First, the ε-neighbourhood of each unprocessed object in the dataset is retrieved and then the object is added to an ordered list. Next, the core-distance for each object is calculated and in case the core object conditions are not satisfied, the scanning process continues to the next unvisited object in the dataset. In the case of a core object, the algorithm enlarges the potential cluster by checking for all the neighbours of the object at distance ≤ *eps*. Objects directly reachable from it are sorted to a seeds list with respect to their reachability-distance from the closest directly reachable core object. In each iteration, the ε-neighbourhood of the object with the smallest reachability-distance in the seeds list is determined. This object is then written directly to the ordered list. For core objects, future candidates for expansion are added into the seeds list also. These tasks of handling reachability distance and sorting into the seeds list is managed by the update(p, N, seeds, eps, MinPts). Objects are sorted with their reachability-distances and added to the seed list if they're not already in it. Objects previously inserted into the list are updated with their new reachability-distances if those values are smaller than their former reachability-distances.

6. RESULTS AND DISCUSSION

Each of the 3 density-based algorithms was implemented to cluster cities of the world based on their measured temperatures. In this section we describe the results obtained on clustering the city data from our dataset with more than quarter million records. Since we are clustering based on temperature, the clusters formed depend on the surface temperature values measured at these locations. Although most cities get clustered along with others, each algorithm also finds a few outliers or noisy items amongst the clusters. It is observed that the outliers consist not only of cities which experience either extremely low or extremely high temperatures, but also cities that experience

mid range temperatures. This is due to it can be seen that these cities experience temperatures that haven't been observed at the other locations in the data. Such a city is an outlier because it doesn't experience temperatures similar enough to other cities to belong in a cluster with them. The runtime complexities, along with the runtime of each neighbourhood query for each of the algorithms is given in Table 1 below.

Runtime Complexity	Single query	DBSCAN	GDBSCAN	OPTICS
Without index	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
With spatial index	$O(\log n)$	$O(n.\log n)$	$O(n.\log n)$	$O(n.\log n)$
With direct access	$O(1)$	$O(n)$	$O(n)$	$O(n)$

Table 1.

