# Homework 3

## CS6720 : Data Mining

CS15M015 : Hasit Bhatt

Indian Institute Of Technology, Madras

## 1 OPTICS

### 1.1 Reachability Plot for $\epsilon = 10, minPoints = 10$

Implementation of OPTICS was ran on the dataset of 200000 points in 2D plane. The reachability plot with $\epsilon = 10, minPoints = 10$ was observed.

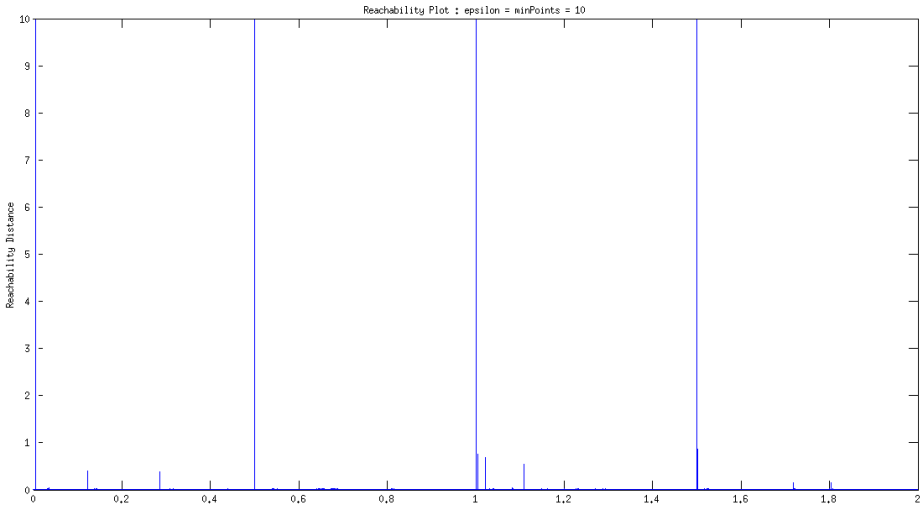



Fig. 1: Reachability Plot for $\epsilon = 10, minPoints = 10$

From the plot, we can deduct, that the points should have 4 clusters, as , only 4 major peaks are visible, and, all the other points are in the neighbourhood of other points in the cluster.

### 1.2 Clusters from reachability plot

For the optimum value of $\alpha$ and $\epsilon$ , the number of clusters were found to be 4. And, all of them are assigned to one of the clusters for value of $t = 0.1 and \epsilon = 2$.

Analysis:

With various parameters for t and $\epsilon$, the clusters were extracted. From the experiments, changing the value of $\epsilon$ does not affect much on the result of the clusters as long as they are not too small.

The dents in the dataset are in a way self-explaining the structure of points. And, we can see, the number of clusters should be 4.

But, as clustering is not a unsupervised, and, some additional information is needed, the data points were plotted on 2-dimensional surface.

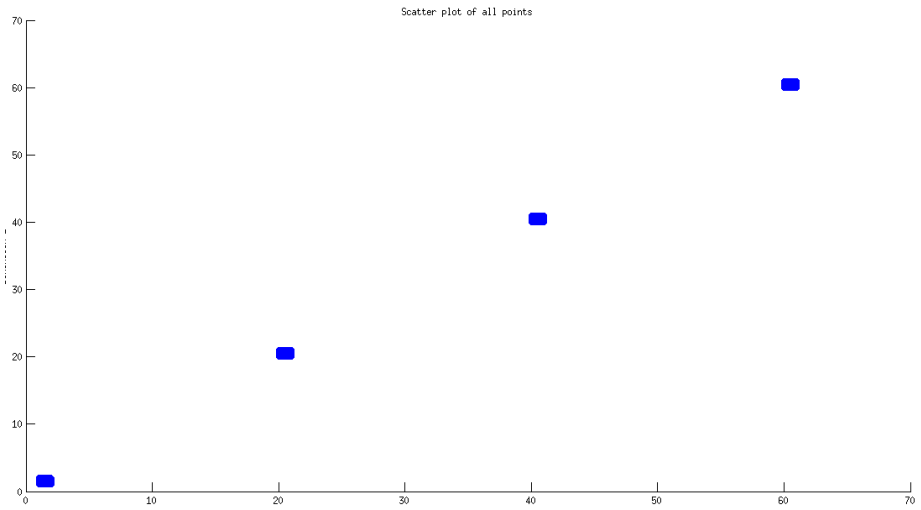



Fig. 2: Scatter plot of points on 2D plane

From the cluster, it was evident that, there are four clusters at almost equal distances.

