Fall 2012

HIERARCHICAL CLUSTERING USING LEVEL SETS

Francesco Indaco
San Jose State University

HIERARCHICAL CLUSTERING USING LEVEL SETS

A Thesis

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Francesco Indaco

December 2012

SAN JOSÉ STATE UNIVERSITY

The Designated Thesis Committee Approves the Thesis Titled

HIERARCHICAL CLUSTERING USING LEVEL SETS

by

Francesco Indaco

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

May 2013

Dr. Teng Moh
Department of Computer Science

Date

Dr. Mark Stamp
Department of Computer Science

Date

Dr. Robert Chun
Department of Computer Science

Date

ABSTRACT

LEVEL SET CLUSTERING

Over the past several decades, clustering algorithms have earned their place as a go-to solution for database mining. This paper introduces a new concept which is used to develop a new recursive version of DBSCAN that can successfully perform hierarchical clustering, called Level- Set Clustering (LSC). A level-set is a subset of points of a data-set whose densities are greater than some threshold, ‘t’. By graphing the size of each level-set against its respective ‘t,’ indents are produced in the line graph which correspond to clusters in the data-set, as the points in a cluster have very similar densities. This new algorithm is able to produce the clustering result with the same O(n log n) time complexity as DBSCAN and OPTICS, while catching clusters the others missed.

ACKNOWLEDGEMENTS

This project would not have been possible without the efforts of the faculty and staff of the Computer Science Department of San Jose State University who provided me with the support I needed to expedite this endeavor.

TABLE OF CONTENTS

1.0 INTRODUCTION

In the Age of Information, the world faces an increasing need for even more efficient and more effective data processing methods. Because of this, clustering finds itself useful in many other fields beyond the realm of Computer Science.

As defined in the abstract, a level-set is a subset of points of a data-set whose densities are greater than some threshold. Although the notion of a level-set is not new, it has seemingly only been used as a part of theorems and has yet to be considered as a key component in actual algorithm. Graphing the size of the level-set against its respective threshold gives a glimpse into the underlying cluster tree. Steep drops appear in this downward line graph, representing a large number of points with similar densities which indicates the presence of a cluster.

Given the nature of level-sets, it is more than fitting to pair this up with DBSCAN, a density-based clustering algorithm, which performs the initial clustering on the data-set. Looking at each of the resultant clusters individually, LSC finds their sub-clusters by graphing the level-sets and clustering again at the next major drop, pruning everything less dense than it. The algorithm then continues in this recursive divide and conquer fashion, clustering each level and searching for more subclusters.

2.0 RELATED WORK

Because of the importance of being able to cope with large datasets, the academic world of Computer Science has been working hard to develop scalable algorithms to help solve this important problem.

2.1 Level-Sets and Cluster Trees

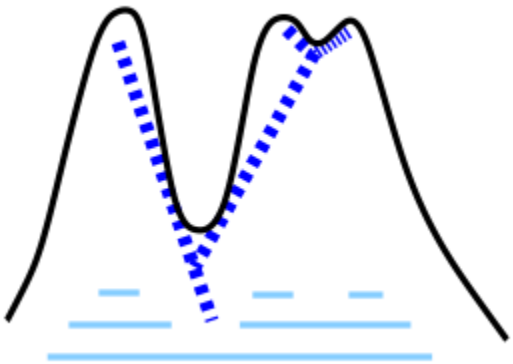


Figure 1: The density graph of a 1 dimensional data set (black line) with its cluster tree superimposed on it (dark blue dotted line)
(Kpotufe, Samory & von Luxburg, Ulrike 2011)

The original inspiration for the work done in this project was the paper entitled "Pruning Nearest Neighbor" (Kpotufe, Samory & von Luxburg, Ulrike 2011). That paper expanded upon the work by the same professor published 2 years earlier, "Optimal construction of k-nearest neighbor graphs for identifying noisy clusters" (Maier, M., Hein, M., & von Luxburg, U 2009) This preliminary paper is an investigation into the merits of clustering algorithms that use different neighborhood graphs, namely ones that use symmetric k-nearest neighbor or mutual k-nearest neighbor. The authors found the bounds on the probability for the successful identification of clusters. They apply graph

theory to the realm of clustering to help uncover cluster trees in data sets. Using the notion of k-NN graphs and t-level-sets, a data set can be effectively modeled and the optimal clustering result can be found.

The authors of the subsequent paper go further to develop a method for pruning fake branches in these cluster trees (Kpotufe, Samory & von Luxburg, Ulrike 2011). One of the challenges that the authors faced was inspecting a level-set that pruned the data in such a way, that it appeared that a larger cluster had split into two. Instead the data points that once connected these two sub-clusters happened to be just below the threshold leading to a split that shouldn’t have happened.. Their method checks a less dense level-set to see if the split is legitimate or not.They were able to carefully walk the line of pruning too aggressively and pruning too conservatively, successfully removing all spurious cluster while leaving the true cluster tree intact.

