ASSIGNMENTS
Density-Connected Subspace Clustering for High-Dimensional Data

Aditya Rajesh Patil
dataHacks
aditya18@iitg.ac.in
180101004

Bhasker Goel
dataHacks
bhasker18@iitg.ac.in
180101015

Kartikeya Saxena
dataHacks
kartikey18a@iitg.ac.in
180101034

V Anirudh
dataHacks
anirudh18@iitg.ac.in
180101084

1 REVIEW OF THE ALGORITHM

1.1 What bottleneck this clustering algorithm attacks?

The algorithm SUBCLU attacks the following bottlenecks -

(1) Most real life datasets are characterized by high-dimensional and sparse dataspaces which in turn make it very difficult to find clusters in the original data space. (The Curse Of Dimensionality)
The Algorithm attacks this by finding interesting clusters in subspaces which would have been hidden/impossible to detect in the original data space.

(2) Earlier clustering algorithms were primarily based on Grid-based approaches (E.g CLIQUE) which made it difficult for them to detect clusters that were shaped or positioned arbitrarily on the grid.
By not relying on any Grid-based approach and using DBSCAN, SUBCLU is perfectly capable of detecting arbitrary shaped and positioned clusters in the subspace.

(3) Subspace clustering algorithms often check for clusters in all Subspaces (the number of which is exponential). SUBCLU however,
i) Using Monotonicity of Density-Connectivity and
ii) Pruning irrelevant subspaces avoids doing this.

(4) Algorithms (E.g - Dimensionality Reduction or Projected Clustering) more or less, find one final clustering of the Data. But it is perfectly possible that different clustering is possible based on different attributes of data. SUBCLU allows objects to be clustered varyingly in different subspaces.

(5) Dimensionality Reduction Clustering Algorithms create new features from existing features to force-eradicate the Curse of Dimensionality. But in doing this, the features lose their intuitive meaning and the clustering becomes very hard to interpret. SUBCLU, even the considering subspaces, keeps the attributes the same as the original dataset.

1.2 What is the overall idea of the algorithm? Do include a flowchart or a figure that explain the whole algorithm.

The algorithm is designed to detect the density- connected sets in all subspaces of high dimensional data in a bottom up, greedy fashion. First, the clusters present in 1-dimension are identified applying DBSCAN as a subroutine to each 1-dimensional subspace. Then k-dimensional clusters are used to generate k+1-dimensional clusters by using DBSCAN recursively and the parameters for DBSCAN are predetermined. The algorithm is built on two properties of clusters.

(1) If two points are not in a cluster in a particular dimension, then they will not be in the same cluster in any higher dimension (Monotonicity of Density-Connected Sets).
(2) A cluster in a lower dimension might not be a cluster in higher dimension. It might lose a few points from the cluster of lower dimension, but never gain other points.

From the above two properties of clusters the following idea can be concluded:
Pruning Irrelevant Subspaces: Given two subspaces S and T, such that T is a subset of S. If T has no cluster present in it, then S will also have no clusters and computation on S can be ignored.
A naive implementation of SUBCLU would result in running DBSCAN 2^d times, that is in all possible subspaces of a d dimension space. But using Pruning of Irrelevant Subspaces reduces the computational overhead of the algorithm. This leads to an efficient implementation of SUBCLU.
SUBCLU uses inverted files data structure to process range queries encountered in the algorithm.
Flowcharts explaining the whole algorithm is shown in Figure 1 and Figure 2.

1.3 How is the evaluation of the algorithm carried out?

The performance evaluation of the algorithm was carried out with (minPts = 8, epsilon = 2 as DBSCAN parameters) using synthetic data sets and real world Gene Expression Data of CDC15 mutant [SSZ+98]. Synthetic data sets of varying size and structure were

Conference'17, July 2017, Washington, DC, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

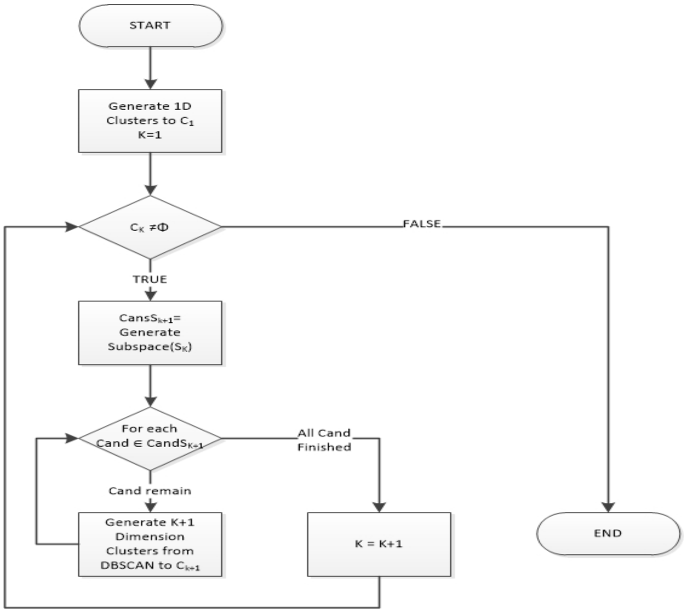

START
Generate 1D Clusters to C_1 K=1
$C_K \neq \Phi$
FALSE
TRUE
$CansS_{k+1}$= Generate Subspace(S_K)
For each Cand ∈ $CandS_{K+1}$
All Cand Finished
Cand remain
Generate K+1 Dimension Clusters from DBSCAN to C_{k+1}
K = K+1
END

Figure 1: SUBCLU Algorithm

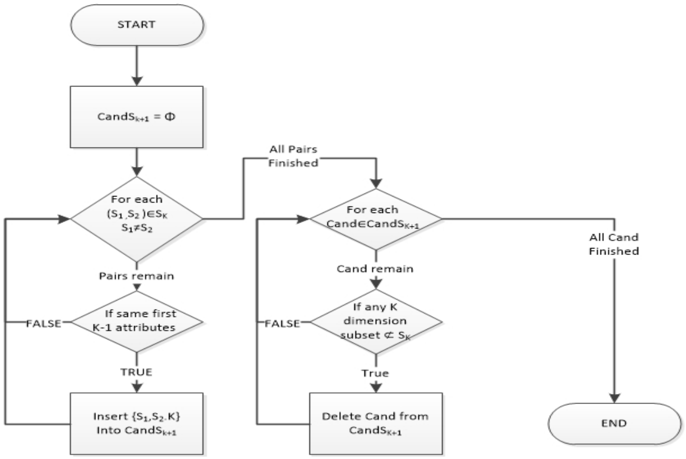

START
$CandS_{k+1} = \Phi$
For each $(S_1,S_2) \in S_K$ $S_1 \neq S_2$
All Pairs Finished
Pairs remain
If same first K-1 attributes
FALSE
TRUE
Insert $\{S_1,S_2.K\}$ Into $CandS_{k+1}$
For each Cand∈$CandS_{K+1}$
All Cand Finished
Cand remain
If any K dimension subset ⊄ S_K
FALSE
True
Delete Cand from $CandS_{K+1}$
END

Figure 2: GenerateSubspace Subroutine

created with varying dimensionality of subspace clusters and feature space. It was found that the runtime SUBCLU grows with at least a quadratic factor against data set size, dimensionality of the data set and dimensionality of subspace clusters.

For Accuracy evaluation, the results on the synthetic datasets were compared to an efficient implementation of CLIQUE (CLustering In QUEst) Algorithm and it was shown that SUBCLU outperforms the latter.

1.4 Are able to find out the code and datasets for the experiments in the paper? If not, have you contacted the authors?

The Paper provides the Pseudocode for what the authors did, but not the actual code. One of the Datasets used was real world Gene Expression Data [SSZ+98][1] which we were able to find.

The others they synthetically generated to match the application at hand. Although they provided a vague description of how they generated the data, we decided to contact the authors for the Original datasets they generated and used. We have also requested them for the code they prepared.