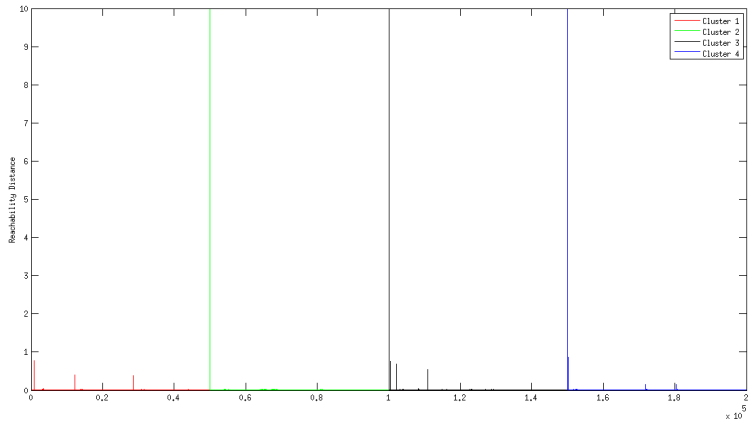
Some of the major observations taken were:

- The value of $\epsilon$ should not be too small, because, keeping too small value like 0.01 will make too many points unreachable, and therefore will not be able to extract the clusters properly.
- Other than keeping $\epsilon$ high enough, the change in $\epsilon$ does not affect much on the algorithm. e.g. taking $\epsilon$ as 100 will also give the same set of clusters given the value of $t$ (upward and downward slope) is chosen carefully.
- The choice of $t$ is a crucial part for correct cluster extractions. This is due to the fact that changing the value of $t$ will affect the number of clusters directly. e.g. if the value of t is too small then, even for small rise in reachability may label those points as a different cluster altogether.
- And, similarly, the points not belonging to any cluster ( with high reachability distance ) can also be classified as clusters if the value of t is too small.

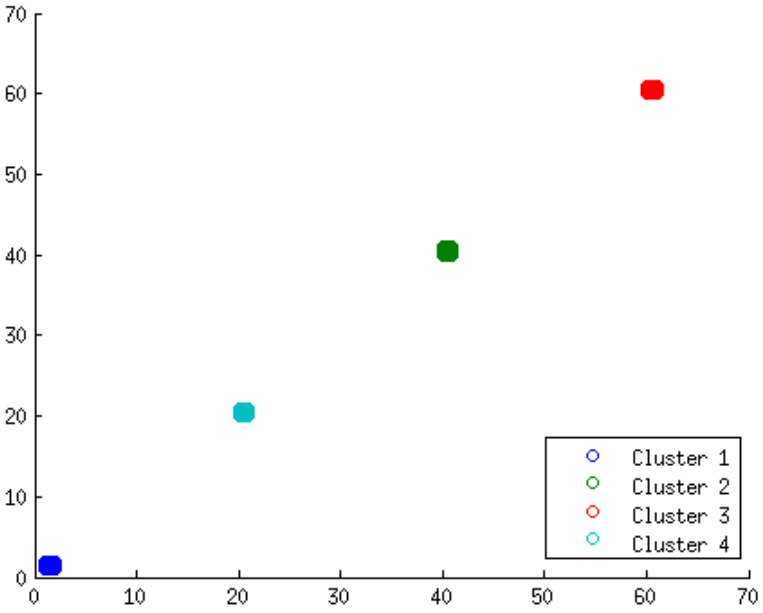
– Here points are too close to each other, so changing minPts does not affect much for this particular dataset.

Therefore, after pruning the ranges of parameters, the candidate range the parameters can belong to was decided. And out of feasible different parameters, suitable parameters $t = 0.1$ and $epsilon = 2$ were chosen, as these parameters were stable in nature (minor changes in any of these parameter does not change the cluster results).

Using these parameters, following clusters were observed.