The paper entitled, "Rates of convergence for the cluster tree" outlines a statistical approach to estimating cluster trees by performing a convergence analysis on Wishart's Scheme from 1969 (Chaudhuri, K. &Dasgupta, S 2010). Earlier efforts provided weak consistency for single-linkage clustering algorithms like Wishart's Scheme, however, this is the first to properly prove it.. Like the two paper's above, this one focuses on the clusters generated from k-nearest neighbor graphs.

It was these papers that introduced me to the notion of t-level-sets.This inspired the idea to graph the sizes of these level-sets on a two dimensional line graph, with

increasing density on the horizontal axis and the level-set size at that density on the vertical axis. The line produced works its way from the upper left corner of the graph (where every single point has a density of 0 or greater) to the lower right corner of the graph (where the densest point is the only point of that density). What is interesting about this graph is that it takes sudden dips when there are a lot of points that have the same density, which is indicative of a cluster.

2.2 Density-Based Clustering

These graphs felt similar to the reachability plots from the OPTICS algorithm, introduced in the paper "OPTICS: Ordering Points To Identify the Clustering Structure". OPTICS is an interesting derivation of the clustering algorithm DBSCAN (Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander 1999). The authors change the portion of DBSCAN that gives a data point its cluster label and instead it writes out that point to a file. Each point is also given a new attribute called it's reachability. The reachability of a point is determined while appending new neighbors too the queue during a region-query (using a spatial indexing structure, such as R-Tree, to return the list of neighbors within eps from a point) (Beckmann N., Kriegel H.-P., Schneider R., Seeger B 1990). The reachability of a neighbor is defined as the distance between the neighbor and the point being queried or the distance to the k-nearest neighbor of the neighbor, whichever is larger. Neighbors are also removed from the queue, lowest reachability first. The result is the following graph.

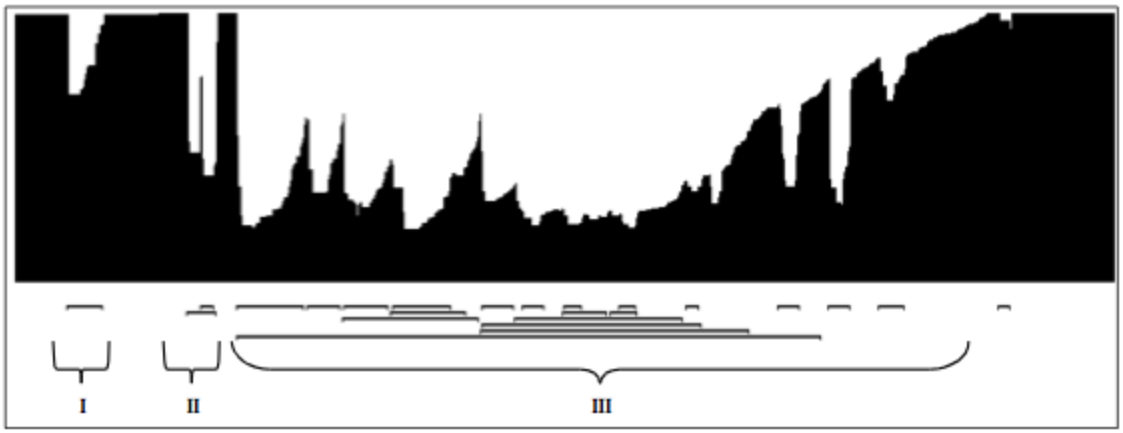


Figure 2: reachability plot produced by the OPTICS algorithm
(Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander 1999)

Valleys mark the occurrence of a cluster, and deeper / narrower valleys within shallower / wider valley indicate cluster within clusters. There is a standalone portion of the OPTICS algorithm that is able to extract these clusters from the graph. Traversing from left to right it uses heuristics to match “steep down areas” to “steep up areas,” successfully identifying clusters. OPTICS will be my measure for comparison in terms of quality of results and time complexity as it is also a hierarchical rendition of DBSCAN.

As OPTICS was based on DBSCAN, it was useful to dig deeper into the paper that introduced and outlined it, "A density-based algorithm for discovering clusters in large spatial databases with noise" (Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu 1996). DBSCAN takes the two parameters 'MinPts' and 'Eps' which refer to how many neighbors a point needs in order to be considered a "core point" and how close a pair of points need to be to each other in order to be considered neighbors, respectively. The algorithm then leap-frogs across core points that are neighbors of each others to identify clusters. The problem that this algorithm faces is picking

appropriate input values for its parameters which can cause it to miss clusters that are small or to identify spurious clusters. Furthermore it is a partitional algorithm, incapable of recognizing sub-clusters.

2.3 Conclusion

Given the amount of work being done in the field of clustering, cluster trees, and machine learning as a whole, finding the solutions to these problems is proving quite challenging and critical. The industry is always looking for more robust and faster ways to handle its large amounts of data.

