Chapter 4

SAMCLIQ: A SAMpling based CLIQue Algorithm

We discussed the details of the first subspace clustering algorithm CLIQUE[1] in Section 2.2. We observed that, one of the major drawbacks of the algorithm is the repeated number of database passes required during step one to find the selectivity of the large number of candidate units that are generated. For very large databases, when the entire data cannot be loaded into the main memory at one time this step will require a tremendous amount of I/O to be done. Hence, if for example 10% of the data fits in the available main memory at a time then for one pass through the entire database, the data will have to be loaded in parts 10 times from the disk. And for \( k \) passes over the data, \( 10 \times k \) loads will be required.

*Can we improve the efficiency of the first step, to handle very large databases?* To address this problem, we developed an algorithm SAMCLIQ which uses a sampling based approach to find the dense units existing in the various subspaces of the data space. In this chapter, we first discuss in Section 4.1 the Use of sampling technique in data mining. In Section 4.2, we propose a sampling technique to get the sample of records from the original data space for finding the initial set of dense units. The details of the SAMCLIQ algorithm are presented in Section 4.3 and Section 4.4 reports the experimental results.
4.1 Use of Sampling in Data Mining

Sampling has played a very important role in data mining and has been mainly used to reduce the I/O activity required for knowledge discovery in large databases. In section 4.1.1, we explain the important role played by sampling in data mining. Section 4.1.2 explains some limitations of sampling and certain solutions to overcome them.

4.1.1 Role played by Sampling in Data mining

The application of sampling for mining association rules has been suggested in [21], and its effectiveness for mining association rules has been evaluated in [31]. It has been noted in [29] that samples of reasonable size provide good approximations for frequent sets. In [18], a general analysis on the relationship between the logical form of the discovered knowledge and the approximate sample sizes needed for discovering the knowledge has been studied. The role played by sampling in data mining has been well explained in [22] also. In the experimental evaluation carried out in [31], it has been shown that samples of reasonable size which fit in the main memory can be used with a reasonably high level of accuracy, to find the data patterns that exist in the database with high confidence.
4.1.2 Limitations of Sampling

To reduce the I/O activity, a random sample from the original database, small enough to be handled totally in main memory is drawn and the approximate regularities that exist in the original database are found out. These approximate results are then used to adjust parameters for a more complete knowledge discovery phase. Choosing sample sizes depending on the available main memory, approximate results can be obtained about the original database. However, we cannot be very sure that we have not missed out any data patterns that exist in the original database. And at the same time, if we do not include the right set of records in the sample we may get some patterns in the sample which actually do not exist in the original database.

Hence, in order to obtain the best results from the sample drawn it is important that we select a proper size for the sample and at the same time ensure that we select those records from the original database which help us in identifying in majority of the cases, all the patterns which exist in the original database. For this purpose we have developed a sampling technique for extracting a sample of data records from very high dimensional huge datasets, which is based on the AOSS method used for storing data. We discuss this method of sampling in Section 4.2.

In business and various other applications, where important decisions have to be taken based on the data patterns that exist in the databases, one cannot rely totally on the results obtained from sampling. Hence, as a tool for further analysis, the concept of negative border has been applied in many applications. The negative
border information has been used in [27], [13] and [23] to achieve efficiency in the incremental mining of association rules. In [23], Manilla and Toivonen have shown that the evaluation of the negative border units ensures that no frequent patterns are missed out. We have adopted the use of the negative border concept to ensure that we do not miss out any of the dense units, which were not present in the sampled records, but are actually found in the original database. More details about the negative border units have been discussed in section 4.3.

4.2 Proposed Sampling technique

In this section, we first discuss some criteria for a good sample under subsection 4.2.1 followed by subsection 4.2.2 which presents a brief discussion on the sampling method used in [29]. Section 4.2.3 discusses the AOSS based sampling technique.

4.2.1 Criteria for a good sample

The efficiency and the accuracy of the results obtained by using sampling, depends on the following two factors-

- *Sample size* – If the sample size selected is too small, compared to the size of the original database then there is a more chance of missing out the patterns found in the original data. And at the same time if the sample
size selected is very large, then we may not get any missed units but the actual purpose of sampling is lost. The size selected should be able to have a balance between the number of missed patterns generated and the extra time that we spent in processing the sample records.

- **Selection of good records** – The selection of a proper sample size is important, but choosing the right set of records for the sample is more important than this. Even if we choose a big sample size, but if most of the records selected are either outliers or noise points then we will fail to identify the correct patterns from the database.