(a) Reachability Plot




(b) Clusters

Fig. 3: Clusters using $t = 0.1$ and $epsilon = 2$

### 1.3 Difference in OPTICS and its Range Tree implementation with variable number of points

Experiment:

Range Tree implementation was done in Java, and neighbours were found using the range tree instead of linearly going through all the datasets. And, the time required for the same were plotted.

The dots were connected using spline function ( Matlab ) to see the trend with increase in the number of data points.

Plot:

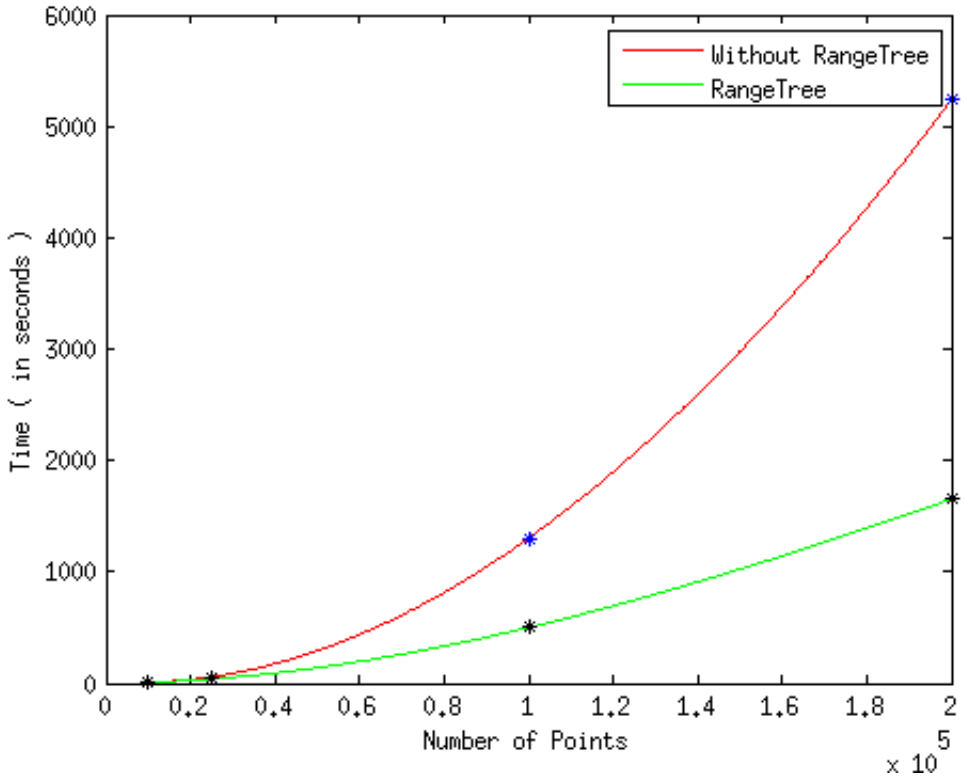



Fig. 4: Time : RangeTree vs without RangeTree

Analysis:

As we can see, for less number of data points, the values are quite similar for both RangeTree and non-RangeTree implementation. e.g. for 10000 data points both implementation takes around 8s. Even for 25000 data points, the difference is comparatively less. but, with increase in data, the different becomes visible. And, for all the datapoints, time taken is almost halved than the non-RangeTree implementation.

| Number Of Points | RangeTree | Non-RangeTree |
|---|---|---|
| 10000 | 7.841s | 8.013s |
| 25000 | 42.186s | 49s |
| 100000 | 11m37s | 21m04s |
| 200000 | 29m12s | 77m54s |

Table 1: Time comparison with change in number of points

Range Tree implementation is output dependent. It is supposed to have complexity of $O(log^{d-1}+k)$ for d dimensional data by trading off space requirement. it needs O(n logn) storage unlike O(n) space complexity of traditional implementation.

Theoretically, RangeTree implementation should perform atleast 4 times faster because, the points are divided into 4 clusters, and therefore, the only 25% of the data points should be returned by the query getNeighbours. But, it is not the case with the data. This can be explained by the fact that, range query is done in range in shape of square, from $P_d - \epsilon$ to $P_d + \epsilon$ in each dimension. Here $P_d$ denotes the value of dimension d for point P.