3.0 Graphing Level- Sets

As mentioned earlier, research into statistical graph theory papers introduced the notion of a level-set. While the simple definition of this special subset has seemingly never been a focal point in a clustering algorithm, the proper application of the concept will prove to be powerful.

3.1 Analyzing the graph

To visualize the capabilities of the information contained in the level-set sizes, we begin by graphing the size of level-set on the vertical axis against its corresponding threshold on the horizontal axis. As explained above, the level-set graph experiences steep dips when there are a lot of points of the same, or at least similar, densities in the data set. These dips are separated by flatter stretches that continue to the right until the next collection of even denser points cause another dip. All the noise in the graph causes a smaller series of decreases in the graph on the left side (less dense side). Given the high variance in possible density values for noise points, it does not amass as a significant dip helping the experimenter ignore them easily.

However, what has show to be problematic is a large variance in the densities for clusters themselves. A dip still exists, but it will not be nearly as sharp, and becomes harder to detect.

3.2 Kernel Density Estimators

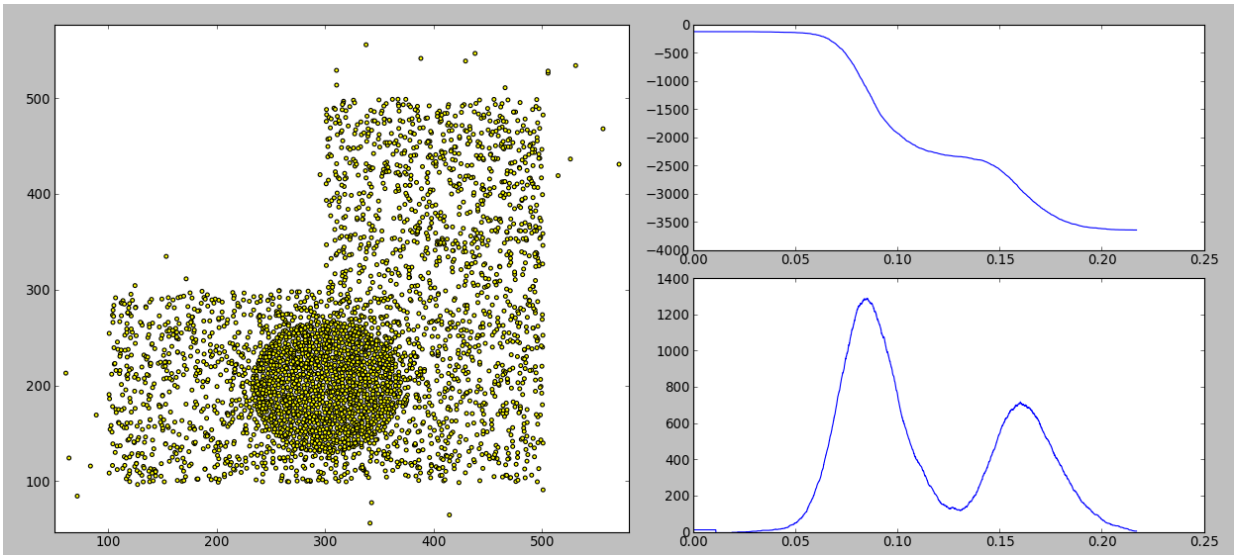


Figure 3: the level-set graph (upper right) and it's kernel-density estimated derivative (lower right) for the subset (left)

It is better to represent the data as a 'derivative' of the current graph. Kernel density estimators from statistics are designed exactly for this purpose. First all the points from the data set are projected onto the horizontal (density) axis of the level set line graph. This means that they are ordered in increasing density, each point sitting on the x-value that corresponds to its density which is also the the level-set threshold that would cause it to be pruned. If this were to be visualized, the axis would have bunches of points directly underneath the dips. Second, a kernel density estimator is run horizontally across the data. This produces peaks where the dips occurred with wide valleys in between them.