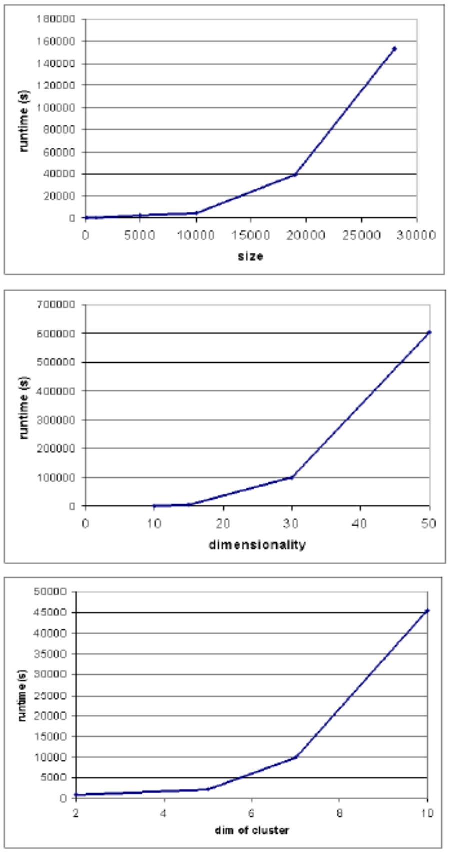
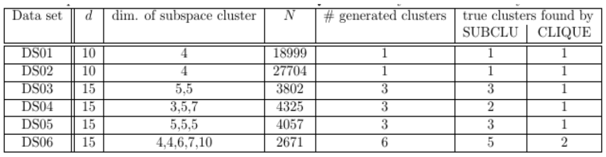


Figure 3: Performance Evaluation



Data set	d	dim. of subspace cluster	N	# generated clusters	true clusters found by	
					SUBCLU	CLIQUE
DS01	10	4	18999	1	1	1
DS02	10	4	27704	1	1	1
DS03	15	5,5	3802	3	3	1
DS04	15	3,5,7	4325	3	2	1
DS05	15	5,5,5	4057	3	3	1
DS06	15	4,4,6,7,10	2671	6	5	2

Figure 4: Accuracy Evaluation

1.5 Could you find any real-world applications of this algorithm? Does any ML/Data analysis package include this algorithm?

Subspace Clustering could potentially be used in the following real-world scenarios:

(1) **Information Integration Systems:** These systems are motivated by the fact that future needs will be satisfied by autonomous, heterogeneous distributed information sources accessible through the internet. Such a system often maintains coverage statistics for each source based on a log of

previous user queries. With such information, it can rank the sources for a given query for faster retrieval. The subspace clustering algorithm can be applied to the query list with the queries being instances and the sources corresponding to dimensions of the dataset. The result is a rapid grouping of queries where a group represents queries coming from the same set of sources. Conceptually, each group can be considered a query class where the classes are generated in a one-step process using subspace clustering.

(2) **Web Text Mining:** A fundamental problem with organizing web sources is that web pages are not machine readable. In addition, semantic heterogeneity is a major challenge. Recently, there has been strong interest in developing ontologies to serve as a semantic, conceptual, hierarchical model representing a domain or a web page. If the web pages are presented in the form of a document-term matrix where the instances correspond to the pages and the features correspond to the keywords in the page, the result of subspace clustering will be the identification of a set of keywords (subspaces) for a given group of pages. These keyword sets can be considered to be the main concepts connecting the corresponding groups. Ultimately, the clusters would represent a domain and their corresponding subspaces would indicate the key concepts of the domain.

(3) **DNA Microarray Analysis:** DNA microarray datasets provide information on the expression levels of thousands of genes under hundreds of conditions. Currently, microarray data must be preprocessed to reduce the number of attributes before meaningful clusters can be uncovered. In addition, individual gene products have many different roles under different circumstances. Subspace clustering is a promising technique that extends the power of traditional feature selection by searching for unique subspaces for each cluster.[2]

We found an implementation of SUBCLU in R in its “subspace” library The SUBCLU algorithm uses the DBSCAN algorithm to actually cluster the dataset. "mlpack" library of C++ also includes an implementation of DBSCAN accelerated with dual-tree range search technique. “scikit-learn” includes a Python implementation of DBSCAN for arbitrary Minkowski Metrics, which can be accelerated using k-d trees and ball trees but which uses worst-case quadratic memory.[3][4]

2 RELATED WORK

2.1 Is there an existing incremental version of the selected algorithm? If yes, details of the existing incremental algorithm. You have to think how will you compete wish the existing incremental algorithm.

No, We did not find any incremental version of SUBCLU. But we still found an incremental version of DBSCAN which is a major subroutine in SUBCLU. The incremental DBSCAN algorithm first checks the ϵ neighbourhood of the newly added point, say p to find the core points which are in the neighbourhood of the core points which were non-core earlier.

(1) If no new non-core points are found, p is classified as noise.
(2) If all the new core points were noise earlier, a new cluster is created using p and these new core points.
(3) If all the new core points were border points of the same cluster, p is absorbed into that cluster
(4) but if these new core points belonged to different clusters earlier, these clusters are coalesced into a single cluster.

Similarly when a point is removed say p, we check its ϵ neighbourhood to find core points which are in the neighbourhood of the non-core points which were core points earlier.

(1) If no such points are found, p is simply removed.
(2) If such points are found and they are density reachable from each other, then some points in the neighbourhood of p will lose their membership and will be classified as noise.
(3) But if these points are not directly density reachable from each other after removing p, the cluster may split so check must be performed.

We did not find any existing incremental version of SUBCLU but we can incorporate the incremental DBSCAN into SUBCLU to do subspace clustering incrementally.[5][6].

2.2 Are there any interesting variants proposed for the selected algorithm? If yes, details of a few such variants. Knowing such variants will help you to design the incremental version in such a way that it will work on the variants as well.

Yes, there is an interesting variant of SUBCLU algorithm. SUBCLU algorithm is observed to have two bottlenecks. SUBCLU suffers from divergence of density and multi density clusters. Due to these bottlenecks, having a constant value of ϵ throughout the algorithm can lead to merging of two different clusters and inclusion of outliers in the clusters .An algorithm has been suggested to overcome these bottlenecks by dynamically computing ϵ for each cluster in the implementation of SUBCLU algorithm[7].

Another interesting variant of subclu is the combination of subclu and a restricting algorithm. Subclu generates clusters for all subspaces. Many of these can be unnecessary and require human effort to go through all the clustering. To tackle this problem a modification to generate subspaces routine in subclu has been suggested.

User provides the input of interesting attributes to the algorithm and this information is used to prune subspaces which do not have all the interesting attributes present in them. This reduces the number of subspaces with clustering and improves the quality of the output[8].

3 CODE ARCHITECTURE

3.1 Is there existing trust-able implementation available of the selected algorithm? If yes, explain the code architecture along with the data structures and class hierarchy. If no, propose your implementation plan with code architecture, data structures, and class hierarchy.

Yes there existed a trust-able implementation available of the selected algorithm inside ELKI (an open source (AGPLv3) data mining software written in Java). The code mentions the authors of the research paper in its source code. The original paper is not very clear on which clusters to return, as any subspace cluster must be part of a lower-dimensional projected cluster, so these results would be highly redundant. In this implementation, they only include points in clusters that are not already part of sub-clusters.
CODE ARCHITECTURE:

- **Data Structures:**

(1) **List and ArrayList:** The implementation uses a variety of List and ArrayList to store clusters in a subspace, valid subspaces etc.
(2) **TreeMap:** The implementation uses the TreeMap(Red-black tree data structure) to efficiently map the subspace to their respective clusters.
(3) **BitsUtil:** The implementation uses BitsUtil (Bitset data structure) to implement a subspace.
(4) **HashSet:** The implementation uses HashSet to store DBIds inside a cluster
(5) **RangeSearcher:** The implementation uses MTreeVariants to do range query searches on the dataset

- **Class Hierarchy:**
SUBCLU implements the SubspaceClusteringAlgorithm Interface on the SubspaceModel class. SubspaceClusteringAlgorithm is an extension of ClusteringAlgorithm interface which inturn is an extension of Algorithm interface. The SubspaceModel class encapsulates the Subspace class which stores a subspace in bitset format. SUBCLU takes assistance from the DBSCAN class to runDBSCAN on the valid subspaces. SUBCLU class stores the following parameters:

(1) protected DimensionSelectingSubspaceDistance<V> distance
(2) protected double epsilon
(3) protected int minpts
(4) protected int mindim

SUBCLU runs on the generic NumberVector interface that defines the methods that should be implemented by any Object that is an element of a real vector space. The entire class hierarchy can be found in the given documentation[9].