Given a sample size $n$, we have designed a sampling algorithm which gets the best set of $n$ points to help in identifying all the possible patterns from the original database of size $N$ in majority of the cases. For this purpose, we assume that the original data has a very large number of attributes, and is very large in size such that the entire data does not fit in main memory at one time. The details of the sampling technique is explained in section 4.2.3.

### 4.2.2 Sampling for finding frequent sets

Till date, many algorithms have been designed for sampling but none of them address in specific, the issue of drawing a sample from a very high dimensional huge dataset. Most of them randomly pick up the points, without giving much importance to the quality of the points that are selected. A lot of the
algorithms have attempted to get the sample sizes for a required level of accuracy. In [29], Toivonen has used sampling for reducing the number of database passes required to find the frequent item sets to be used for finding the association rules from large databases. The performance study in [29] shows that after mining the sample, the sampling algorithm needs only one scan of the original database to find all frequent patterns. However, this algorithm does not focus on the selection of the points for the sample, but uses Chernoff bounds to determine the sample size required for a desired level of accuracy. This process of finding the sample size does not take into account the size $N$ of the original database, hence many times if the accuracy level required is very high it may give a large sample size. Besides the algorithm has not paid much attention to picking the right set of points for the sample, since they were not dealing with very high dimensional data sets.

4.2.3 AOSS based sampling technique

In our proposed sampling technique we have focused on the selection of the points for the sample from those, which contribute to the formation of the various 1-dimensional dense units. The various steps are as under-

1) Using the user-input value for $\xi$, form the various one dimensional candidate units by splitting the range of all attributes into $\xi$ intervals.
2) Using the AOSS attribute tables for all the attributes, the selectivity of all units is found out. The process of finding the selectivity for the candidate units of the various attributes can be carried out in parallel.

3) The 1-dimensional dense units are obtained by choosing those candidate units whose selectivity is larger than the user specified threshold value \( \tau \).

4) We retain only the record-id information of the 1-dimensional dense units. We call this set of record-ids the sample pool. Naturally this sample pool will be much smaller in size compared to the total data size. We select the points for the sample from this pool.

5) Sample selection –

The details of the sample selection are discussed after example 4.1.

**Example 4.1**

Consider the following transaction table 4.1 consisting of 10 records. Each record has 6 attributes namely A, B, C, D, E and F. The values of all these attributes, are in the range of 1 to 100.

<table>
<thead>
<tr>
<th>TID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1</td>
<td>21</td>
<td>41</td>
<td>0</td>
<td>4</td>
<td>18</td>
</tr>
<tr>
<td>T2</td>
<td>4</td>
<td>24</td>
<td>44</td>
<td>9</td>
<td>9</td>
<td>49</td>
</tr>
<tr>
<td>T3</td>
<td>6</td>
<td>26</td>
<td>46</td>
<td>45</td>
<td>23</td>
<td>83</td>
</tr>
<tr>
<td>T4</td>
<td>9</td>
<td>29</td>
<td>49</td>
<td>57</td>
<td>56</td>
<td>5</td>
</tr>
<tr>
<td>T5</td>
<td>2</td>
<td>8</td>
<td>25</td>
<td>58</td>
<td>78</td>
<td>30</td>
</tr>
<tr>
<td>T6</td>
<td>53</td>
<td>9</td>
<td>92</td>
<td>59</td>
<td>79</td>
<td>52</td>
</tr>
</tbody>
</table>
Consider a threshold value of 0.4 and the number of intervals equal to 10.

The various units will have the following range values:

Unit 1: 1-10,
Unit 2: 11-20,

Unit 10: 91-100

The various 1-dim candidate units formed are as under —

A1, A2, ... , A10,
B1, B2, ... , B10,

F1, F2, ... , F10.
After pass one through the transactions in the table 4.1, we get the following dense units —

<table>
<thead>
<tr>
<th>unit</th>
<th>freq. count</th>
<th>TID lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>5</td>
<td>T1, T2, T3, T4, T5</td>
</tr>
<tr>
<td>B1</td>
<td>4</td>
<td>T5, T6, T7, T8</td>
</tr>
<tr>
<td>B3</td>
<td>4</td>
<td>T1, T2, T3, T4</td>
</tr>
<tr>
<td>C5</td>
<td>4</td>
<td>T1, T2, T3, T4</td>
</tr>
<tr>
<td>D6</td>
<td>5</td>
<td>T4, T5, T6, T7, T8</td>
</tr>
<tr>
<td>E8</td>
<td>4</td>
<td>T5, T6, T7, T8</td>
</tr>
</tbody>
</table>

From the 1-dimensional dense units, the following 2-dimensional candidate units will be generated —

(A1 B1),
(A1 B3),
...
(A1 E8),
(B1 C5),
...
(B1 E8),
...
(D6 E8),
After second pass we get the following 2-dim dense units -

(A1 B3) : 4,
(A1 C5) : 4,
(B3 C5) : 4,
(B1 D6) : 4,
(B1 E8) : 4,
(D6 E8) : 4.