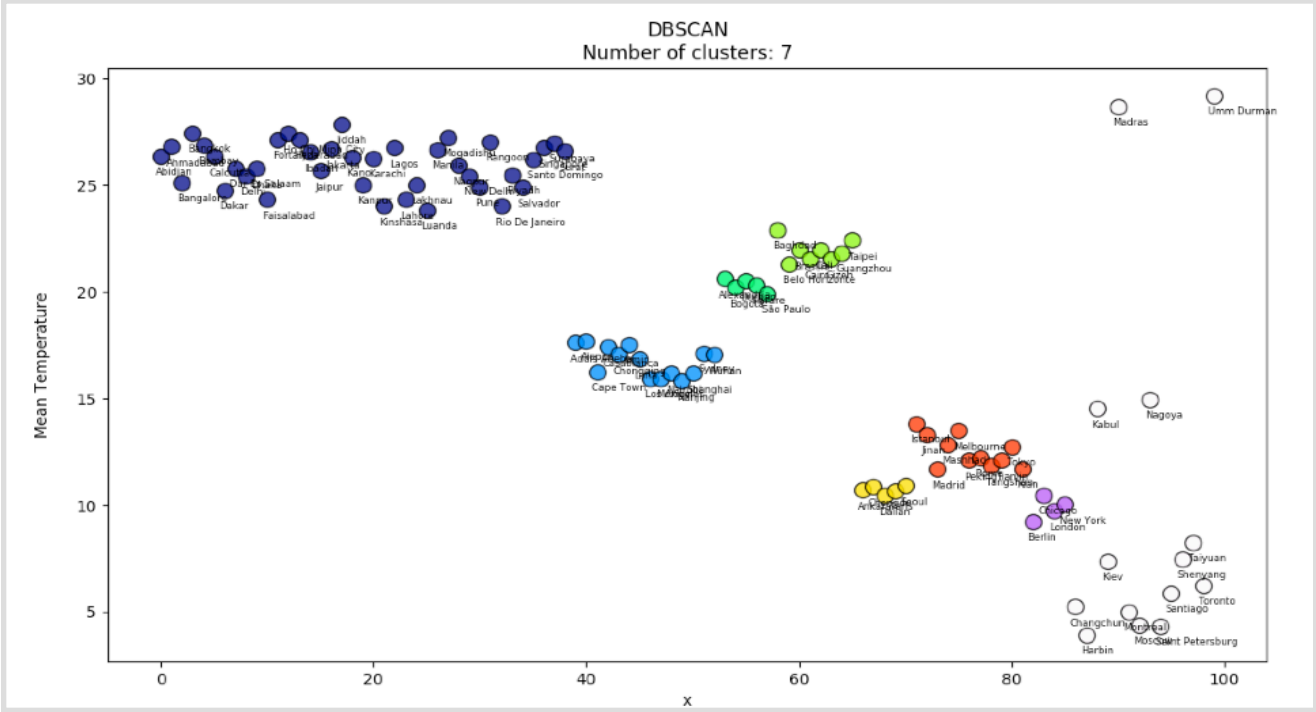

Fig 3.

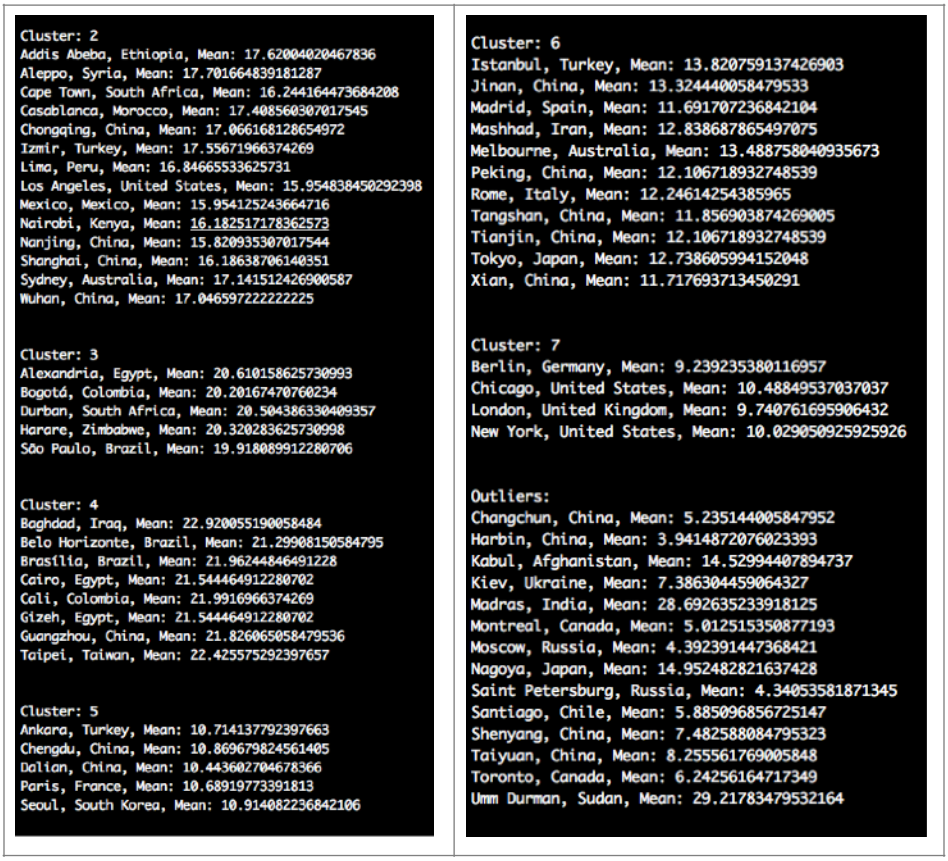

Fig 4.

