CLIQUE: Automatic Subspace Clustering of High Dimension Data for Data Mining Applications

Incremental Data Miners
Suraj Kesharwani 204161016, Vikash Chauhan 204161020,
Garima Singh 204161007, Ashita Batra 206101102

Keywords: CLIQUE, Grid-based, Subspace, Dimesionality,Clustering

Abstract

Data mining applications place special requirements on clustering algorithms including: the ability to find clusters embedded in subspaces of high dimensional data, scalability, end-user comprehensibility of the results, non-presumption of any canonical data distribution, and insensitivity to the order of input records. We present an incremental algorithm for the static clustering algorithm iė. CLIQUE which satisfies above requirements. CLIQUE identifies dense clusters in subspaces of maximum dimensionality. It generates cluster descriptions in the form of DNF expressions that are minimized for ease of comprehension. It produces identical results irrespective of the order in which input records are presented and does not presume any specific mathematical form for data distribution.Here,we have build an incremental algorithm for CLIQUE in which we have performed addition of points, deletion of points and addition followed by deletion.Through experiments, we observe that incremental approach we are able to get satisfactory results in terms of time and memory as compared to the static algorithm.

1 Introduction

CLIQUE is a grid-based algorithm based on subspace clustering.This algorithm automatically identifies dense clusters in sub-spaces of a high-dimensional data space that allows better clustering than original space using Apriori principle.Moreover, it addresses each of the bottleneck of other clustering algorithms like scalability , minimal cluster descriptions etc.It produces identical results irrespective of the order in which the input records are presented and it does not presume canonical distribution for input data. Cluster descriptions are generated in the form of DNF expressions and strives to generate minimal descriptions for ease of comprehension.It also scales the data in a linear fashion. [1, 2]

2 Variants of CLIQUE

There are two aspects of the CLIQUE algorithm that can be improved.
The first one is the criterion for the subspace selection. The second is the size and resolution of the grid structure. The former is addressed by the **ENCLUS** algorithm by using entropy as subspace selection criterion.
The latter is addressed by the **MAFIA** algorithm by using adaptive grids for fast subspace clustering.

2.1 ENCLUS:Entropy-Based Approach

The algorithm ENCLUS (ENtropy-based CLUStering) is an adaptation of CLIQUE that uses a different, entropy-based criterion for subspace selection. Rather than using the fraction of total points in a subspace as a criterion to select subspaces, ENCLUS uses an entropy criteria and only those subspaces spanned by attributes A_1, ...,A_p with entropy H(A_1, ...,A_p) $<$ ω(a threshold) are selected for clustering. A low-entropy subspace corresponds to a more dense region of units. An analogous monotonicity condition or Apriori property also exists in terms of entropy. If a p dimensional subspace has low entropy, then so does any $(p-1)$ dimensional projections of this subspace:

$$H(A_1, ..., A_{p1}) = H(A_1, ..., A_p)H(A_p|A_1, ..., A_{p1}) < \omega.$$

2.2 MAFIA:Adaptive Grids in High Dimensions

MAFIA (Merging of Adaptive Finite IntervAls) proposed by Nagesh et al. is a descendant of CLIQUE. Instead of using a fixed size cell grid structure with an equal number of bins in each dimension, MAFIA constructs adaptive grids to improve subspace clustering and also uses parallelism on a shared-nothing architecture to handle massive data sets. MAFIA proposes an adaptive grid of bins in each dimension. Then using an Apriori algorithm, dense intervals are merged to create clusters in the higher dimensional space. The adaptive grid is created by partitioning each dimension independently based on the distribution (i.e., the histogram) observed in that dimension, merging intervals that have the same observed distribution, and pruning those intervals with low density. This pruning during the construction of the adaptive grid reduces the overall computation of the clustering step.

3 Base Static Algorithm

The Steps Involved in static clique algorithm are:

1. Start with 1-D space and discretize numerical intervals in each axis into grid.
2. Find dense regions(clusters) in each subspace and generate their minimal descriptions.
3. Use the dense regions to find promising candidates in 2-D space based on apriori principle (combine the 1-D dimensions and then form cluster).
4. We combine these dimensions in a level-wise manner in higher dimensional spaces(based on intersection).