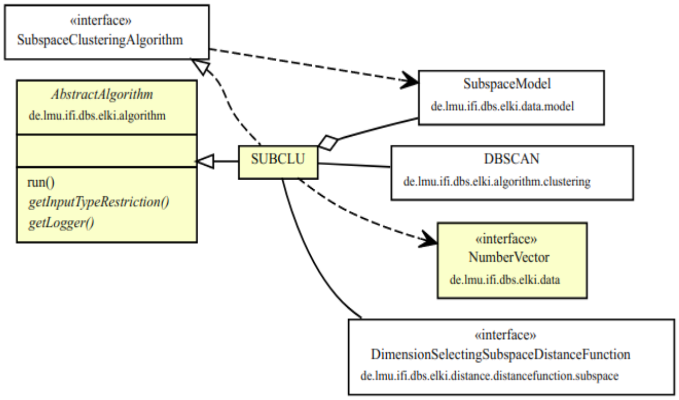

«interface»
SubspaceClusteringAlgorithm
AbstractAlgorithm
de.lmu.ifi.dbs.elki.algorithm
run()
getInputTypeRestriction()
getLogger()
SUBCLU
SubspaceModel
de.lmu.ifi.dbs.elki.data.model
DBSCAN
de.lmu.ifi.dbs.elki.algorithm.clustering
«interface»
NumberVector
de.lmu.ifi.dbs.elki.data
«interface»
DimensionSelectingSubspaceDistanceFunction
de.lmu.ifi.dbs.elki.distance.distancefunction.subspace

Figure 5: SUBCLU Class Hierarchy

The algorithm starts by call to run(Relation<V> relation) which runs various subroutines for assistance:

(1) **private List⟨Cluster⟨Model⟩⟩ runDBSCAN (Relation⟨V⟩ relation, DBIDs ids, Subspace subspace):** Runs the DBSCAN algorithm on the specified partition of the database in the given subspace.
(2) **private List⟨Subspace⟩ generateSubspaceCandidates (List⟨Subspace⟩ subspaces):** Generates d+1 dimensional candidate subspaces from d dimensional subspace.
(3) **private boolean checkLower(Subspace candidate, List⟨Subspace⟩ subspaces) :** Checks for every d+1 dimensional candidate subspaces its each d dimensional subspace has a cluster.
(4) **private Subspace bestSubspace(List⟨Subspace⟩ subspaces, Subspace candidate, TreeMap⟨Subspace, List⟨Cluster⟨Model⟩⟩⟩ clusterMap) :** Selects the bestSubspace of d dimension with minimal number of objects in the cluster for better runtime performance.

4 C++ IMPLEMENTATION

4.1 Module Organization

We have organized our codebase taking inspiration from the trustable ELKI SUBLCLU implementation in java.[9]

1. BitsUtil
   - `static vector<int> orVectors(vector<int>& a, vector<int>& b);`
   - `static int intersection(vector<int>& a, vector<int>& b);`
2. Subspace
   - `vector<int> dimensions;`
   - `int dimensionality;`
   - `Subspace(int dimension);`
   - `Subspace(vector<int>& dimensions);`
   - `bool isSubspace(Subspace& subspace);`
   - `Subspace join(Subspace& other);`
   - `bool hasDimension(int i);`
   - `void addDimension(int i);`
   - `void removeDimension(int i);`
   - `bool operator<(const Subspace &s2) const;`
3. Cluster
   - `static int cnt;`
   - `string name;`
   - `set<int> ids;`
   - `bool noise;`
   - `Subspace subspace;`
   - `vector<double> mean;`
   - `Cluster(string name, set<int> &ids, bool noise, Subspace &subspace, vector<double> &mean);`
   - `int size();`
4. Relation
5. ReadInput
   - `ifstream inputFile;`
   - `ReadInput(string file);`
   - `~ReadInput();`
   - `Relation<double> read()`
6. DBSCAN
   - `Relation<double> m_points;`
   - `double m_eps;`
   - `uint m_minPts;`
   - `vector<int> m_clusterIDs;`
   - `uint m_numPoints;`
   - `Subspace m_subspace;`
   - `map<vector<double>, int> m_ids;`
   - `int expandCluster(int, uint);`
   - `vector<int> rangeQuery(vector<double>);`
   - `double dist(vector<double>, vector<double>);`
   - `vector<double> getMean(vector<int> &v);`
   - `vector<Cluster> getClusters();`
7. SUBCLU
   - `double epsilon;`
   - `int minPnts;`
   - `int minDim;`
   - `Relation<double> dataBase;`
   - `map<Subspace, vector<Cluster>> Clustering;`
   - `map<vector<double>, int> dbids;`
   - `map<Subspace, vector<Cluster>> run();`
   - `vector<Cluster> runDBSCAN(Subspace &currSubspace, set<int> &ids);`
   - `vector<Subspace> generateSubspaceCandidates(vector<Subspace> &subspaces);`
   - `Subspace besttSubspace(vector<Subspace> &subspaces, Subspace &candidate);`
   - `bool checkLower(Subspace &candidate, vector<Subspace> &subspaces);`

4.2 Class Hierarchy

BitsUtil and Relation are the basic classes. Subspace is built using BitsUtil, in turn Cluster is built using Subspace Class. DBSCAN utilizes Cluster and Subspace Classes. Finally SUBCLU utilizes all ReadInput, Cluster, Spuspace and DBSCAN classes. Figure 6 provides a better understanding of the hierarchy.

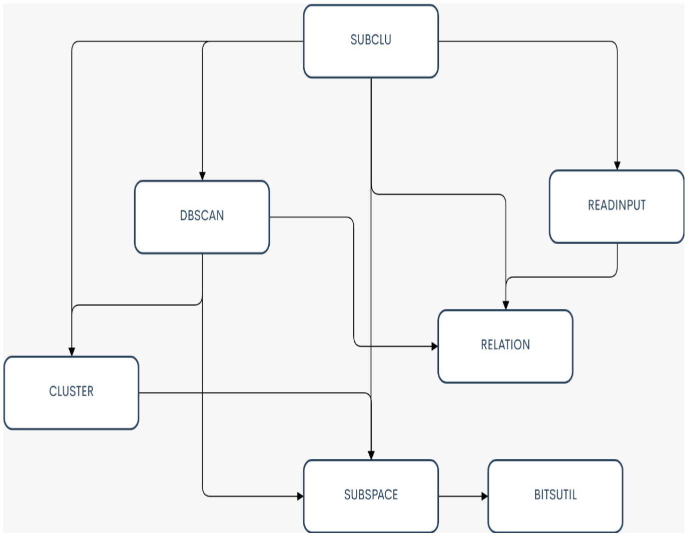


Figure 6: Hierarchy

4.3 Class Description

1. **Subspace**
   - (a) Subspace is implemented using a vector to store the dimensions and variable to store the number of dimensions.
   - (b) A value 1 in ith position in vector denotes that ith dimension is present and the value of missing dimensions is 0.
2. **Cluster**
   - (a) A cluster is represented using a name, IDs of points in the cluster and subspace of the cluster.
   - (b) IDs of points are stored in a set data structure. ID of a point denotes it's position (index) in the dataset.
   - (c) Cluster also maintains a variable noise to distinguish all the noise from the clusters.
3. **Relation**
   - (a) Each data point in the Relation is represented as a vector of attributes.
   - (b) All the data points are structured in a vector to form the dataset (Relation)
4. **DBSCAN**
   - (a) DBSCAN takes data, subspace, epsilon and minimum points as inputs. Then returns a vector of clusters.
   - (b) Our implementation of DBSCAN has quadratic time complexity. That is, for processing n points of d dimension time taken is in O(n^{2}d).
5. **SUBCLU**
   - (a) SUBCLU takes file name, epsilon , minimum points and minimum dimension as input. Then returns map of subspace and corresponding clusters.
   - (b) SUBCLU maintains a map of each point to it's ID and also a map of each subspace to it's clusters. Map here is implemented using red-black tree and is present in stl library of c++.

5 EVALUATION OF C++ IMPLEMENTATION

5.1 Code Walkthrough

- *Main*
  Main.cpp starts with the creation of the *SUBCLU* module with appropriate dataset, *minpoints* and ϵ and then calls $SUBCLU.run()$ to obtain clusters in all the sub-spaces. Then prints the clustering to various files used for testing.
- *SUBCLU*
  SUBCLU begins with finding the clusters in 1-D database by calling $DBSCAN.getClusters()$. Then follows Apriori method to build the future clusters. This begins with calling generateCandidates() function to produce high dimensional sub-spaces. Then best sub-space for each high dimensional sub-space is decided using $bestSubspace()$ function. Then clusters for each new sub-space are calculated using $runDBSCAN()$ function using the best sub-space clusters.Then sub-spaces with at least one cluster are added to subspace vector.
  This Apriori process is repeated until all sub-spaces are exhausted.
- *DBSCAN*
  DBSCAN module is used to find the clusters in a particular subspace. It first classifies all the points as $UNCLASSIFIED$ and then sequentially runs $expandCluster()$ to classify the points as noise or and element of a cluster. It uses *Euclidean Distance* to measure distance between two points. It returns a *vector* of clusters.[12]

5.2 Testing Measures

To calculate the quality of our C++ implemention of SUBCLU algorithm we have used a score metric called Silhouette Coefficient or Silhouette Score.