Therefore, the points are copied in an Array list, and validated again to remove the points who are not at distance less than $\epsilon$ from the point. And, this is a clear overhead, and therefore, the RangeTree implementation does not work as fast as it should be.

But, still it scales way better than the implementation without RangeTree.

### 1.4 Difference in OPTICS and its Range Tree implementation for different values of $\epsilon$

Experiment:

Similar to the previous experiment, OPTICS with its Range-Tree implementation was compared with its simpler version for different values of $\epsilon$

Plot:

Running time plot for the different value of $\epsilon$ was found as below.

| $\epsilon$ | RangeTree | Non-RangeTree |
|---|---|---|
| 5 | 29m17s | 78m14s |
| 10 | 29m12s | 77m54s |
| 25 | 40m37s | 78m05s |
| 50 | 70m26s | 77m43s |
| 100 | 83m14s | 77m04s |

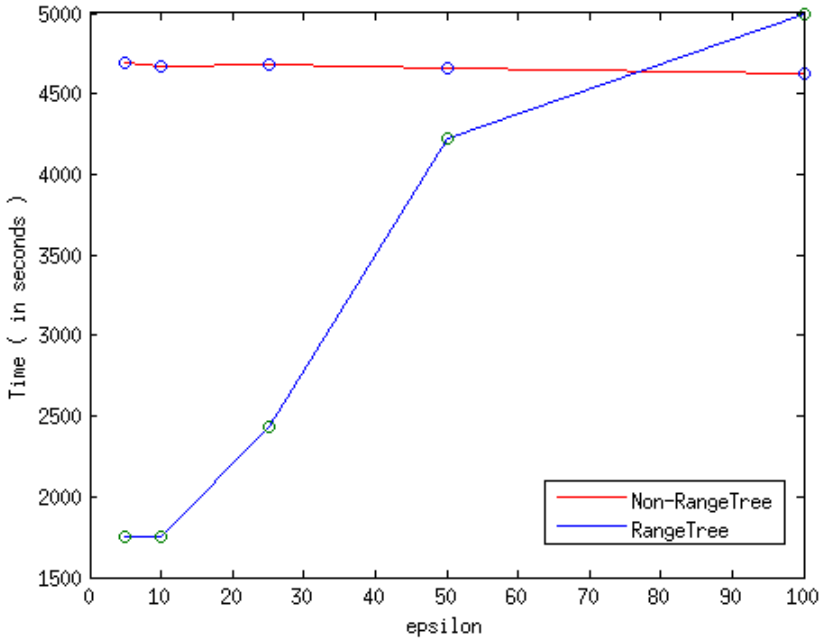



Fig. 5: Time comparison for RangeTree and non-RangeTree implementation with varying epsilon

Analysis:

Here, we can observe that, with the change of epsilon, the time for the implementation which does not use RangeTree remains almost the same, whereas, the RangeTree implementation changes with value of epsilon.

To understand why this happens, we must go through the structure of dataset. As we can see, clusters are separated from others. So, for the small epsilon, only the points belonging to the same region will be fetched, and as RangeTree implementation's time complexity depends on the number of points in that range, it takes less amount of time.

But with increase in the value of $\epsilon$, more data points come into the range, and therefore time required increases. And with increase in epsilon, the points which are not in the range ( which belongs in corner of square ) will also be fetched, because, range given in range query is square in nature and not circle.

Therefore, with $\epsilon$ as 100, the points fetched which are not in range keeps on increasing, and therefore it takes even more time than the conventional implementation.

**Note: The implementation of the algorithm can be found under Code folder. ( It contains both implementations with and without RangeTree)**.

## 2 Angle between pairs of points in High dimension

### 2.1 Experiment

For the experiment, for each point of dimension d, values were generated randomly between -1 to 1. For each pair of points, the cosine similarity $sim(A, B)$ were calculated and using inverse cosine function, the angle between the points were found.

The histogram for each points were plotted ans analysed. The function for angle can be written as below.

$$\theta = cos^{-1}(sim(A, B)) \tag{1}$$

$$sim(A, B) = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{d} A_i B_i}{\sqrt{\sum_{i=1}^{d} A_i^2} \sqrt{\sum_{i=1}^{d} B_i^2}} \tag{2}$$

where, $A$ and $B$ are points of dimension $d$.