The clusters formed on executing DBSCAN on our dataset are given above. GDBSCAN, being a generalisation of the DBSCAN algorithm, can also be used to implement DBSCAN by passing it the following parameters: *NPred:* distance$\leq$*Eps*, *wCard*: cardinality, *MinWeight*(*N*): $|N| \geq MinPts$. [4] For each implementation of GDBSCAN each object is visited and exactly one neighbourhood query is performed for each of them. The number of neighbourhood queries cannot be reduced since a cluster id is to be determined for each object. Thus, like DBSCAN, the overall runtime depends on the performance of the neighbourhood query. The average runtime complexity of a single neighbourhood query is O(log *n*). Using a spatial index, the runtime of GDBSCAN becomes to *O* (*n.log n*), and with direct access to the neighbourhood of each object, such as in the case when the objects are organised in a grid or when using a distance matrix, the runtime is further reduced to *O*(*n*).

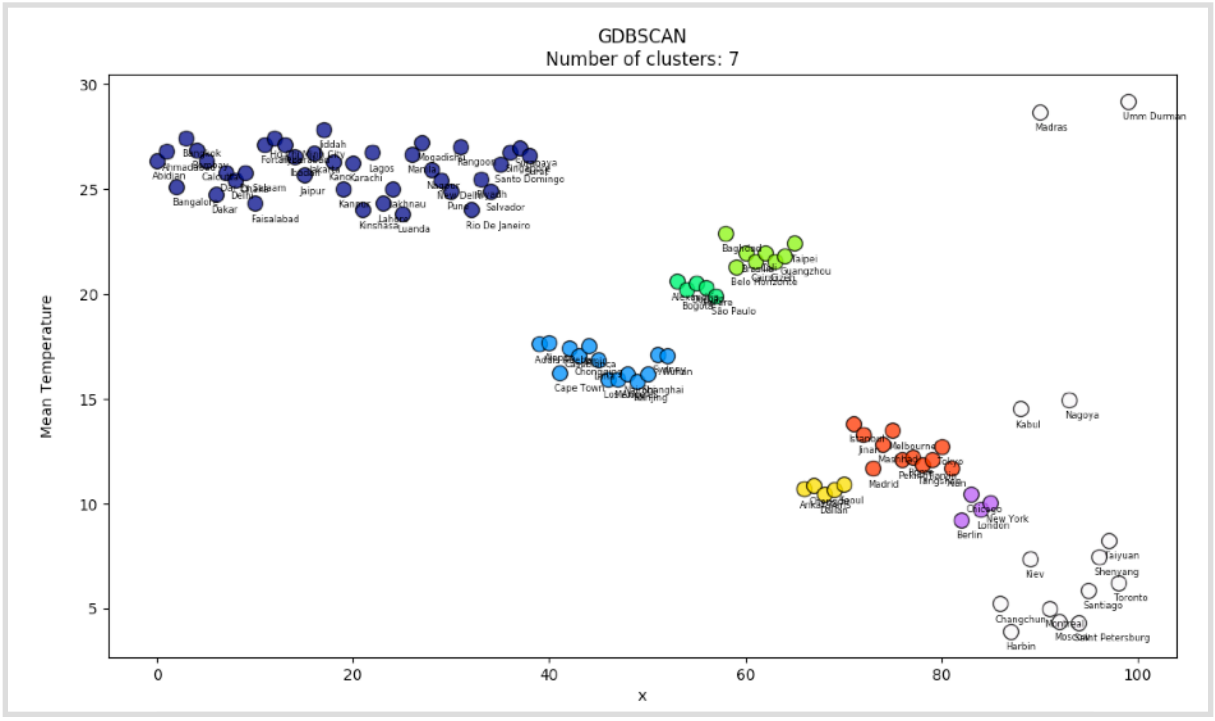

Fig. 5

```
Cluster: 2
Addis Abeba, Ethiopia, Mean: 17.62004020467836
Aleppo, Syria, Mean: 17.701664839181287
Cape Town, South Africa, Mean: 16.244164473684208
Casablanca, Morocco, Mean: 17.408560307017545
Chongqing, China, Mean: 17.066168128654972
Izmir, Turkey, Mean: 17.55671966374269
Lima, Peru, Mean: 16.84665533625731
Los Angeles, United States, Mean: 15.954838450292398
Mexico, Mexico, Mean: 15.954125243664716
Nairobi, Kenya, Mean: 16.182517178362573
Nanjing, China, Mean: 15.820935307017544
Shanghai, China, Mean: 16.18638706140351
Sydney, Australia, Mean: 17.141512426900587
Wuhan, China, Mean: 17.046597222222225


Cluster: 3
Alexandria, Egypt, Mean: 20.610158625730993
Bogotá, Colombia, Mean: 20.20167470760234
Durban, South Africa, Mean: 20.504386330409357
Harare, Zimbabwe, Mean: 20.320283625730998
São Paulo, Brazil, Mean: 19.918089912280706


Cluster: 4
Baghdad, Iraq, Mean: 22.920055190058484
Belo Horizonte, Brazil, Mean: 21.29908150584795
Brasília, Brazil, Mean: 21.96244846491228
Cairo, Egypt, Mean: 21.544464912280702
Cali, Colombia, Mean: 21.9916966374269
Gizeh, Egypt, Mean: 21.544464912280702
Guangzhou, China, Mean: 21.826065058479536
Taipei, Taiwan, Mean: 22.425575292397657


Cluster: 5
Ankara, Turkey, Mean: 10.714137792397663
Chengdu, China, Mean: 10.869679824561405
Dalian, China, Mean: 10.443602704678366
Paris, France, Mean: 10.68919773391813
Seoul, South Korea, Mean: 10.914082236842106
```

```
Cluster: 6
Istanbul, Turkey, Mean: 13.820759137426903
Jinan, China, Mean: 13.324440058479533
Madrid, Spain, Mean: 11.691707236842104
Mashhad, Iran, Mean: 12.838687865497075
Melbourne, Australia, Mean: 13.488758040935673
Peking, China, Mean: 12.106718932748539
Rome, Italy, Mean: 12.24614254385965