The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

For data point $i \epsilon C_i$ let

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \epsilon C_i, i \neq j} d(i, j) \qquad (1)$$

where d(i, j) is the distance between data points i and j. a(i) is a measure of how ell i is assigned to its cluster.

For data point $i \epsilon C_i$ let

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \epsilon C_k} d(i, j) \qquad (2)$$

b(i) is a measure of how poorly the data point is matched to its neighbouring clusters.

Silhoutte value s(i) is defined as

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}, \text{if } |C_i| > 1 \qquad (3)$$

$$s(i) = 0, \text{if } |C_i| = 1 \qquad (4)$$

The Silhouette Coefficient is the maximum value of the mean Silhoutte value per k cluster over all data of the entire dataset.[10]

Confusion Matrix was also used for testing. In contrast to silhouette, confusion matrix measures similarity between computed and actual clustering. In confusion matrix each row represents the instances in a predicted class, while each column represents the instances in the actual class. The matrix shows the precision of the clustering algorithm from a single look.

5.3 Toy Dataset Used

We have used 2 toy Dataset to demonstrate SUBCLU.

1. **Iris flower data set:**
   The data set consists of 50 samples from each of three species of Iris (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.[11]

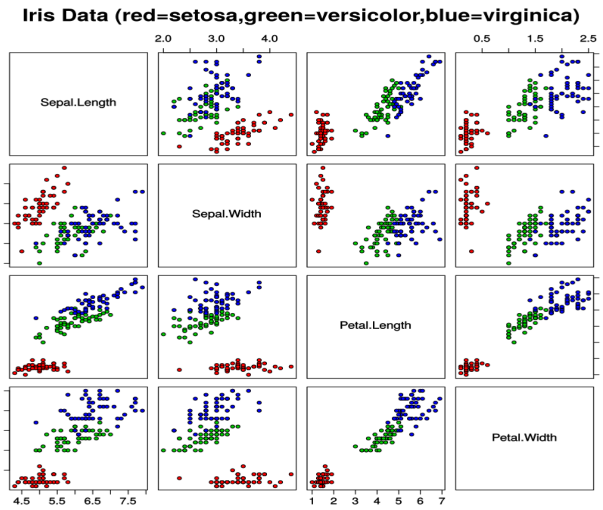


Figure 7: Iris flower data set

2. **Synthetic data set:**
   The data set consists of 600 samples with 3 dimensions. The data set represents a total 6 cylinders in the 3-D space. 2 Cylinder parallel to xy plane, 2 parallel to xz plane and 2 parallel to yz plane. The data set is specifically constructed so that there is no cluster in the 3 dimension space but several clusters in the lower dimension.

5.4 Working of the code on toy dataset

1. **Iris flower data set**
   Initially clusters on each 1-D sub-space is calculated. Three sub-spaces produce one cluster and one sub-space produces 2 clusters. Since all 1-D sub-spaces at least one clusters, all 6 2-D sub-spaces are produced. Then it is observed that each subspace contains at least one cluster to at most 4 clusters.

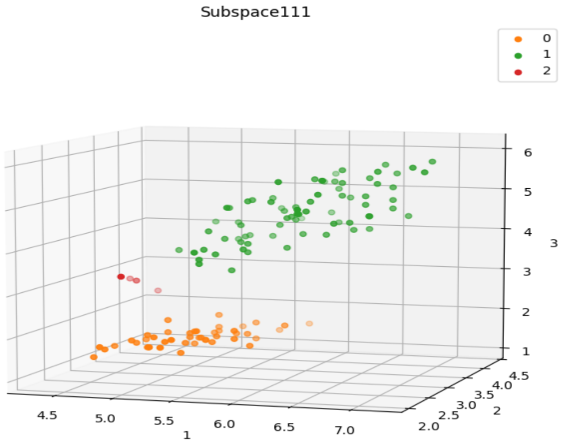


Figure 8: iris-1110

Again all 4 3-D clusters are produced as all 2-D clusters have at least one cluster. In 3-D sub-space two sub-spaces have 2 clusters, one sub-space with 4 clusters and one with 3 clusters. Finally in the 4-D space 3 clusters are identified. Figure 9

(2) **Synthetic data set**
Initially clusters on each 1-D sub-space is calculated. Two sub-spaces produced 2 clusters and one sub-space produced 4 clusters. Then clusters in all 2-D sub-spaces are calculated. Each of the 2-D subspace produced two clusters. Finally on the 3-D space no cluster was formed.

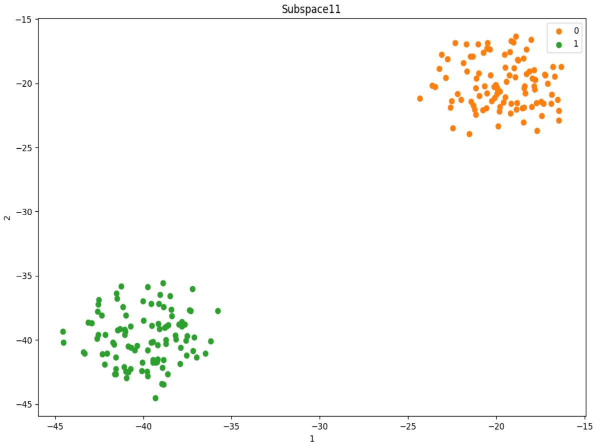


Figure 9: disk-110

5.5 Results

We ran SUBCLU on Iris flower data set with *minPnts* = 4 and $\epsilon = 0.4$ and found pretty good clusters.

The Silhoutte coefficients were found to be positive and pretty close to 1.

SUBSPACE	SILHOUTTE COEFF.
Subspace0011.csv	0.7636385866059445
Subspace0111.csv	0.7457831575244798
Subspace1011.csv	0.44092212189635116
Subspace1111.csv	0.477243627675687
Subspace101.csv	0.4472828927140069
Subspace011.csv	0.7514993620555627
Subspace111.csv	0.543086711268547
Subspace1101.csv	0.5624266451800497
Subspace0101.csv	0.6337012123223621
Subspace001.csv	0.7818150679001757

We also ran SUBCLU on our sythetic data set with *minPnts* = 4 and $\epsilon = 4$ and found extremely good clusters.

The Silhoutte coefficient were found to be > 0.87 in each subspace.

SUBSPACE	SILHOUTTE COEFF.
Subspace01.csv	0.8858785730320692
Subspace1.csv	0.8918129320193205
Subspace11.csv	0.8780394220364249
Subspace001.csv	0.8889065129411609
Subspace011.csv	0.8778008283618736
Subspace101.csv	0.8774749747464436

6 INCREMENTAL ALGORITHM-ITERATION 1

6.1 Description of static algorithm

Algorithm 1: SUBCLU

```
Result: Generate clusters in all possible sub-spaces
vector<Subspaces> S;
for each 1-D subspace do
    Find clustering in 1-D sub-spaces using DBSCAN;
    if clusters are present then
        add subspace to S;
    end
end
for $d = 2;\ d < deminstions;\ d{+}{+}$ do
    /* generate d dimension subspaces from S    */
    vector<Subspace> candiates = generateSubspaces(S);
    Initalize vector<Subspaces> Sub_d;
    for each cadidate in candidates do
        Find the bestSubspace of the candidate;
        Initialize vector<Cluster>C_clusters;
        for each cluster in bestSubspace do
            run DBSCAN on cluster;
            Add clusters generated to C_cluster;
        end
        if C_clusters is not empty then
            Add candidate to Sub_d;
        end
    end
    S = Sub_d;
    if S is empty then
        break;
    end
end
```

6.2 Description of Incremental DBSCAN

(1) **Insertion**
First a set UpdSeed is computed after insertion and deletion of points. UpdSeed is a set of core points which are directly reachable from new core points formed after insertion of points.
- If UpdSeed is empty, then the inserted point is noise and no change is needed.
- If points in UpdSeed do not belong to a cluster then a new cluster is created along with p
- If the points in UpdSeed are from same cluster then inserted point is absorbed into that cluster along with some noise possibly.
- If the points in UpdSeed are from different clusters then these clusters are merged to form a single cluster.