### 2.2 Results

For different values of dimensions, the histograms were observed as observed in Fig. 6. All the histogram follows the normal distribution. And, with increase in dimensions, the variance keeps on decreasing, and, the histogram keeps on becoming dense around mean ( 90 degrees ).

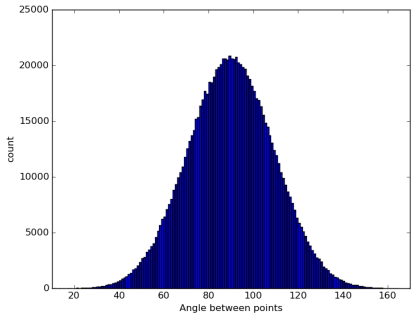
Therefore, with the increase in number of dimensions in space, more pairs having angle nearly 90 can be observed, and, from the trend it can be inferred that, for very high dimensional space, the angle between a pair of any two points will tend to 90 degree, and hence, the cosine similarity would tend to 0.

### 2.3 Explanation

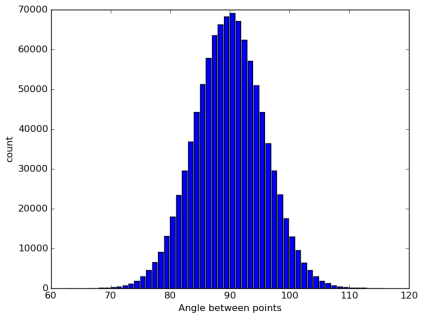
Cosine similarity between a pair of points (A,B) is defined as Eq. 2. If the equation becomes zero, it is evident, that the angle between them would become 0.

Now, by the law of large number, if the number of samples taken are enough, the expectation of variable tends to the mean ( 0 in this case ). So, $E(A)$ and $E(B)$ will tend to mean $\mu = 0$. But, the same can not be said for $E(A^2)$ and $E(B^2)$.

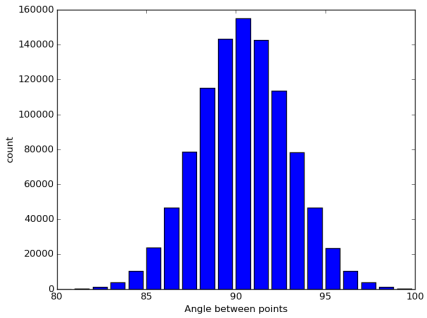
In the similarity function denominator will keep on increasing, because, expectation of $E(A^2)$ and $E(B^2)$ will just keep on increasing with dimension $d$. And the numerator ( dot product of two points ) will always range from -d to d.



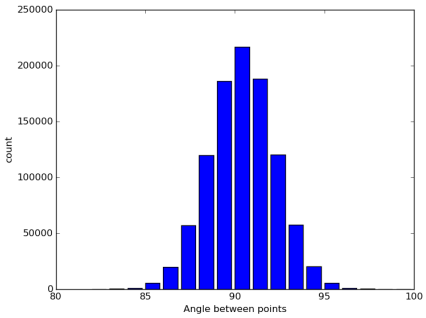

(a) for dimensions d=10



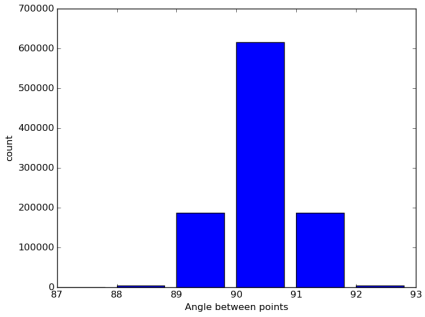

(b) for dimensions d=100




(c) for dimensions d=500




(d) for dimensions d=1000




(e) for dimensions d=10000

Fig. 6: Histogram plots for different dimensions (n=1000 points).

And, the dot product will never increase faster than the denominator. Therefore, when going to higher dimensions, the probability of cosine similarity among any two points keeps decreasing (as a whole). And, therefore by the eq. 1, a pair having angle 90 just keeps on increasing.

Therefore, when going into higher dimensions, the angle histogram is populated around 90 degrees.