Tangshan, China, Mean: 11.856903874269005
Tianjin, China, Mean: 12.106718932748539
Tokyo, Japan, Mean: 12.738605994152048
Xian, China, Mean: 11.717693713450291


Cluster: 7
Berlin, Germany, Mean: 9.239235380116957
Chicago, United States, Mean: 10.48849537037037
London, United Kingdom, Mean: 9.740761695906432
New York, United States, Mean: 10.029050925925926


Outliers:

Changchun, China, Mean: 5.235144005847952
Harbin, China, Mean: 3.9414872076023393
Kabul, Afghanistan, Mean: 14.52994407894737
Kiev, Ukraine, Mean: 7.386304459064327
Madras, India, Mean: 28.692635233918125
Montreal, Canada, Mean: 5.012515350877193
Moscow, Russia, Mean: 4.392391447368421
Nagoya, Japan, Mean: 14.952482821637428
Saint Petersburg, Russia, Mean: 4.34053581871345
Santiago, Chile, Mean: 5.885096856725147
Shenyang, China, Mean: 7.482588084795323
Taiyuan, China, Mean: 8.255561769005848
Toronto, Canada, Mean: 6.24256164717349
Umm Durman, Sudan, Mean: 29.21783479532164
```

Fig 6.

In the case of OPTICS, the reachability-distance represents the least distance from an object to its predecessors. Although OPTICS doesn't explicitly define clusters, they may be determined by grouping together consecutive objects with reachability-distance less than a chosen threshold value. The reachability-distance thus represents the distance of the object from the closest cluster. A high value for distance indicates a large distance from the other objects and such an object is an outlier. The distance ε influences the number of clusters that are seen in the reachability-plot. For smaller values of ε, there will be a greater number of objects having UNDEFINED reachability-distance resulting in few or no clusters of lower density. If no sufficiently dense clusters are found, the core and reachability distances for the data objects are not defined and all objects are outliers in this case. The reachability-plot is itself insensitive to the input parameters but still contains information of each of the clusters. Precise input values are not needed because the change in the resulting reachability plot is negligible for a wide range of input values. We may start with any arbitrary object since the algorithm puts together all the objects in their resulting cluster ordering. The objects are visited in the original order until a core object is found, after which, the core object's neighbourhood is expanded by visiting each the density-connected object. The order of visiting objects thus depends on the distances between them and not on their order in the dataset. [3] The run-time of the algorithm OPTICS is nearly the same as the runtime for DBSCAN, or to be precise, almost constantly 1.6 times the runtime of DBSCAN. [6]