(2) **Deletion**
First a set UpdSeed is computed after insertion and deletion of points. UpdSeed is a set of core points which are directly reachable from points which were core points before deletion but do not remain core points after deletion.
- If UpdSeed is empty, then the cluster corresponding to p is removed.
- If all points in UpdSeed are directly reachable from each other then few points in UpdSeed reduce to noise after removal of the deleted point.
- If all the points are not directly density reachable then there is a chance for potential split.

Remarks: From the above two ideas in the incremental dbscan we can observe that only a small section of points are being affected on inserting a new point i.e points which are in the epsilon neighbourhood of inserted/deleted point and their directly reachable points. Computation on these small numbers of points results in speedup of the algorithm compared to running complete dbscan again.

6.3 Intuition behind Incremental SUBCLU and related Theorems

Since the algorithm is double-stepped i.e DBSCAN over apriori, we saw two ways to move forward. One being incremental apriori with vanilla DBSCAN. The second being Incremental DBSCANs connected by an apriori algorithm. Since, DBSCAN is the primary algorithm on which the clustering is based on, and also because this made more intuitive sense to us, we chose to move along the second path.

(1) The Incremental DBSCAN Paper (Incremental Clustering for Mining in a Data Warehousing Environment, Ester et al.)[13] guarantees that the clusters it returns are the same as the ones that would have been returned if vanilla DBSCAN would have been run all over again.
(2) A space may change its cluster structure only if there is change in cluster structures in all its subspaces. Because if there is a subspace which has not changed, we can run DBSCAN with this subspace as the bestSubspace resulting in exactly the same clusters as before.

Algorithm 2: Incremental DBSCAN

```
Data: Clusters C, Real $\epsilon$, Integer m, ChangedObject p
Result: Updated clusters in all sub-spaces
Bool Change ;                                  // Change Boolean
SetOfObjects DB = getObjects(C);
UpdSeed = $\Phi$;
$N_p$ = RangeQuery(DB $\cup$ {p}, p, $\epsilon$);
for each $q' \in N_p$ do
    $N_{q'}$ = RangeQuery(DB $\cup$ {p}, q', $\epsilon$);
    $N_q$ = RangeQuery(DB, q', $\epsilon$);
    if $N_{q'} \geq m$ & $N_q < m$ & p.type == " + " then
        UpdSeed = Updseed $\cup$
          DirectlyReachableCorePoints(q');
    else if $N_{q'} < m$ & $N_q \geq m$ & p.type == " − " then
        UpdSeed = Updseed $\cup$
          DirectlyReachableCorePoints(q');
    end
end
if p.type == " + " then
    if UpdSeed == $\Phi$ then
        Change = false;
    else if UdpSeed $\cap$ DB == {p} OR UdpSeed $\cap$ DB == $\Phi$
      then
        Change = true;
        Cluster c = newCluster(UpdSeed); // new cluster
    else if ||UdpSeed.IDs()|| == 1 then
        Change = true;
        expandCluster(UdpSeed);                 // Absorb
    else
        Change = true;
        for each c in C do
            if c.ID in UdpSeed.IDs() then
                C = C-c;
            end
        end
        Cluster c = newCluster(UdpSeed);
        C = $\cup$ c;                                // Merge
    end
else
    if isNoise(p) then
        Change = false;
    else
        Change = true;
        if UdpSeed== $\Phi$ then
            Cluster c = clusterOf(p);
            C = C - c;                          // Removal
        else if DirectDensityReachable(UpdSeed) then
            Cluster c = clusterOf(p);           // Reduction
            for each q in $N_p$ do
                $N_q$ = RangeQuery(DB-{p},q,$\epsilon$); if
                  corePoint($N_q$==0 then
                    c = c-{q};                  // q becomes noise
                end
            end
        else
            checkSplit(UpdSeed);
        end
    end
end
```

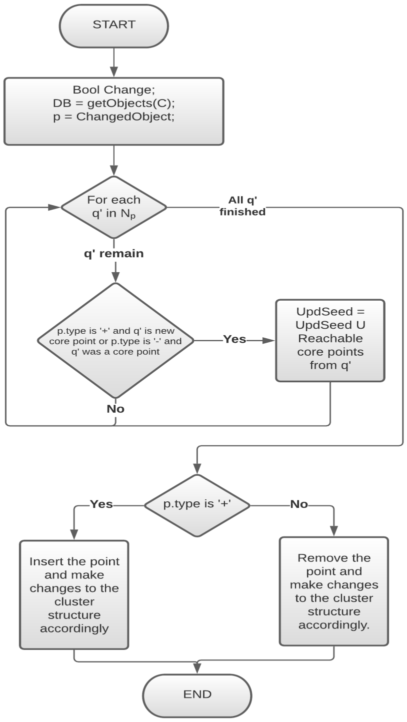

START
Bool Change;
DB = getObjects(C);
p = ChangedObject;
For each q' in N_p
All q' finished
q' remain
p.type is '+' and q' is new core point or p.type is '-' and q' was a core point
Yes
UpdSeed = UpdSeed U Reachable core points from q'
No
p.type is '+'
Yes
No
Insert the point and make changes to the cluster structure accordingly
Remove the point and make changes to the cluster structure accordingly.
END

Figure 10: :Incremental DBSCAN

I claim that our incremental version of SUBCLU will return the same clusters as the static version because:

- ensures all individual Subspaces have the same results as that of the Static version
- ensures the apriori part of the algorithm is not affected.

As a direct result of the above, we believe that the proposed algorithm will work.

Remark. To minimize the cost of the runs of DBSCAN in cand, we choose that subspace bestSubspace ⊂ cand from S_k in which a minimum number of objects are in the cluster,This heuristics only minimize the number of range queries necessary during the runs of DBSCAN in S and has no other effect on the results of the algorithm.

THEOREM 6.1. *If there is a change in a (k + 1)-D space then there is a change in all its k-D subspaces*

Proof: Contrapositive of the above statement is as follows:

Contrapositive. *If there is no change in at least one of the k-D subspaces then the (k + 1)-D space does not change.*

Proof by contradiction
Assume that a (k + 1)-D space S changed its cluster structure due to insertion/deletion but there exists a k-D subspace T with no change in its cluster structure. Let C be the cluster structure of S before change and C' after the change. C can be obtained by running DBSCAN on the clusters of T 6.3(See Remark). This property holds true even after deletion/insertion of points. But we already have C' as the cluster structure of S which is different from C. This is not possible. Hence contradiction.
=><=

COROLLARY 6.2. *If there is a change in a space then there is a change in all its subspaces.*

Incremental SUBCLU uses the above theorem to prune subspaces.

6.4 Incremental Algorithm Flow and Pseudocode

We start by running incDBSCAN on all 1-D Subspaces and storing all the subspaces with cluster changes in them (new subspaces with clusters might be created, old subspaces might lose all their clusters and existing subspaces might have modifications in their clusters).

Now generate candidate (k+1)-D subspaces with these k-D subspaces with changes combined with each other. Run Incremental DBSCAN on these (k+1)-D candidate Subspaces and store all the subspaces with changes in cluster structure in them and repeat this step until convergence.