3.3 Identifying Peaks

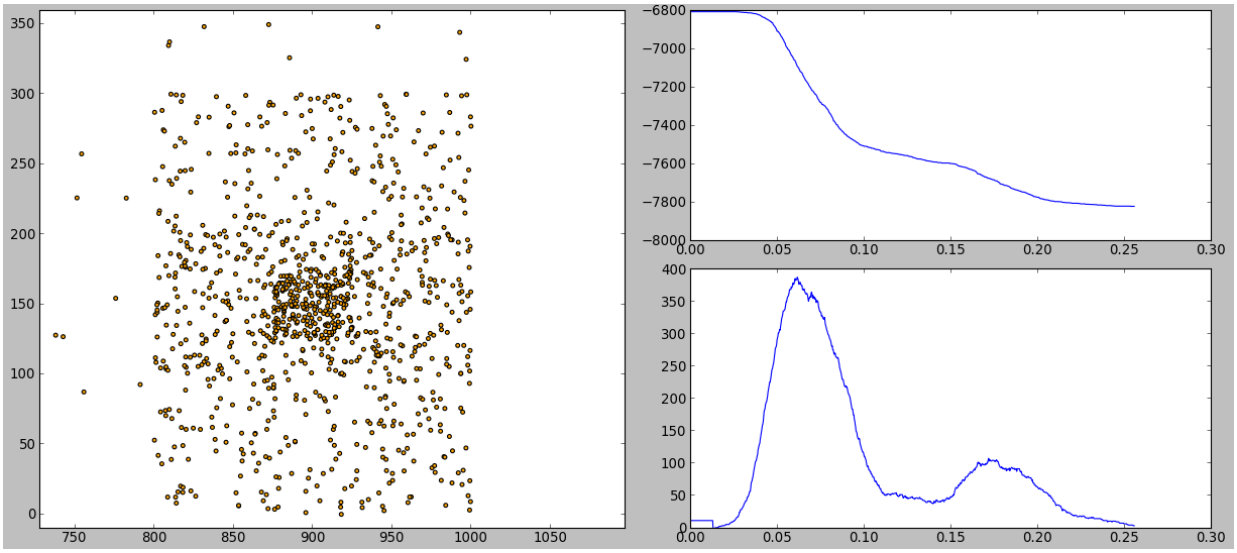


Figure 4: Here the cluster produces a more jagged level-set graph

Visually, the peaks are seemingly easier to work with. The varying degree of similar densities affect the steepness of the dips more than the height of the peaks. The effect on the peak is a larger taper at its base, as well as jaggedness in the line. The valleys on both sides of the peak would need to be identified in order to successfully determine the range of densities for the cluster. The peak itself marks the most common density held by the points within the cluster, but there is variance in the densities of the points of the cluster. The points towards the edges of the cluster as well as some of the points within the clusters will not be as dense. These points will be missed if too dense of a point is chosen as the start of the peak. Choosing a point that is not dense enough will cause too many of the surrounding points to be chosen as well.

Programmatically, this jaggedness proved problematic in choosing valleys. Without more expensive methods of developing less jagged peaks, like the Gaussian kernel density estimator, it is impossible for the level-set graph to work. Instead, we turn to the projected points from section 3.3, lying on the density axis bunched up underneath the peaks which superimpose on the dips. The valleys that need to be identified are essentially the edges of a one dimensional cluster. Even when successfully identified, this method would not be able to handle cases such as the multiple clusters of the same density, as they would all be represented as the same peak. There needs to be some level of integration with existing clustering methods in order to handle such scenarios.

The algorithm outlined on the next page shows a rendition of the graph generation that can be used as pruning technique for a recursive partitional clustering framework which is outlined in section 5.0.

4.0 Level-Set Algorithm

```
Prune_base_cluster( data )
1    projection = project_densities( data )
2    clusters = DBSCAN( projection )
3    cluster = Find_Least_Dense_Cluster( cluster )
4    valley = Find_Densest_Point( cluster )
5    return Prune_Everything_Less_Dense_Than( data, valley )
```

The pseudo code above shows the general steps taken by the level-set algorithm used by the general clustering algorithm. It receives a set of data for which it will remove the least dense cluster of points that appear in the density graph.

The algorithm begins by representing the data as having 1 dimension, that being each point's density. It is assumes that the densities for each point have already been calculated beforehand, which will be discussed in section 5. Next the algorithm runs the one dimensional projection through DBSCAN to find the peaks described in section 3. This algorithm is only interested in the least dense cluster, as it is removing it. In order to do this, it needs the densest point in the least dense cluster which is essentially the valley to the right of the left-most peak.

Once found, all the points that are less dense than this valley are removed. This subset is then returned so that it can then be clustered to find the denser sub clusters.

```
Prune_base_cluster( data )
1    qsort( data )
2    left_i = 0, right_i = 0, num_in_range = 0, is_first = true
3    first_core_point = -1, last_core_point = -1, valley = -1
4    w = (data[|data|-1]->density - data[0]->density) * MINPTS *
          2 / ( |data| * sqrt(pi) )
5    for i = 0, |data| -1 do
6         if data[i]->density == 0 then continue
7         while right_i < |data| && data[right_i]->density <
                    (data[i]->density + w) do
8              num_in_range++
9              right_i++
10        while left_i < |data| && data[left_i]->density <
                    (data[i]->density - w) do
11             num_in_range—
12             left_i++
13        if num_in_range > MINPTS then
14             last_core_point = i
15             If is_first then
16                  is_first = false
17                  first_core_point = i
18        elseif last_core_point != -1 && (data[i]->density –
19                  data[last_core_point]->density) > w then
```

```
20              if last_core_point-first_core_point > MINPTS then
21                   return prune( data, last_core_point )
22              last_core_point = -1
23              is_first = true
24    return {}
```

The pseudo code above shows the true steps taken by the level-set algorithm. It sorts the data-set in order of ascending density. Next, lines 5-12 perform a uniform kernel density estimator to determine if each point is a DBSCAN core-point. Line 4 is a formula for *epsilon* that is explained in further detail in section 5. The *epsilon* generated is used a kernel for the estimator, which means that all other points that fall into range are the neighbors. Lines 13-14 mark down if the current point is a core point. If the current point is not a core point and it is farther than *epsilon* from the last core point, then the last core point marks the end of a cluster, as long as there were *minpts* points in that cluster (Lines 18-21). If there were not *minpts* points, then the variables are reset and the estimator continues moving (Lines 22-23).