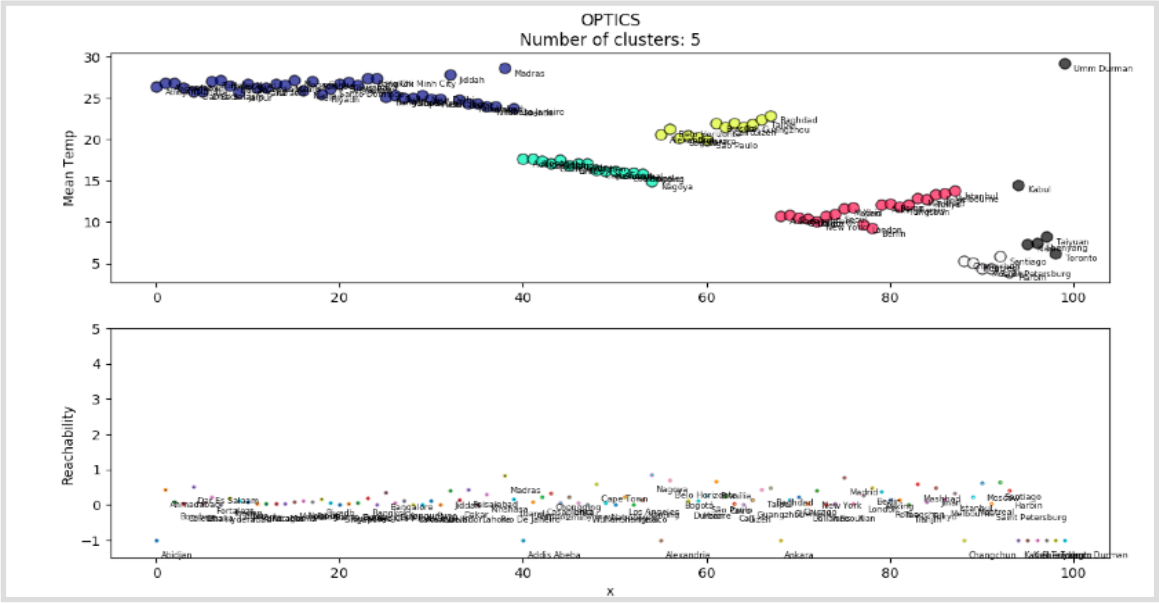

Fig. 7

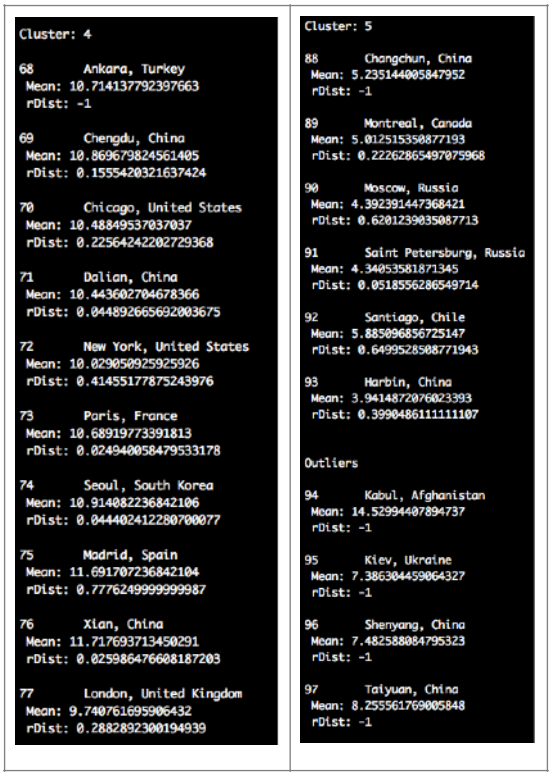

Fig 8.