The same can be observed in real life scenario, where few dimensions are not able to distinguish two entities, we introduce a new attribute. Here, new attribute just decreases the similarity among entities, which can be viewed as a higher dimensional representation decreases the similarity as a whole. And, if similarity tends to 0, the angle between points tends to 90.

This property of high dimensional data has been used in many techniques like BUBBLE, where, assuming the vectors orthogonal makes computations far easy and faster.

## 3 Markov Clustering Algorithm

Markov Clustering Algorithm uses the convergence proof to find the clusters ( aka communities ) for the given dataset. The clustering quality depends on two factors, the type of graph and inflation factor

If the graph is constructed in such a manner, that there is no convergence than, it is very hard to extract clusters from it.

Gephi tool was not able to handle the large dataset provided, therefore, mcl software from micans.org is used for this problem.

Python script for finding the community size can be found under Code folder.

After finding the communities, matching how much is it similar to the actual ground truth communities is a time consuming task, because, we need to check whether each of the communities have some common community.

It is $O(n^4)$ task, because for each inferred community, you need to check each ground-truth communities, and check whether all the elements are there or not.

Using set structure we can find the intersection between ground truth communities and our inferred communities to check which model works the best.

The histograms for the community size are as below.

### 3.1 Database and Logic Programming Dataset

Initial chaos: 201.54
Largest Community (Ground Truth): 7556
Smallest Community (Ground Truth): 6
Largest Top Community (Ground Truth): 7556
Smallest Top Community (Ground Truth): 6