This approaches stops as soon as the first cluster is discovered and prunes on the last core point that was seen. Since the estimator is moving in ascending density, this will be the densest point of that least dense cluster. In terms of time complexity, it is limited by qsort which is O(n log n). In the simplified version on the previous page, the supposed call to DBSCAN would have also limited the complexity to O(n log n).

5.0 CLUSTERING ALGORITHM

```
Main(data)
1    findDensities(data)
2    clusters = DBSCAN( data )
3    for each c in clusters do
4         Recursive_Cluster( c )
5    return clusters
```

```
Recursive_Cluster( parent, eps )
1    pruned_points = Prune_base_cluster( parent )
2    if  |pruned_points| > MINPTS  then
3         parent->children = DBSCAN( pruned_points)
4         for each c in parent->children do
5              Recursive_Cluster( c )
```

```
calculate_epsilon(data)
1    volume = 1;     dim = |data->dimensions|;
2    for each d in data ->dimensions do
3         volume *= (data[d]->max - data[d]->min)
4    return ((volume * MINPTS * 2 * Γ(dim/2 - 1))/
          ( |data| * sqrt(PI^ dim))^(1/ dim)
```

```
DBSCAN(data)
1    eps = calculate_epsilon(data)
2    for each point in data do  point->processed = false
3    clusters = {}
4    for each point in data do
5         if point->processed then     continue
6         point->processed = true
7         queue = findNeighbors(data,point,eps)
8         if |queue| < MINPTS then     continue
9         current_cluster = {}
10        clusters.add(current_cluster)
11        current_cluster.add(point)
12        while |queue| > 0 do
13             p = queue.pop()
14             if p->processed then     continue
15             p>processed = true
16             new_neighbors = findNeighbors(data,p,eps)
17             if |new_neighbors| >= MINPTS then
18                  queue = queue + new_neighbors
19                  current_cluster.add(p)
20   return clusters
```

(Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu 1996)

5.1 Integrating DBSCAN

The above pseudo-code illustrates the full clustering algorithm that makes use of the level-set graphing code. The core clustering technique used is DBSCAN, which is the last function shown. LSC calls DBSCAN to perform an initial clustering and then recursively calls DBSCAN on the individual clusters produced. Each iteration, the level-set code is used to prune the base / parent cluster out of the subset to make it easier to find the denser children clusters.

$$\varepsilon = \sqrt[d]{\frac{Vol \times 2 \times minpts \times \Gamma(\frac{\mathrm{d}}{2} + 1)}{n \times \sqrt{\pi^d}}}$$

Figure 5: The formula for calculating epsilon, from the OPTICS paper. (Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander 1999)

In addition to pruning every iteration, the epsilon parameter is modified each time as well. Traditionally DBSCAN requires the user to enter epsilon as one of its input parameters. LSC makes use of the formula suggested in the OPTICS paper (Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander 1999), to better tailor DBSCAN to the next iteration. In the figure above: Vol is the volume occupied by the data set, d is the number of dimensions, n is the number of points, minpts is the input parameter to the algorithms, and Γ is the gamma function. In the pseudo code, this formula manifests itself as the function 'calculate_epsilon().' It requires knowing the bounds of the data set, which requires finding the maximum and minimum attribute value for each dimension, in order to produce the volume of the sample space. This linear operation (shown in simplified on line 2-3) entails making a full pass through the data-set, or subset, that is about to be clustered. However, the ability to chose a proper

epsilon has proven itself a necessary expense, as will be further explained in the evaluation.

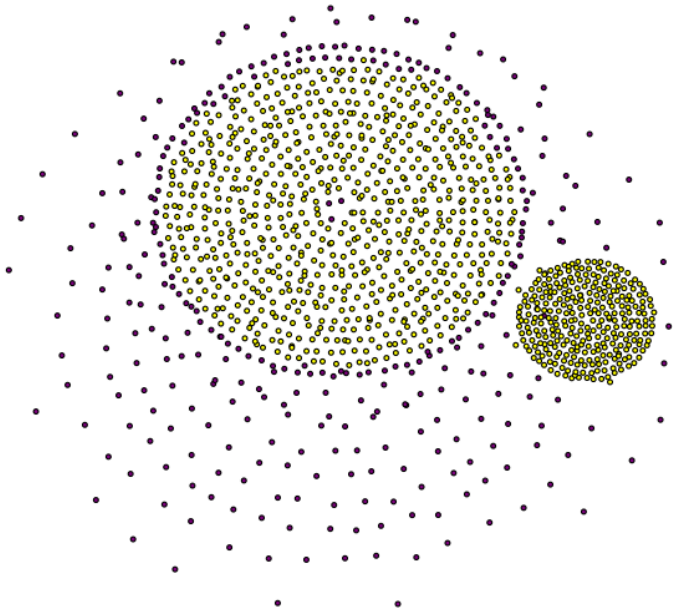


Figure 6: an example of DBSCAN marking 2 separate subclusters as a single sub-cluster, due to their proximity.