3.1 Evaluation of Algorithm

The algorithm is evaluated on the following criterian :

- **Efficiency** Determine how the running time scales with:
  - Dimensionality of the data space.
  - Dimensionality of clusters.
  - Size of database.
- **Accuracy** Test if CLIQUE recovers known clusters in some subspaces of a high dimensional data space. Empirical evaluation shows that CLIQUE scales linearly with the size of input and has good scalability as the number of dimensions in the data or the highest dimension in which clusters are embedded is increased.
  Output:
  1. Number of clusters formed in the dataset.
  2. Number of outliers(noise) found in the dataset according to the threshold.

3.2 Computation time of the Algorithm

1. **Finding dense units(create_grid()):-** If a dense unit exists in k dimensions, then all of its projections in a subset of the k dimensions ,that is, **O(2^k)** different combinations, are also dense. The running time of our algorithm is therefore exponential in the highest dimensionality of any dense unit. It is known that the candidate generation procedure produces the minimal number of candidates that can guarantee that all dense units will be found. Let k be the highest dimensionality of any dense unit and m the number of the input points. The algorithm makes k passes over the database. It follows that the running time of our algorithm is **O(c^k + m*k)** for a constant c.
2. **Finding clusters(expand_cluster()):-** The number of dense units for a given subspace cannot be very large, because each dense unit must have selectivity greater than or equal to threshold . We give asymptotic running times in terms of dense unit accesses; the dense units are stored in a main memory data structure(container).The algorithm performs **Depth First Search(DFS)** to find neighbors of a block. For each dense unit visited, the algorithm checks its **2*k** neighbors to find connected units. If the total number of dense units in the subspace is n, the total number of data structure accesses is **2*k*n**.

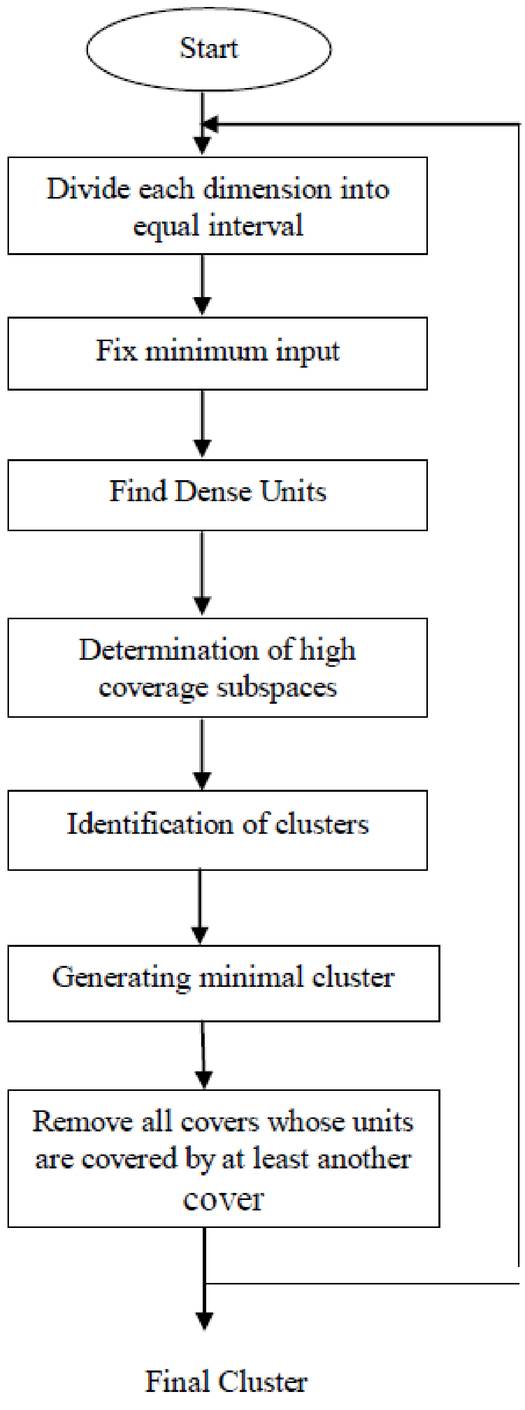
Start
Divide each dimension into equal interval
Fix minimum input
Find Dense Units
Determination of high coverage subspaces
Identification of clusters
Generating minimal cluster
Remove all covers whose units are covered by at least another cover
Final Cluster

Figure 1. Flowchart For CLIQUE Algorithm[3]

4 Proposed Approach

Modification from the static method :

- Saving the final clusters subspace dimension.
- Saving number of points in each unit hyper-rectangle in that dimension.