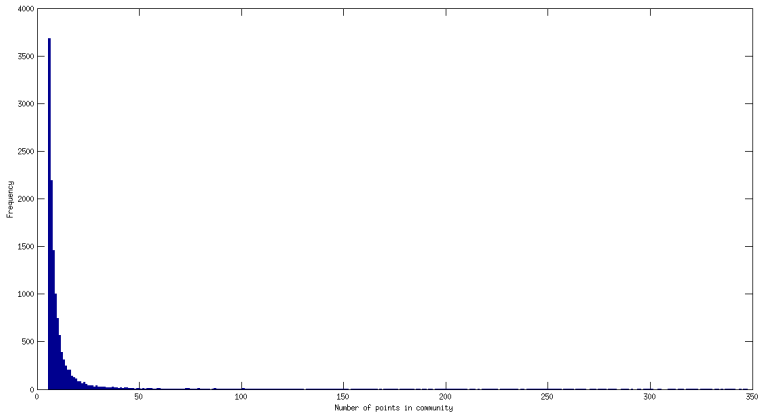



Fig. 7: DBLP Community

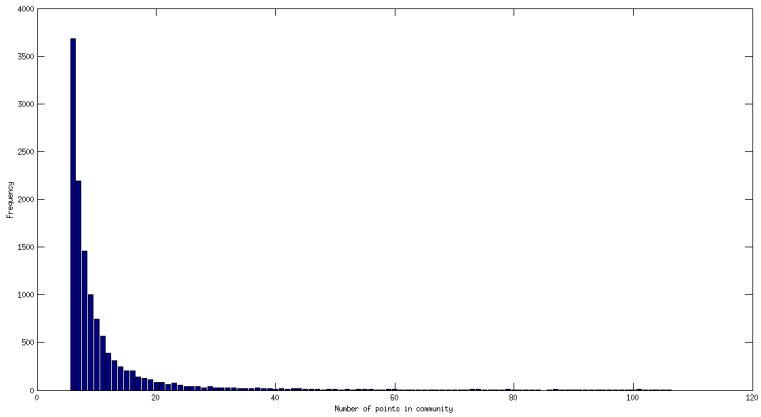



Fig. 8: DBLP Top Community

– Inflation value : 1.4
  51 iterations, 14830 clusters

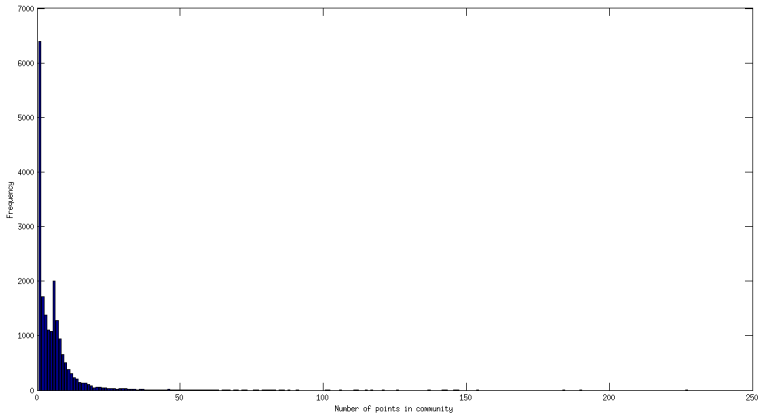



Fig. 9: Common communities (intersection) with top communities : i = 1.4

– Inflation value : 2
  26 iterations, 37966 clusters

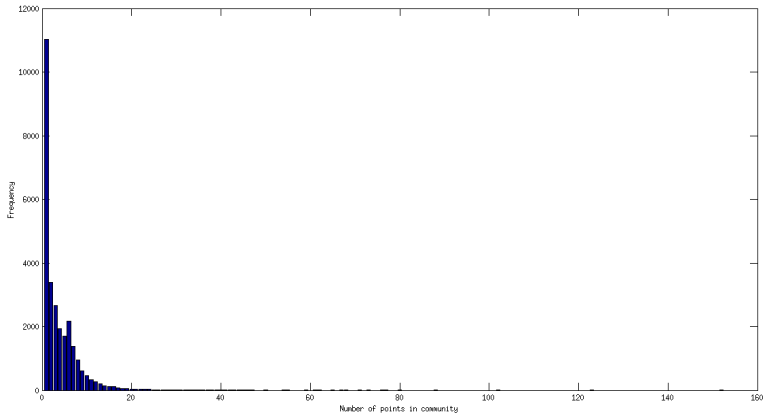



Fig. 10: Common communities (intersection) with top communities : i = 2

– Inflation value : 4
  15 iterations, 67333 clusters

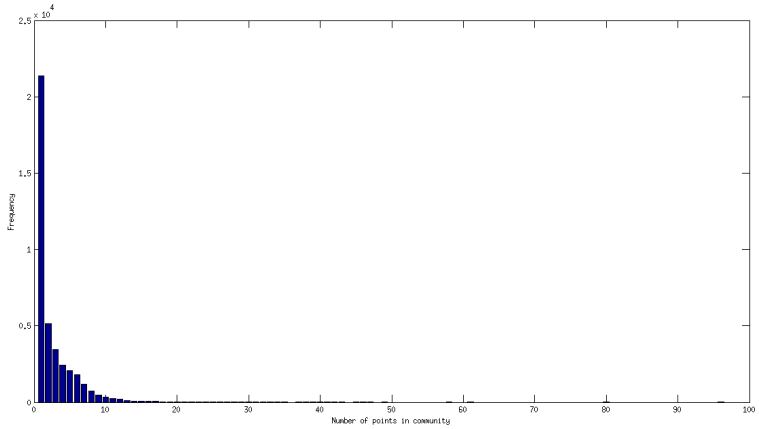



Fig. 11: Common communities (intersection) with top communities : i = 4

– Inflation value : 6
  13 iterations, 76173 clusters

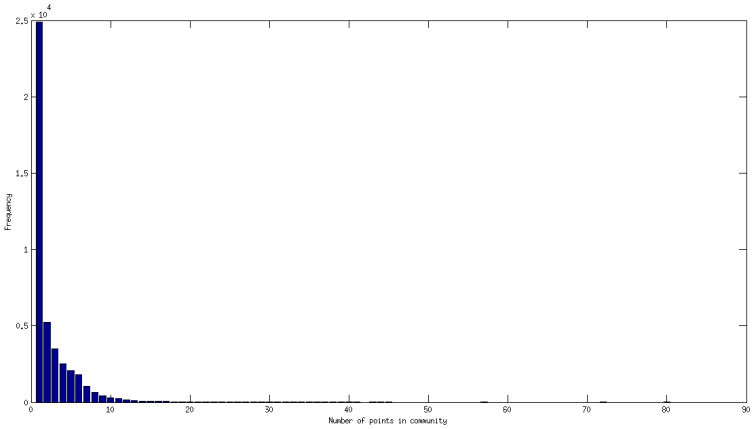



Fig. 12: Common communities (intersection) with top communities : i = 6

– Inflation value : 30
  Not possible to converge, becomes 0.