Furthermore, this was found to be especially effective as the epsilon that worked at the base iteration was far too large at denser levels. Before adding this step, the subsequent iterations of DBSCAN had the problem of marking all of the smaller higher density clusters as a single cluster as the gap between these smaller clusters was not large enough, as is illustrated above.

```
findDensity(point,data)
1    neighbors = findNeighbors(data,p,eps)
2    if |neighbors| < MINPTS then
3         return 0
4    avg_dist = 0
5    for each neighbor in neighbors
6         avg_dist += dist(neighbor,p)
7    return avg_dist / MINPTS
```

In order for the level-set graph to be made, the density of each point must be known. Traditionally calculating can be an expensive procedure, in the realm of O(n^2). Since the data was already in a spatial indexing structure, the density values used here are generated by performing the same region queries and then calculating the density amongst the neighbors returned. Just like for DBSCAN, each region query is O(log(n)), making the whole operation O(n*log(n)).

Densities are not recalculated every iteration as that would require removing all the points that have been pruned from the spatial indexing structure as the recursion gets deeper and then re-inserting them when it comes time to move onto another branch. Although it may be interesting to explore what effect this would have on each recreation of the level-set graph, it was assumed to be unnecessary given how effective the algorithm proved itself to be without performing these extra steps.

This findDensity() function actually occurs as part of the findNeighbor() during the first clustering of the data (line 3 of main). Performing this separately as indicated in the pseudo code, would lead to an additional ‘n’ calls to findNeighbor(). The result of findDensity() is only used by findLevelSets() which is only called during the recursion.

5.3 Higher Dimensionality

LSC as it stands is also capable of clustering high-dimensional data, given the proper distance functions and spatial indexing structure. Subspace clustering techniques were explored as an alternative to this method to see if better results could be achieved. The subspace method investigated was the SUBCLU algorithm, which was also developed by the OPTICS authors.

SUBCLU clusters high dimensional data by clustering on the individual dimensions and then combining the dimensions in an apriori-style fashion and clustering within those subspaces. Pairs of dimensions are combined if they only differ in one attribute and they are clustered using a cluster found in the previous iteration as the data-set to be traversed to pruned the number of points to be considered in this iteration.

Originally SUBCLU calls DBSCAN every iteration. Here, it would have called LSC every iteration instead. The only issue that arises is the combination of subspace clustering and non-partitional clustering. A hierarchical algorithm like the one outlined by

this paper finds a cluster tree with in the subspace defined by the attributes from the data. A blind incorporation of a hierarchical algorithm into a subspace clustering would not only cause additional runs due to more clusters per iteration being found (clusters within clusters), but it may lead to entire branches that might not be meaningful. Any such branches that would be meaningful, would more than likely be rediscovered in another branch of the tree. Because of this, redundancy checks would be required to see if any of the clusters produced from an iteration already exist from iterations of the same subspace.

Because of the above, a more intuitive combination of the two algorithms would be required to not lose the hierarchical information gained from a single iteration. Such information would essentially be ignored. The authors of OPTICS and SUBCLU went on to develop two other algorithms, HiSC and DiSH (an improvement upon HiSC that has post-processing capable of support more complex hierarchies). Both use derivations of OPTICS algorithm, and neither use the apriori-style combinations of subspaces from SUBCLU. They instead weight the subspace occupied by each point based upon the neighbors they have in each respective dimension. Such an endeavor is beyond the scope of this project.

5.4 Analysis

This algorithm definitely favors data set that have a shallow underlying cluster tree, which would help split up the data for all of the subsequent runs, each recursive iteration running through smaller amounts of data. Given this fact it could be argued that

the worst case scenario for this algorithm would be one that results in the level-set pruning to prune only *minpts* points each iteration causing DBSCAN to potentially be called n/*minpts* times.

In order for the above to happen, the data-set would have to have a series of sub-clusters of min pts whose differences in densities are large enough for the formula-derived epsilon. A data set that could produce such a phenonmena is a series of rings, with enough spacing between each inner and outer ring for epsilon to mark them separately. Such a specific edge case like this seems extremely unlikely.

A more common edge case is the scenario where a cluster has a sub-cluster whose density is similar enough that its peak overlaps with the peak of the base cluster. This will cause the clustering of the level-set graph to identify the two peaks as one.