Algorithm 1: Incremental Version

```
Result: Clusters in the High Dimensional Subspace
Identifying points that are added or deleted from the dataset.
Identifying the unit to which it belongs.
while Traversing the datapoints do
    if If points are added to a unit hyper-rectangle: then
        Then increase the count of number of points in that unit hyper-rectangle.
        if unit is already a dense unit then
            exit
        end
    else
        if unit is identified as newly dense unit: then
            Find the nearest dense unit in first neigbourhood and if dense unit is found make new dense unit the part of same cluster as that of neighbour. If not then make it a new cluster.
        end
    end
    if If points are deleted from unit hyper-rectangle: then
        if unit is still a dense unit then
            exit
        end
    else
        Find the cluster details and apply DFS only on that cluster space area.
    end
end
```

4.1 Intuition behind proposed approach

- **Addition of points :-**When we add a point it will only increase the density so if it is not already a part of cluster we need to check if it can be added to a cluster or not. To get a nearest cluster we can just search in the nearest neighbourhood with max distance of 1 in that space and if we found a dense unit in that neighbourhood we can make the newly dense unit the part of same cluster as that of neighbour and it can also lead to the collision of two previously independent clusters. If no dense unit in neigbourhood is found then we can make our newly dense unit a independent cluster.
- **Deletion of points :-** When we delete a point it can decrease the density and already founded cluster can split. So we need to apply dfs for each nearest dense neighbour of the now non-dense unit. The previous cluster should be deleted and and newly found clustered should be added. This needs to be done only when there is more than one dense neighbour of now non- dense neighbour.

4.2 Comparison between static and incremental algorithm:

In **static algorithm**, if you want to find clusters with newly added or deleted points in the dataset, you have to run every step of the algorithm on the complete dataset i.e. units are formed again according to the new dataset and then clusters are formed with these newly formed units.
However, in **incremental version**, when any addition or deletion is done in the dataset, every step of the algorithm need not be performed as we utilise previous results i.e. count of points in a unit hyper rectangle and cluster description to identify new dense units if any. If there is an addition, only the nearest neighbours are looked at and processed as explained above. In case of deletion, DFS is performed only on the nearest dense neigbour (more than one) affected unit to find new clusters.

4.3 Contribution towards incremental strategy

We have implemented the incremental version for addition of new points to the dataset. We are storing number of points of a block, logical location, min corner, max corner and visited value for each block from the static algorithm in a text file. We are also storing the cluster information in a separate file for further use in incremental version. In incremental version,capture_points() method in the create_grid() function will be called for the newly added points only and the clusters will be formed accordingly using the information stored in the text file.

5 Dataset

1. **Abalone :-**A real-world dataset that contains the physical measurements of a snail called abalone. We are clustering them according to their measurements given.
   - Number of attributes :- 8
   - Number of instances :- 4177
2. **Target:-**A synthetic dataset for testing the algorithm.
   - Number of attributes :- 2
   - Number of instances :- 16

6 Experimental Results

In the following experimental results, we have divided the dataset on percentage basis i.e. 1%(2 points), 3%(3 points), 6%(5 points), 10%(7 points) and then performed various operation suchlike addition,deletion,addition & deletion. Following are the results observed on the Testing dataset: **Target**.The results shown for the target dataset are the average