Algorithm 3: Incremental SUBCLU

```
Data: Real $\epsilon$, Integer m, NewObject p
Result: Updated clusters in all sub-spaces
/* STEP 1 Update all 1-D clusters                    */
$dS_1 = \Phi$ ;    // 1-D subspaces with changes in them
for each $a_i \in A$ do
    Clusters $Ca_i$ = getClusters($a_i$) ; // original clusters
     found by DBSCAN
    Bool change = IncDBSCAN($Ca_i$,{$a_i$},$\epsilon$,m,p);
    if change then
        $dS_1 = dS_1 \cup \{a_i\}$;
    end
end
/* STEP 2 * Update (k + 1)-D clusters from k-D
   clusters                                          */
k = 1;
while $dS_k != \Phi$ do
    /* STEP 2.1 Generate (k+1)-D candidate
       subspaces                                     */
    $dS_{k+1}$ = GenerateCandidateSubspaces($dS_k$);
    /* Test and Update (k+1)-D subspaces             */
    for each cand $\in dS_{k+1}$ do
        Clusters $C_{cand}$ = getClusters(cand);
        Bool change = IncDBSCAN($C_{cand}$,cand,$\epsilon$,m,p);
        if change == false then
            $dS_{k+1} = dS_{k+1}$-{cand};
        end
    end
    k = k+1;
end
```

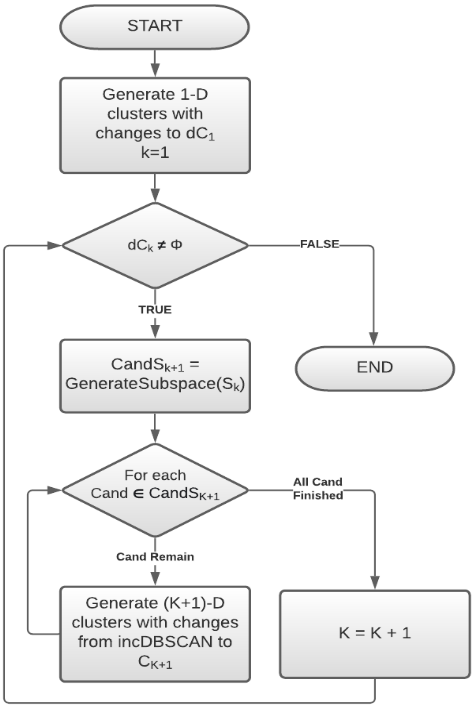

START
Generate 1-D clusters with changes to dC₁ k=1
dCₖ ≠ Φ
FALSE
TRUE
CandSₖ₊₁ = GenerateSubspace(Sₖ)
END
For each Cand ∈ CandSₖ₊₁
All Cand Finished
Cand Remain
Generate (K+1)-D clusters with changes from incDBSCAN to Cₖ₊₁
K = K + 1

Figure 11: :Incremental SUBCLU

6.5 Performance Evaluation of Incremental SUBCLU Algorithm

We are planning to test the performance of our Incremental algorithm:

- in case of only addition, only deletion and both.
- on dataset with large data
- on dataset with large dimensionality

Time for computation will be compared as the algorithm is 100% accurate as SUBCLU.

Reasons for speedup:

(1) **Pruning of subspaces :** The incremental algorithm will only focus on the subspaces where possible change can be observed, reducing redundant computations on the other subspaces.
(2) **Using Incremental DBSCAN:** In the subspaces where changes are observed , an incremental DBSCAN algorithm is used to speed up the computation.

7 INCREMENTAL ALGORITHM-ITERATION 2

7.1 Description of Modified Incremental DBSCAN Algorithm

For each point maintain the count of number of points in its epsilon neighbourhood.We plan to restrict the changes to the new points neighbourhood only.

(1) **Point Insertion** For each Subspace: get the points in the neighbourhood of the inserted point and increment their count by 1.

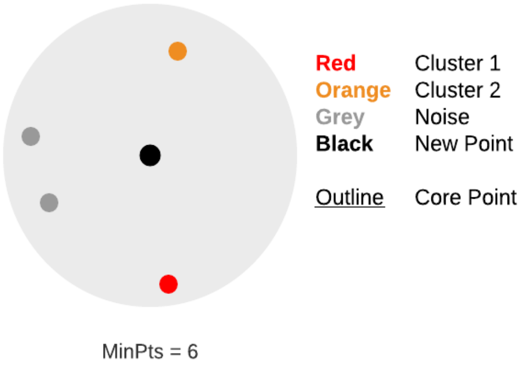


Figure 12: insertion 1

- If neighbourhood has no core point then label inserted point as noise.

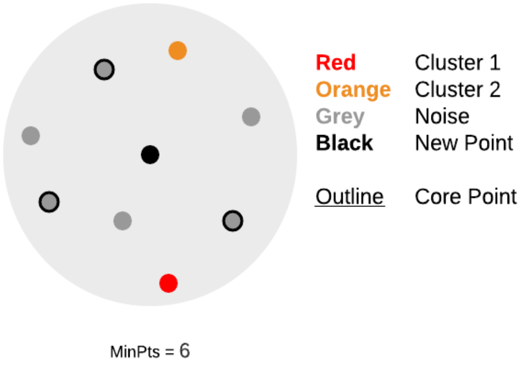


Figure 13: insertion 2

- If some points in neighbourhood are core points:
  - If the points are all noise: Form a new cluster with all the noise points in the neighbourhood of the new point and extend the neighbourhood of the core points.
  - If the points in neighbourhood belong to a single cluster and some noise points: Add all the noise points in the neighbourhood to that cluster if inserted point is a core point, otherwise add only the inserted point to the cluster.
  - If the points belong to different clusters: Merge all those clusters into a single cluster if the if inserted point is a core point, otherwise add only the new point arbitrarily to any of those clusters.

(2) **Batch Insertion**
- In each subspace apply DBSCAN on the new points to be inserted:

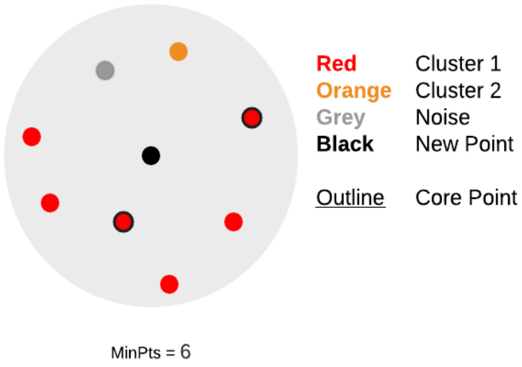


Figure 14: insertion 3

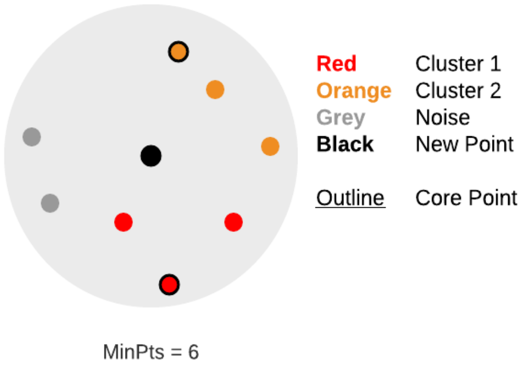


Figure 15: insertion 4

- For each cluster run the Incremental DBSCAN on mean of the cluster using the above mentioned Point Insertion method.
  - If the mean is labeled noise then the cluster exists as an independent cluster in the subspace.
  - if the mean is a part of a cluster then all the points in the mean's cluster are added to the cluster, that mean is a part of.
- For all the noise points run Point Insertion algorithm separately.

(3) **Delete** For each Subspace: get the points in the neighbourhood of the deleted point and decrement their count by 1.
- If the deleted point is a core point (i.e $cnt >= minpts$): Temporarily, mark all points in the deleted point's neighbourhood as noise. Then, assign these points to the cluster of the nearest core point (with distance < epsilon) if such core point exists. If none does, then the point stays as noise.
- If the deleted point is not a core point:
  - If none of the points in the neighbourhood loose their core property due to deletion, then simply remove the point to be deleted and do no perform any changes.

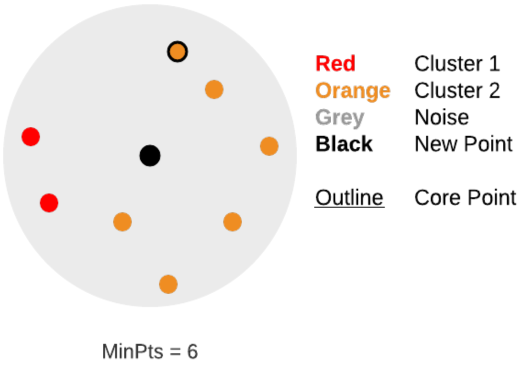


Figure 16: deletion 1

  - If points in the neighbourhood loose their core property, then run part 1 considering each of these points to be deleted.

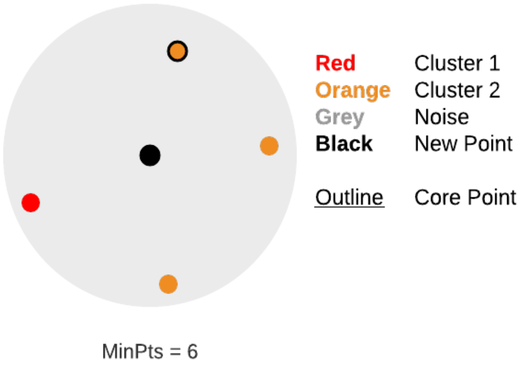


Figure 17: deletion 2

- A count is maintained for each cluster which is incremented every time a deleted point creates the possibility of a potential split. When this count reaches a particular threshold (specified as a parameter), we run a static-DBSCAN on this cluster so that all the missed splits are accounted for.