After third pass we get the following 3-dim dense units -

(A1 B3 C5) : 4
(B1 D6 E8) : 4

Given above are the various steps carried out in step one of the CLIQUE algorithm.

The Sample Selection procedure is as follows -

1) find all one-dimensional dense units and their tid-lists (record-ids)

2) group transactions (record-ids) based on number of such 1-dim dense units they are contained in and have these groups sorted in descending order of the record-id counts.

3) Choose a proportionate number \( f / t_f \cdot S \), of record-ids randomly from each group in the sorted order. \( f \) represents the number of dense-units, \( t_f \) the
count i.e the number of record-ids present in that group and $S$ the desired sample size.

Grouping of record-ids based on number of 1-dim dense units they are present in, for the data in example 4.1 above this will be as follows –

<table>
<thead>
<tr>
<th>No. of units</th>
<th>Record-ids</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>T1, T2, T3, T6, T7, T8</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>T4, T5</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

Assume sample size = 4. Randomly select any 4 record-ids from group with number of unit equal to 3. If number of record-ids in unit 3 is less than 4 then select from remaining units i.e 4 in this case. This process ensures that we get good set of records for the sample i.e records containing dense units.
4.3 Subspace Clustering Using Sampling

The first step of the CLIQUE[1] algorithm is quite complex for huge datasets having high dimensional subspace clusters. If $k$ is the highest dimensionality of any unit that is found than it will require as many database passes over the data as equal to the highest dimensionality of any dense unit in the data. Hence in order to reduce the number of database passes and the I/O required, we have developed the SAMCLIQ algorithm. The SAMCLIQ algorithm basically tries to improve the performance by using an efficient sampling technique for identifying the dense units in step one of CLIQUE. After selecting the sample using the method discussed in section 4.2.3, we find all the dense units in the sample using the method discussed in section 4.3.1. After getting the dense units from the sample, we use the concept of negative border units to make sure that we have not missed out on any units, which are present in the original data space.

The Negative Border $N$, consists of all the candidate units generated in the level-wise algorithm that were not dense units. In other words if $C$ is the set of all the candidate units generated, $D$ is the set of dense units then $C = D \cup N$. After obtaining the results using sampling, we want to make sure that we have not missed out any units, which are dense in the original database but were not detected in the sample. Obviously the subsets of all such likely missed units will be found in the negative border $N$ of the sample.
4.3.1 Algorithm for identification of dense units

The Algorithm we propose requires an initial pass, for selecting the sample during which it generates the 1-dimensional dense units also. After selecting the sample we apply the level wise algorithm used in step one of CLIQUE to get all the candidate and dense units present in the sample. Then a first pass over the original database (O.D) is carried out to find if any units are missed out, by using sampling. If any units are missed then an additional pass is made over the O.D. This work was carried out prior to the development of the AOSS method. Hence, in this chapter we have not used it as such for the main algorithm. The details of the algorithms are as under

Sampling for identification of dense units – The accuracy of the results obtained using sampling, to a large extent depends on the size of the sample and the method used to select the sample points from the database. Since we are considering very large databases and we want our sample to fit in main memory, we choose sample size s such that it is neither too large and nor too low by using the technique discussed in section 4.2.3. We know that, with increasing sample sizes the probability of finding the dense units identified in the sample, in the original database also are high and thereby the possibility of occurrence of false dense units and of missed units are almost negligible. Hence, an extra pass over the database will not be required, but if the sample size is too large, then the time taken to process the sample is very large compared to the gain in performance achieved by reducing
the number of passes. In place of CLIQUE, other subspace clustering algorithms like MAFIA [16] can also be used. Depending on the quality of sample selected, there are three cases that we can encounter -

a) There may be some units, which were dense in the sample but are not dense in the original database. In such cases we have unnecessarily counted such units, we will call such units as false dense units. These false dense units get discarded after the first pass over the original database and do not affect the accuracy of the final results obtained for the original database.

b) There may be some units, which were not dense in the sample but are dense in the original database. In such a case, we say that there has been a miss i.e., we have missed to capture these units and some higher-level units of these in the sample. There are two types of misses that we may come across first type is where we fail to capture some dense units in some subspaces and second wherein some subspaces containing dense units were fully missed. Whenever there are such missed units say $M$, then some higher level candidate units say $C_1$ generated using $M$ may be dense in the original database. But this set $C_1$ would not be generated by the sample, hence there is a need to generate higher level candidate units of such missed units and evaluate them i.e find their counts in the original database by doing an additional pass over the database.

c) The units which were dense in the sample are dense in the original database also and vice-versa. This is an ideal case and gives the best performance if the sample size is selected properly i.e it is not too large, but at the same
time very closely resembles the original database. In this case the results will be obtained by doing a single pass over the original database, besides the initial pass required to draw a sample and find 1-dimensional candidate and dense units for the original database.