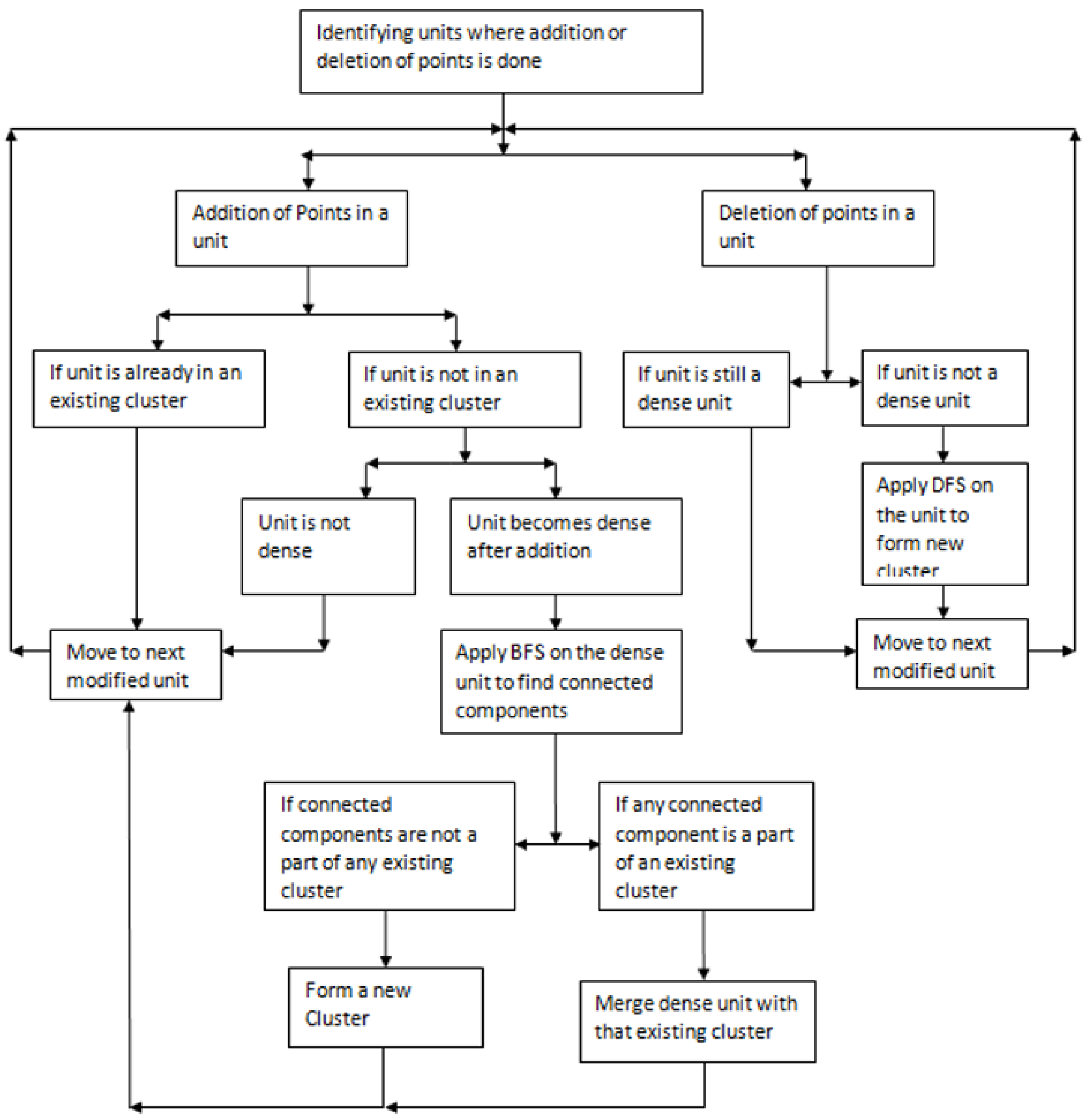
Identifying units where addition or deletion of points is done
Addition of Points in a unit
Deletion of points in a unit
If unit is already in an existing cluster
If unit is not in an existing cluster
If unit is still a dense unit
If unit is not a dense unit
Unit is not dense
Unit becomes dense after addition
Apply DFS on the unit to form new cluster
Move to next modified unit
Apply BFS on the dense unit to find connected components
Move to next modified unit
If connected components are not a part of any existing cluster
If any connected component is a part of an existing cluster
Form a new Cluster
Merge dense unit with that existing cluster

Figure 2. Flowchart For Incremental Version

results which were found on percentage basis.

Assumptions

1. Boundary points are not considered for deletion.
2. We are considering that none of the points are coming on the boundary of any cell block.

Running Time & Memory footprint of the algorithm

- In **Target** dataset

1. Number of intervals = 5
2. Threshold = 1

Table 1. **Dataset :Target**

Parameters	Static	Incremental
Time (microsec)	8775.01	7733.73
Memory(RAM used)(KB)	1005.15	1027.26

After testing the algorithm on the test dataset, we evaluated the same for the high dimanesional dataset.