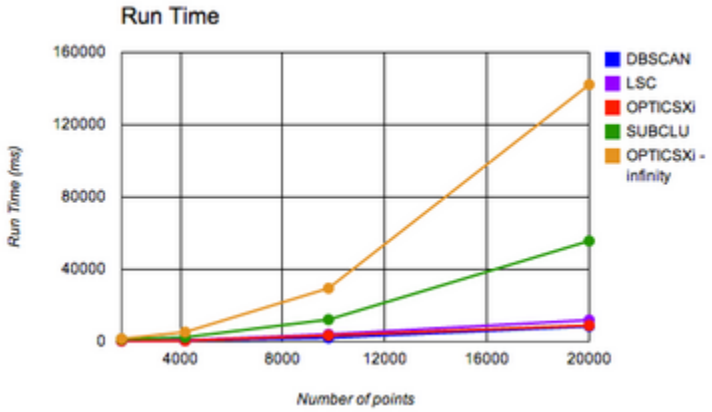
6.0 EVALUATION

In order to properly compare the run-time and clustering result of LSC, it has been integrated into the ELKI framework. ELKI (Environment for Developing KDD-Applications Supported by Index-Structures) is an open-source software project produced by the German University, Ludwig-Maximillians-Universitat Munche. This framework contains implementations of many different clustering algorithms as well as the accelerated spatial indexing structures used by algorithms such as DBSCAN. LSC was re-coded and added into the framework to provide as fair of a comparison as possible, by using the same databases, indexing structures, programming language, and abstraction overhead. For the multiple recursive iterations of DBSCAN, the actual DBSCAN code in the framework is called.

6.1 Other Algorithms

LSC will be compared against DBSCAN, OPTICS, and SUBCLU. As DBSCAN is not hierarchical, comparing against this algorithm is merely to show how much the run-time is increased by the additional iterations. DBSCAN should be just as fast as the base iteration. OPTICS is also DBSCAN based. Unlike DBSCAN, it produces a hierarchical clustering by causing a second pass through the entire data-set. LSC causes additional passes on subsets of the data, so the comparison against OPTICS will be for run-time as well as quality. Third, is SUBCLU which also calls DBSCAN multiple times for various subspaces.

6.2 Comparing Run-Time:



Size	DBSCAN	LSC	OPTICSXi	SUBCLU	OPTICSXi - infinity
1700	471	674	456	1330	1565
4188	656	738	518	2440	5305
9800	2122	4076	3434	12217	29540
20000	8504	11930	8985	55709	142207

This graph shows run-times for the algorithms on synthetic 2D data sets. The smallest is approximately 2,000 and the largest is 20,000. It illustrates that LSC scales at about the same rate as DBSCAN, with minimal overhead.

During the original evaluation, DBSCAN performed slightly worse than LSC. After investigation, it became apparent that the reason was the chosen epsilon value. Epsilon was chosen to be 50 (where the range of values in both dimensions was 1000) as it appeared to successfully identify all the clusters. The formula for epsilon used by LSC often chose much lower values for epsilon which would result in the regionQuery() to return a smaller neighborhood of points. For the sake of fair comparison, DBSCAN run again with these values from the formula. On a similar note, SUBLCLU time complexity was almost quadratic before it began using this modified DBSCAN. Before it even

begins its apriori style combining of subspaces, SUBCLU must first perform a full unpruned run of DBSCAN for each dimension (which is 2 in this case). This drives up its run-time as is shown in the graph.

OPTICSXi originally scaled worse than SUBCLU (denoted in the chart as OPTICSXi-infinity) until it too was fed the epsilon value from the LSC epsilon calculator. If no epsilon is specified the default behavior is to use an epsilon of infinity which causes each region query to return every other point in the data as a neighbor. OPTICS does this because it only looks at the closest ‘minpts’ neighbors, thus leaving epsilon as infinity guarantees that no point will be accidentally pruned. This essentially causes its time complexity to be O(n^2) instead of O(n log n) which defeats the purpose of using an accelerated indexing structure. In the paper, the authors stated that the run-time was approximately 1.6 times slower than DBSCAN (Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander 1999).

6.3 Cluster Quality

LSC was able to discover the hierarchies in the datasets with relative ease. It occasionally incurred an additional iteration when it shouldn't have if there was enough variance in the density to cause minor valley on the right slope of a peak. This causes a handful of small spurious sub clusters to be wrongfully identified. OPTICSXi produced similar such artifacts, dues to minor valleys that satisfied their thresholds.

In addition, OPTICSXi seemed to miss entire base clusters marking them as noise, regardless of whether or not the algorithm received a value for epsilon. However, the reachability plot does have apparent valleys that did not receive a cluster. This leads this author to believe that this rendition of OPTICS may still be a work in progress, as the ELKI framework is currently only version 0.5. Moreover, all the valleys that didn't get marked as clusters did have a shallow right edge which may have failed to be caught by the threshold used by its "Steep Down Area" heuristic. Lowering the threshold from the recommended 9% to 1% did lead to the other clusters to be found, but also caused an extremely large number of intermediary sub-clusters to be found.