From above we observe that it is case (a) and case (b) that needs to be handled properly. To handle case (b) one method that is discussed in [29] is to lower the density threshold value, while generating the candidate units for the sample. This will definitely reduce the chances of a miss, but will lead to an increase in the number of false dense units. The aim to avoid the misses, is to achieve the results in just one pass. If there are missed units, then two complete passes will be required over the original database. Another method to reduce the number of passes to less then two complete passes is to adopt the technique used in [7]. Instead of waiting for the end of the first pass to find the missed units, we check for missed units after every M records have been processed and generate the higher level candidate units for such missed units and start counting the occurrence of these units from that point onwards. If all the missed units were detected, after x number of transactions were processed during the first pass, then we will need one complete pass and scan only the un-scanned x transactions during the second pass for the missed units in C1. We will need two complete passes only when we have found missed units towards the end of the first pass. This will normally happen if the data is very correlated. The value for M should be selected carefully, in such a way that there is not much of processing overhead.
Algorithm for Generating the dense units in the Sample drawn:

Inputs:

- The Sample points of size s from the original database (O.D.),
- number of attributes(dim),
- density threshold \( \tau \),
- set \( C[1] \) of 1-dimensional candidate units
- set \( D[1] \) of 1-dimensional dense units obtained from the sample.

Outputs:

- set \( C \) of candidate units in the sample,
- set \( D \) of dense units obtained from the sample data records
- the 1-dimensional dense units of O.D.

Processing:

1. Use \( D[1] \) to find \( C[2] \) set of 2-dimensional candidate units; // this avoids the chances of a 1-dimensional missed unit
2. While more candidates are generated

\{
    find selectivity of \( C[k] \) in the sample; // for \( k \geq 2 \) and \( \leq \text{dim} \)
    find \( D[k] = \text{dense units in sample from } C[k] \);
    generate \( C[k+1] \) from \( D[k] \); // the candidate generation procedure used in CLIQUE is used.
\}
CHAPTER 4. SAMCLIQ: A SAMPLING BASED CLIQUE ALGORITHM

++k;

}

Algorithm for the first pass  // to find if any missed units are found

Inputs:

- set C and D obtained from the sample,
- the Original Database O.D,
- the threshold value $\tau$.

Outputs:

- set D1 set of dense units in the O.D
- set C1 set of missed units.

Processing:

1. Num_parts = N/BUFFSIZE;
2. Initialize counts of all units in C to 0;
3. for (n = 1; n <= Num_parts; n++)
   
   { read BUTFFSIZE records into main memory buffer[BUFFFSIZE];
     update counts of units in C using buffer[BUFFFSIZE];
   }
4. find D1 = set of dense units in O.D ; // those units from C whose count is $\geq N^* \tau$
5. find $M$ - missed units by comparing $D$ and $D_1$;

6. if $M$ is empty
   
   goto step 9; //stop

else

   find $C_1 = \text{set of all candidate units formed from } M \text{ and } D \text{ units ;}$

// $C_1$ is the candidate units missed in the sample, which may be dense in the O.D

7. Do a second pass through the O.D; // required only if there are missed units.

8. The set $D_1$ consists of all the dense units in the Original Database (O.D).

9. Stop

This forms the input to the second step of the subspace clustering algorithm.

Algorithm for the second pass

Inputs:

- set $C_1$ // the set of all candidate units formed from $M$ and $D \text{ units obtained from first pass,}$

- O.D,

- the threshold value $\tau$.

Output:

- the final set $D_1$ of O.D // all dense units of O.D
CHAPTER 4. SAMCLIQ: A SAMPLING BASED CLIQUE ALGORITHM

Processing:

1. Find the counts of all units in set $C_1$ in O.D.
2. Find additional dense units obtained from $C_1$ and add to set $D_1$;
3. Stop.