7.2 Memory usage of static algorithm

Data	Memory(KB)	time(sec)
2D-500	4256	0.48
3D-400	4424	0.57
4D-150	4164	0.27
8D-500	16056	44.03
8D-1500	39908	326.34
16D-100	776128	954.00

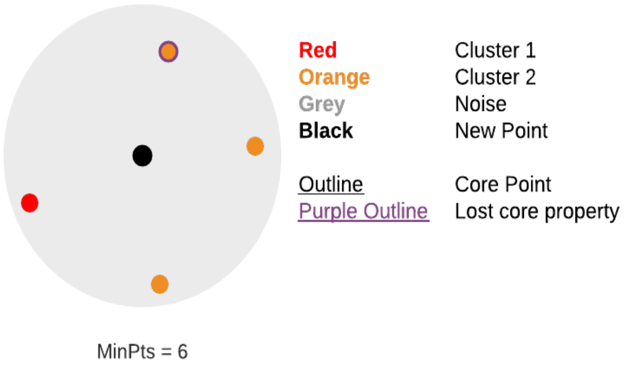


Figure 18: deletion 3

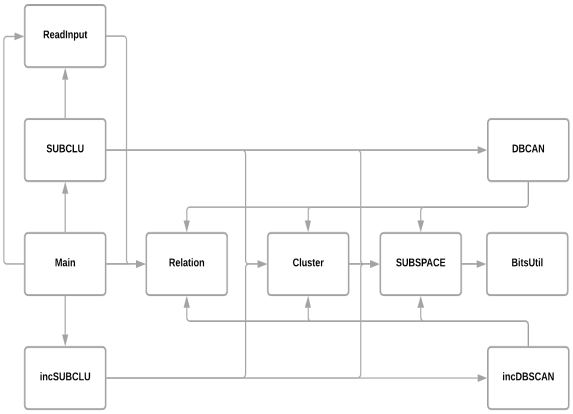


Figure 19: Class Hierarchy

8 INCREMENTAL ALGORITHM-EXPERIMENTAL RESULTS, ITERATION 1

8.1 Dataset used

For our first iteration of experimentation we used synthetically created datasets to check for our implementation correctness.

(1) **INSERTION:**
We started our dataset with 100 random points inside a cube centered at (0,0,0) and of side 2 units.
 (a) **New Cluster:**
 We took 100 new random points inside a cube centered at (3,3,3) and of side 2 units and inserted each point one by one in the original dataset.
 (b) **Merge Clusters:**
 After the new cluster has been inserted we inserted another 100 random points inside a cube centered at (2,2,2) and of side 4 units so that the original two cubes combine with it to form a single cluster.
 (c) **Random Points Addition:**
 After the cubes have been merged we inserted 100 more random points randomly inside a cube centered at (2,2,2) and of side 8 units.

(2) **DELETION:**
We used separate starting dataset for each delete case.
 (a) **Reduce Cluster Size:**
 We started with 200 random points inside a cube centered at (2,2,2) and of side 2 units. We then selected 20 random points from it and deleted those points from it one by one.
 (b) **Split Clusters:**
 We started with 3 cubes of side 2 units centered at (0,0,0), (1,1,1) and (2,2,2) respectively containing 100 random points each. These initially formed a single cluster. We then removed the points from the middle cube so as to disconnect the other two clusters.

Note: All observations and visualisations are based on the 3D Subspace111.csv

8.2 Testing Measures

Table 1: Memory and Time

Data	St(KB)	St(S)	Inc(KB)	Inc(S)
cube_new.csv	4300	5.41	4132	0.41
cube_bigger.csv	4396	55.67	4404	1.78
cube_merge.csv	4516	24.7	4240	0.98
cuboid_del.csv	4144	2.18	4308	0.253
cuboid_split.csv	4248	7.61	4488	0.93

Table 2: Silhouette Constant for Incremental Run

Subspace	1	01	001	11	101	011	111
cube_new.csv	0.76	0.76	0.77	0.75	0.75	0.75	0.75
cube_bigger.csv	1	1	1	1	1	1	0.098
cube_merge.csv	1	1	1	1	1	1	1
cuboid_del.csv	1	1	1	1	1	1	1
cuboid_split.csv	1	1	1	1	1	1	0.37

Table 3: Silhouette Constant for Static Run

Subspace	1	01	001	11	101	011	111
cube_new.csv	0.76	0.76	0.77	0.75	0.75	0.75	0.75
cube_bigger.csv	1	1	1	1	1	1	0.098
cube_merge.csv	1	1	1	1	1	1	1
cuboid_del.csv	1	1	1	1	1	1	1
cuboid_split.csv	1	1	1	1	1	1	0.37

8.3 Observations and Conclusions

(1) There is a significant improvement in the running time of the incremental SUBCLU which was expected from the incremental DBSCAN approximations.
(2) The memory consumption of incremental SUBCLU is also more or less the same as static SUBCLU because we are only exploring those subspaces which have clusters in all of their subspaces.

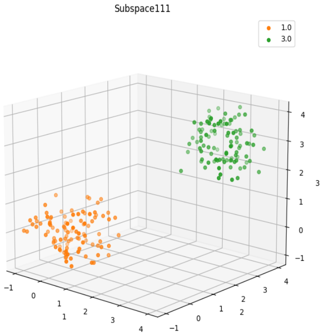


Figure 20: Insert new cluster

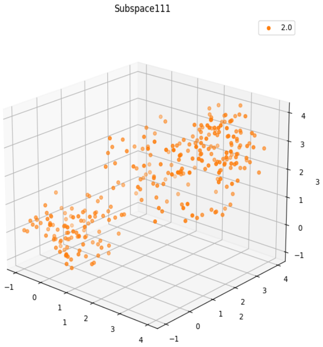


Figure 21: Merging between clusters

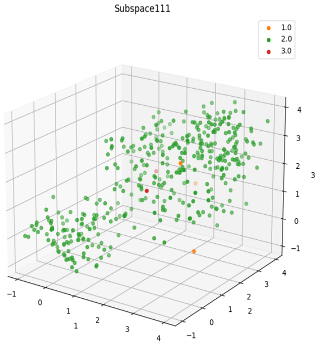


Figure 22: Add few random points

(3) The synthetic datasets were specifically made to cover all the cases of our incremental DBSCAN and the implementation worked nearly perfectly for all the cases.

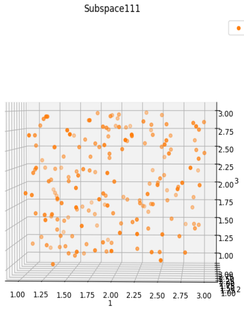


Figure 23: Delete few random points

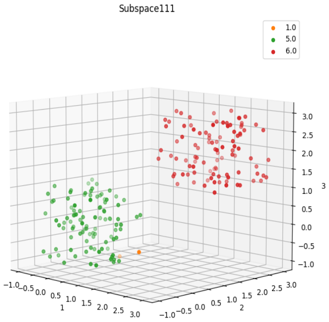


Figure 24: Split two clusters

(4) The expected potential split cases were very few in split cluster case, we expected more. We can try generating more datasets to verify this fact in future experimentations.

8.4 Further Experimentation Plans

(1) Currently testing has been performed on toy dataset. We need to extend this to real world datasets.
(2) So far, testing involved a bit of hard coding in each test. We plan to build a streamlined program module to perform all the testing automatically for us.
(3) Until now, testing can only handle all insertions or all deletions we are planning to create tests with mixed insertions and deletions.
(4) We plan to further improve on the efficiency by pruning the epsilon neighbourhood.

9 INCREMENTAL ALGORITHM-EXPERIMENTAL RESULTS, ITERATION 2

9.1 Accuracy

Having generated clusters for both static run as well as incremental run of the algorithm, we need to be able to tell how well incremental run performs compared to the static run. For this, for each cluster in the static run, we look for the most similar cluster in the clusters formed by the incremental run. The similarity measure is calculated by counting the number of points common to both the clusters. The number of common points are classified as *Correct predictions* while the rest are classified as *Incorrect predictions*. In case two or more different clusters formed by static run has the same most similar cluster in the clusters formed by the incremental run, then the one with most number of common points is assigned that cluster while the rest are assigned to the next most similar cluster. Ties are broken by the assigning it to the cluster with the least number of total points. If no more clusters can be assigned to any cluster all its points are considered *Incorrect predictions*.