Interestingly enough, when using a real dataset, as opposed to the 2 dimensional synthetic datasets that were generated for the run-time comparison, OPTICS performed much better since the valleys in the reachability plot that it generated were much steeper. It is not clear what is special about the synthetic data that leads it to generate unfavorable reachability. Either way, this marks another weakness for OPTICS as well as any other algorithms that make use of the reachability plot.

When it comes to higher dimensionality, LSC is only as good as its definition of density, distance, and the underlying algorithm that is recursively called. Since LSC here uses DBSCAN for clustering, it's ability to cope with the curse of dimensionality is entirely dependent on whether or not the attributes of the given data-set can be effectively compared with the Euclidean distance metric.

6.4 Memory

Given the fact that a framework with so many other clustering techniques was used as the test bench, it was anticipated to compare against more algorithms then what is presented here. Unfortunately, it appears that the framework (in its current alpha state) is not ready to handle data-sets that have thousands of points as a number of the other algorithms run out of heap space. Regardless, the fact that algorithms like DBSCAN and OPTICS were able to run and the others weren't, indicates their higher space complexity. Included in these other algorithms are the subspace successors to OPTICS, HiSC and DiSH.

The space requirements of LSC are trumped by the space requirements of DBSCAN, which is quadratic thanks to the distance matrix requiring O(($n^2 - n$) / 2) entries. LSC adds the need to store the density value for each point as well as an array that is the size of the level-set in question for the kernel density estimator graph. OPTICS and SUBCLU only add a handful of entities of size, also being held back from being more space-efficient by this distance matrix.

7.0 CONCLUSIONS & FUTURE WORK

Level Set Clustering proved capable not only of successfully performing a hierarchical clustering at speeds comparable to existing methods, but was able to find clusters that those existing methods could not.

There is a lot of opportunity for future work from the algorithm developed in this paper. The level-set information and graphing technique is versatile and modular enough to be used with and within other clustering algorithms. LSC is not specific to DBSCAN and any partitional clustering algorithm can be used in its place. DBSCAN was chosen because its biggest issue is clusters of varying densities, but this algorithm removed that problem from the picture. The framework itself could be completely reversed, performing the level-set generation first and then perform partitional clustering at each of major peaks. The only additional challenge going that direction is that there is no immediate mapping from parent clusters to children clusters. Some form of post-processing would be required to stitch them all together, and it would technically be possible to receive clusters that overlap.

Different heuristics of peak identification and different methods of peak generation can be easily swapped in. The current methods were chosen for speed, as they are called every iteration, but other possibilities can be explored. It may also be possible to to gather sufficient information from the original level-set graph, removing the need for peak generation.

8.0 REFERENCES:

1. Achtert, E.; Böhm, C.; Kriegel, H. P.; Kröger, P.; Müller-Gorman, I.; Zimek, A. (2006). "Finding Hierarchies of Subspace Clusters". *LNCS: Knowledge Discovery in Databases: PKDD 2006*. Lecture Notes in Computer Science 4213: 446–453.doi:10.1007/11871637_42. ISBN 978-3-540-45374-1

2. Achtert, E.; Böhm, C.; Kriegel, H. P.; Kröger, P.; Müller-Gorman, I.; Zimek, A. (2007). "Detection and Visualization of Subspace Cluster Hierarchies". *LNCS: Advances in Databases: Concepts, Systems and Applications*. Lecture Notes in Computer Science 4443: 152–163. doi:10.1007/978-3-540-71703-4_15. ISBN 978-3-540-71702-7

3. Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: “The R*-tree: An Efficient and Robust Access Method for Points and Rectangles”, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, ACM Press, New York, 1990, pp. 322-331.

4. Chaudhuri, K. & Dasgupta, S. Rates of convergence for the cluster tree. Neural Information Processing Systems, 2010.

5. *Environment for Developing KDD-Applications Supported by Index-Structures*. 30 June. 2012. Ludwig-Maximillians-Universitat Munche. 20 Sept. 2012 <http://elki.dbs.ifi.lmu.de/>

6. Karin Kailing, Hans-Peter Kriegel & Peer Kröger. *Density-Connected Subspace Clustering for High-Dimensional Data*. In: *Proc. SIAM Int. Conf. on Data Mining (SDM'04)*, pp. 246-257, 2004

7. Kpotufe, Samory & von Luxburg, Ulrike (2011). Pruning Nearest Neighbor Cluster Trees. Proceedings of the 28th International Conference on Machine Learning. ArXiv:1105.0540

8. Maier, M., Hein, M., & von Luxburg, U (2009). Optimal construction of k-nearest neighbor graphs for identifying noisy clusters. Theoretical Computer Science, 410:1749–1764, 2009

9. Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise". Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231.

10. Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander (1999). "OPTICS: Ordering Points To Identify the Clustering Structure". ACM SIGMOD international conference on Management of data. ACM Press. pp. 49–60.