4.4 Experimental Results

In this section we present an empirical evaluation of the above algorithm, which we call SAMCLIQ (SAMpling based CLIQue) algorithm using synthetic datasets. The goal of the experiments was to compare the performance of step one of CLIQUE with the step one of SAMCLIQ. The MDL pruning used in step one of CLIQUE is not used in our implementation of CLIQUE. We compared the performance by varying the size of the database, dimension of the data space and dimension of the Clusters. The experiments were run on a 800 MHz Intel Pentium III processor running Linux.

4.4.1 Synthetic data generation

The synthetic data generation method described in [1] has been used for the data generation. The data generator takes as input the number of records to be generated, the number of attributes and the range of values for each attribute. The range of values was set to $[0,100]$ for all attributes. The clusters are hyper-rectangles.
in a subset of dimensions such that the average density of points inside the hyper-rectangle is much larger than the average density in the subspace. The cluster description details provided by the user include the number of clusters, the maximum dimensionality of the clusters, and the cluster descriptions which specify the subspaces of each hyper-rectangle and the range of each attribute in the subspace. The attribute values for a data point assigned to a cluster are generated as follows. For those attributes that define the subspace in which the cluster is embedded, the value is drawn independently at random from the uniform distribution within the range of the hyper-rectangle. For the remaining attributes, the value is drawn independently at random from the uniform distribution over the entire range of the attribute. We add 90% of the specified number of points equally among the specified clusters, and the remaining 10% points are added as random noise. Values for all the attributes of these points are drawn independently at random from the uniform distribution over the entire range of the attribute.

4.4.2 Synthetic data results

We studied the performance of CLIQUE v/s SAMCLIQ algorithm, by varying the following parameters: the database size, the dimension of the data space and the dimension of the clusters. The values for $\xi$ and $\tau$, were set to 10 and 0.15 respectively.
**Database size:** Figure 4.1 shows the results of the experiments carried out by varying the database size from 5,00,000 records to 20,00,000 records. The sample size was selected as 1% of the database size and the main memory buffer size was taken equal to the space required to load 50,000 records. The data space had 50 dimensions. We found that the difference between the time taken by CLIQUE and SAMCLIQ increases significantly with the increase in the database sizes.

**Database size:** Figure 4.1 shows the results of the experiments carried out by varying the database size from 5,00,000 records to 20,00,000 records. The sample size was selected as 1% of the database size and the main memory buffer size was taken equal to the space required to load 50,000 records. The data space had 50 dimensions. We found that the difference between the time taken by CLIQUE and SAMCLIQ increases significantly with the increase in the database sizes.

[Figure 4.1: Scalability with the number of records.]

[Figure 4.2: Scalability with the dimension of the data space.]
Dimensionality of the data space: Figure 4.2 shows the scalability as the dimensionality of the dataspace is increased from 25 to 100. The experiments were carried out with a database containing 10,000,000 records. There were 5 clusters each in a different 7 dimensional subspace. The sample size selected for SAMCLIQ was 5% of the database size.

![Figure 4.3: Scalability with the dimensionality of the clusters.](image)

Dimensionality of the clusters: Figure 4.3 shows the scalability as the highest dimensionality of the clusters embedded in the different subspaces is increased from 3 to 9. Again the database size was selected to be equal to 10,000,000 records and the sample size was taken equal to 5% of the database size.

4.5 Summary

We have used the CLIQUE algorithm as the base on which the SAMCLIQ algorithm has been developed. But this can also be used in other subspace clustering
algorithms like MAFIA to further boost the performance. Our algorithm can easily be implemented as a parallel algorithm. After the first pass through the data base and the generation of candidate units using the sample, the data base can be split into \( n \) parts and the counts of all the candidate units can be computed in parallel in all the subparts. Then the counts in all \( n \) parts can be summed up and if there are missed units then again the counts for the missed units can be found in parallel during the second pass in SAMCLIQ.

We compared the performance of SAMCLIQ with CLIQUE by varying the database size, the dimension of the data space, and the dimensionality of the clusters. We found that there is a significant gain in performance when we use SAMCLIQ for large databases and higher dimensional data spaces. As we vary the dimensionality of the clusters, also the performance of SAMCLIQ is much better compared to CLIQUE but for very high dimensional clusters, the performance of SAMCLIQ also suffers because of very large number of candidate units produced. Hence, there is a need to use a different technique to find the high dimensional dense units in such cases.

The different techniques that can be used for this purpose are to make use of the concepts of g-closed itemsets, use FP-tree for generating dense units, or use of maximal frequent itemsets for finding the subspaces. The performance can be significantly improved by using the AOSS method for storing the high dimensional huge datasets, which gives the freedom to access only those attributes or records, which are needed during the process of finding the selectivity of the various candidate units.