9.2 Testing Module

So for testing each dataset we created two files namely dataset_initial.csv and dataset_update.csv. By default, both the testing scripts runs static SUBCLU on the dataset_initial.csv. The update file has points to be inserted to or deleted from the dataset, these points are differentiated by an extra column which has +1 for insertion and -1 for deletion. The testing script for incremental SUBCLU reads one point at a time from the dataset_update.csv file, runs incremental SUBCLU and write backs the clusters for each Subspace.csv. After running incremental subclu for each point the script returns the time taken for the execution, memory usage and silhouette coefficients for each Subspace. The testing script for static SUBCLU reads one point at a time from the dataset_update.csv file and create new dataset file containing all the updates until that point. It then runs static SUBCLU for each created dataset and finally returns the time taken, memory usage and silhoute coefficients for each Subspace. Then the accuracy.py script is used to compare the two subspaces of the static SUBCLU results and incremental SUBCLU results.

9.3 Dataset Description

(1) **Seeds Dataset**
No. of Attributes - 7
Data points - Initial: 168
Update: Insert: 42 , Delete: 24
Data Collection Year - 2012
Real Valued.
Measurements of geometrical properties of kernels belonging to three different varieties of wheat. A soft X-ray technique and GRAINS package were used to construct all seven, real-valued attributes.

(2) **User Knowledge Modelling Dataset**
No. of Attributes - 5
Data points - Initial: 328
Update: Insert: 75 , Delete: 70
Data Collection Year - 2013
Integer Valued
It is the real dataset about the students' knowledge status about the subject of Electrical DC Machines. The dataset had been obtained from Ph.D. Thesis of students

(3) **Iris Dataset**
No. of Attributes - 5
Data points - Initial: 130
Update: Insert: 20 , Delete: 30
Data Collection Year - 1936
Real Valued
The dataset contains a set of 150 records under five attributes - sepal length, sepal width, petal length, petal width and species to distinguish between three species of Iris - Iris setosa, Iris virginica and Iris versicolor.

(4) **Mouse (ELKI)**
No. of Attributes - 2
Data points - Initial: 400
Update: Insert: 100 , Delete: 36
Real Valued
The Dataset has been used as a toy dataset for comparing K-means and EM clustering in the ELKI Library. It has 4 clusters, namely head, ear-left, ear-right and noise.

9.4 Results

System Specs
Processor: Intel® Core™ i5-8250U CPU @ 1.60GHz × 8
RAM: 8GB
Cores: 8
Swap Space: 2GB

Table 4: Memory and Time

Data	St(KB)	St(S)	Inc(KB)	Inc(S)
test.csv	5832	236.78	6016	10
test_insert.csv	6288	141.19	5784	5.37
Iris_update.csv	4324	4.58	4308	0.57
seeds_update.csv	8384	118.66	7648	8.97
Mouse_update.csv	4272	28.6	4528	1.03

Table 5: Accuracy

Data	Static	Incr	Purity	MinPoints	Epslion
test.csv	1	1	0.78	15	0.25
test_insert.csv	0.34	0.32	0.85	15	0.25
Iris_update.csv	0.46	0.46	0.97	4	0.4
seeds_update.csv	0.47	0.36	0.96	4	0.7
Mouse_update.csv	0.57	0.52	0.90	12	0.05

9.5 Conclusion

(1) Time : Incremental SUBCLU is consistently performing better in time compared to static SUBLU. The reason being,

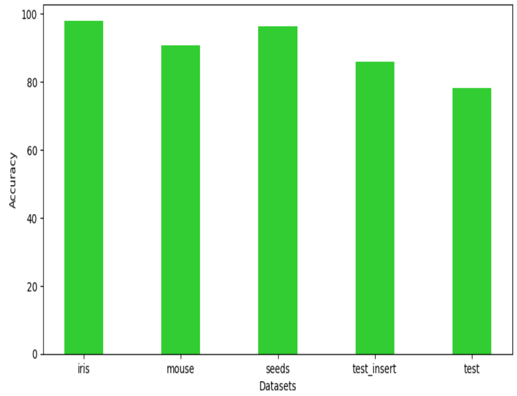


Figure 25: Accuracy bar graph

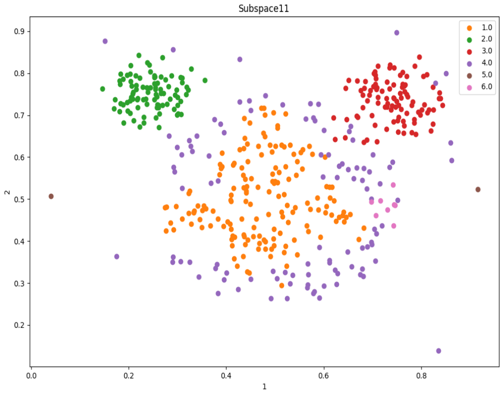


Figure 26: Mouse.csv Incremental Subspace11.csv

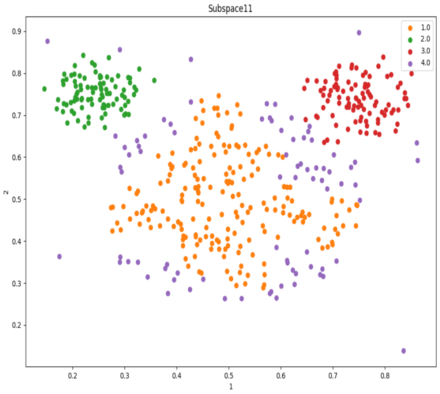


Figure 27: Mouse.csv Static Subspace11.csv

for both insert and delete, incremental algorithm is only updating the epsilon neighbourhood points.

(2) Memory : Incremental SUBLU and static SUBLU are performing very similar in terms of memory consumption. Data regarding subspace data consumes maximum amount of memory. Since both the algorithms on reaching the end obtain similar subspace data, they have very similar memory.

(3) Accuracy : Incremental SUBLU is not completely accurate. Since the update is performed for the points in epsilon neighbourhood, points in the epsilon neighbourhood of epsilon neighbourhood are not updated. This leads to noise points being considered as cluster points and vice versa, leading to a loss of accuracy.

9.6 Limitations

(1) Due to low end specifications of our test machine, testing on high dimensional and large data-sets was not feasible.

(2) In sub-spaces, clusters store id of each point and id is the location a point in the dataBase array. After deleting a point, the point is being replaced with zeros and not really deleted from the dataBase as this would change id of all the points in the clusters.

(3) Same min points and epsilon values are being used for all the sub-spaces. This is leading to inaccurate results in the smaller dimension sub-spaces.

REFERENCES

[1] Gene Expression Dataset,
https://www.molbiolcell.org/doi/suppl/10.1091/mbc.9.12.3273

[2] Example reference,
https://www.kdd.org/exploration_files/parsons.pdf

[3] Density-based spatial clustering of applications with noise(DBSCAN),
https://en.wikipedia.org/wiki/DBSCAN

[4] SUBCLU R data analysis package,
https://rdrr.io/cran/subspace/man/SubClu.html

[5] Enhanced incremental DBSCAN,
https://www.sciencedirect.com/science/article/pii/S1110016815001489

[6] Incremental DBSCAN,
https://www.researchgate.net/publication/281556454_Incremental_DBSCAN_for_Green_Computing

[7] SUBCLU variant 1,
http://j.mecs-press.net/ijitcs/ijitcs-v9-n6/IJITCS-V9-N6-4.pdf

[8] SUBCLU variant 2,
https://www.scielo.sa.cr/scielo.php?pid=S0379-39822018000300074&script=sci_arttext&tlng=en

[9] ELKI SUBCLU documentation,
https://elki-project.github.io/releases/current/doc/de/lmu/ifi/dbs/elki/algorithm/clustering/subspace/SUBCLU.html

[10] Silhoutte Coefficient,
https://en.wikipedia.org/wiki/Silhouette_(clustering)

[11] IRIS dataset,
https://archive.ics.uci.edu/ml/datasets/Iris

[12] Static Code Implementation
https://github.com/bg2404/CS568-Data-Mining

[13] Incremental DBSCAN algorithm
https://www.dbs.ifi.lmu.de/Publikationen/Papers/VLDB-98-IncDBSCAN.pdf