This is because both their runtimes rely heavily on the runtime of the neighbourhood queries that are implemented for each object in the dataset. Without index support for the neighbourhood queries, a scan through the whole dataset must be conducted. This is similar to setting the value of ε so high that

every query returns the entire dataset and the worst case run-time complexity becomes $O(n^2)$. ε is therefore necessary in cutting off the density of clusters that are not considered relevant. and helps in speeding up the algorithm. If needed, a distance matrix may be used to avoid distance re-computations. This requires $O(n^2)$ memory compared to a non-matrix based implementation which only needs O(n) memory.

For OPTICS too, we find some of the same outliers as we when we ran DBSCAN and GDBSCAN on our data. At the same time, the clusters formed also exhibit large degrees of similarity with those formed using the previous algorithms. This is due to two main reasons. First is the type of distance function we have used in the algorithms to calculate the distance between the data objects and, by extension, find the *eps*-neighbourhood of that object. We use a euclidean measure to calculate the difference in temperatures of different cities but any other suitable distance functions may be used instead. This includes distance estimating functions that don't depend on the spatial attributes of the objects in the data as we saw in the case of GDBSCAN, where any notion of a neighbourhood for an object may be applied through the *NPred* method, which can use non-spatial attributes to define the cardinality of that neighbourhood based on the distance amongst objects. The second reason has to do with the data and the attributes on which the clustering is performed. Since we have clustered the cities based on mean yearly temperatures, the clusters are plotted accordingly, relying on the surface temperatures at these locations. In general, it is seen that cities, which experience similar temperatures, low or high, are found to be close together within the same clusters. This is the nature of clustering itself, to find similarities in data based on relevant input attributes of the data objects, and signifies that we have been successful in our implementations of the algorithms to cluster the temperature dataset from Berkeley Earth.

7. CONCLUSION

Density based methods of clustering largely depend on an effective execution of the neighbourhood query. In order to improve the performances of such algorithms it is necessary to have the availability of valid index data structure. [6] When dealing with spatio-temporal data, it is necessary to adapt the existing approaches for the spatio-temporal domain [7] or use a general distance based index. [8]

The approach of choosing a clustering method and a distance function is just a starting point for a more evolved approach to data mining. There is a need for discovering interesting time intervals in which object behaviour can be organised into meaningful clusters using all possible time intervals or time windows, evaluating the results and finding the best clustering approach. Hence, it is very crucial for the algorithm to efficiently work on different spatial and temporal granularities. In most cases, the problem of finding the best clusters converges to finding the best-input parameters for the algorithm.

8. REFERENCES

A. https://berkeleyearth.org/data/

Papers:

1. Jiawei Han, Micheline Kamber and Anthony K. H. Tung, School of computing science, Simon Fraser University, Burnaby, BC Canada V5A 1S6, Spatial Clustering Methods in Data Mining: A Survey.
2. Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei. Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M. A density-based algorithm for discovering clusters in large spatial databases with noise. (DBSCAN) - Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 1996
3. Slava Kisilevich, Florian Mansmann, Mirco Nanni, Salvatore Rinzivillo. Spatio-Temporal Clustering: A Survey, Data mining and Knowledge Discovery Handbook, Springer
4. Jörg, Sander, Martin Ester, Hans-Peter Kriegel, Xiaowei Xu. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. Data Mining and Knowledge Discovery, 1998
5. Bindiya M. Varghese, Unnikrshnan A, Poulose Jacob. Spatial Clustering Algorithms – An Overview, Asian Journal of Computer Science and Information Technology, 2013
6. Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel and Jörg Sander. OPTICS: Ordering Points To Identify the Clustering Structure, 1999
7. Frentzos E, Gratsias K, Theodoridis Y. Index-based most similar trajectory search, 2007
8. Ciaccia P, Patella M, Zezula P. M-tree: An efficient access method for similarity search in metric spaces. (1997)