- In **Abalone** dataset

1. Number of intervals =5
2. Threshold = 25

Table 2. **Dataset :Abalone(1 percent addition)**

Parameters	Static	Incremental
Time (microsec)	27143633	107810
Memory(RAM used)(KB)	149580	199464

As we can see through the above results there is improvement in running time of the algorithm.

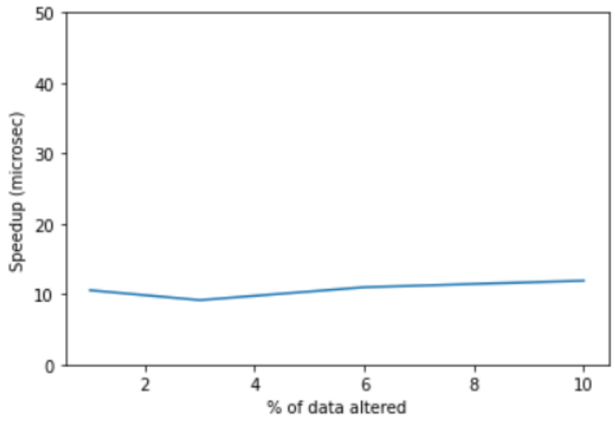

Figure 3. Time Consumption

7 Lesson learnt from the project

In this project, we learned about the concept of incremental algorithm and how we can adapt an existing algorithm to do the modifications in a dataset. We learnt how high dimensional spaces can be split into subspaces to speedup the clustering process. We understood what scenarios should be considered for addition and deletion of points in a dataset while converting static version into incremental version of the algorithm. Although our project focused on a particular clustering algorithm i.e. Clique, we understood the basic idea of how other algorithms like DBSCAN, MAFIA also performs clustering.

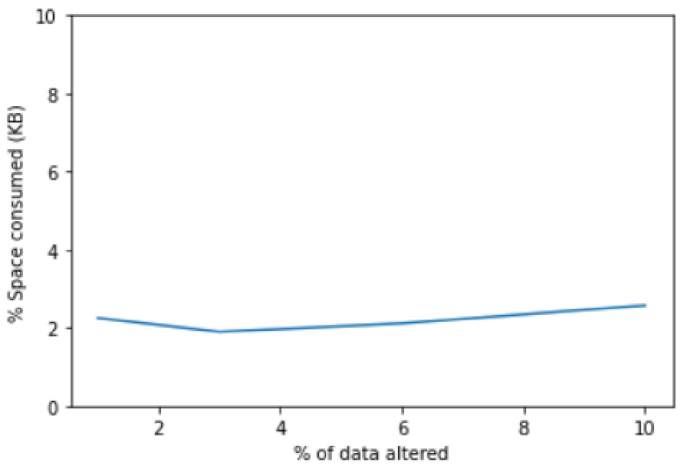

Figure 4. Space Consumption

8 Future Work

Our incremental version of clique algorithm does not focus on the diagonal neighbors of a block while forming a cluster, so in future the algorithm can be modified to adapt to consider the diagonal neighbors of the block. Algorithm works efficiently upto five dimensions but as the dimensions increase, it can effectively take longer time which can be improved in future versions of the algorithm and can be adapted to any type of dataset.

Table 3. **Links**

Type	URL
Base paper	`https://www.cs.cornell.edu/johannes/papers/1998/sigmod1998-clique.pdf`
Code of base paper	`https://github.com/annoviko/pyclustering`
Dataset: Target	`https://github.com/annoviko/pyclustering/tree/master/pyclustering/samples/samples/fcps`
Dataset: Abalone	`https://archive.ics.uci.edu/ml/datasets/Abalone`
Code of Incremental version	`https://github.com/VikashChauhan007/Data-Mining`
Max speedup	99%
Extra memory(Incremental version)	25%
Accuracy	80.76%
Accuracy Measure	It is measured by whether the cluster which are given by static algorithms contains the same set of points as that of Incremental Version.
Interested in improving the work?	No

Formula for accuracy

$$Measureofaccuracy = \frac{Numberofcorrectprediction}{TotalPredictions} * 100$$

References

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications," *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, June, 1998.

[2] A. Kaur and A. Datta, "A novel algorithm for fast and scalable subspace clustering of high-dimensional data. Journal of Big Data," in *International Conference on Something*, 2015.

[3] J. Yadav and D. Kumar, "Subspace Clustering Using CLIQUE:An Exploratory Study," in *IJARCET*, 2014.