### 3.2 Amazon Dataset

For Amazon Dataset, the initial chaos value is 238.64.
Largest Community (Ground Truth): 53551
Smallest Community (Ground Truth): 3
Largest Top Community (Ground Truth): 328
Smallest Top Community (Ground Truth): 3

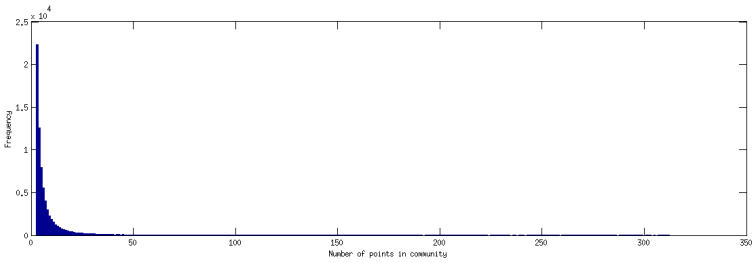



Fig. 13: Amazon Community

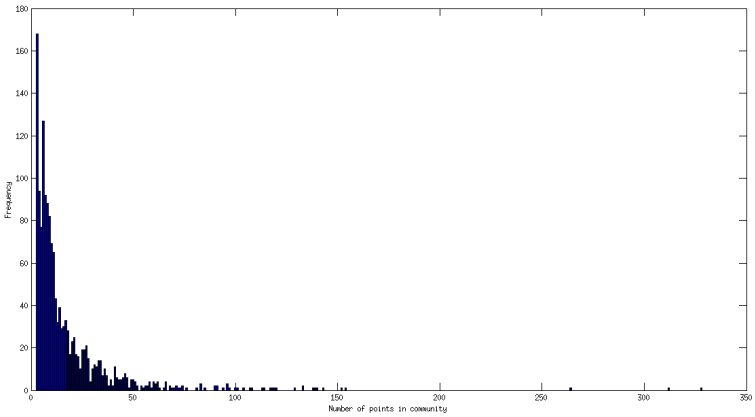



Fig. 14: Amazon Top Community

– Inflation value : 2
  26 iterations, 46557 clusters

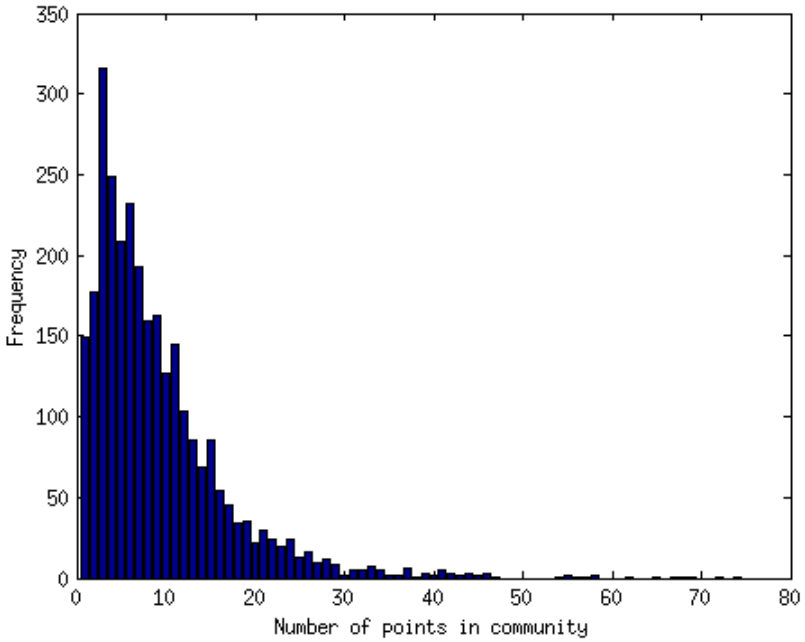



Fig. 15: Common communities (intersection) with top communities : i = 2

– Inflation value : 4
  17 iterations, 120588 cluster

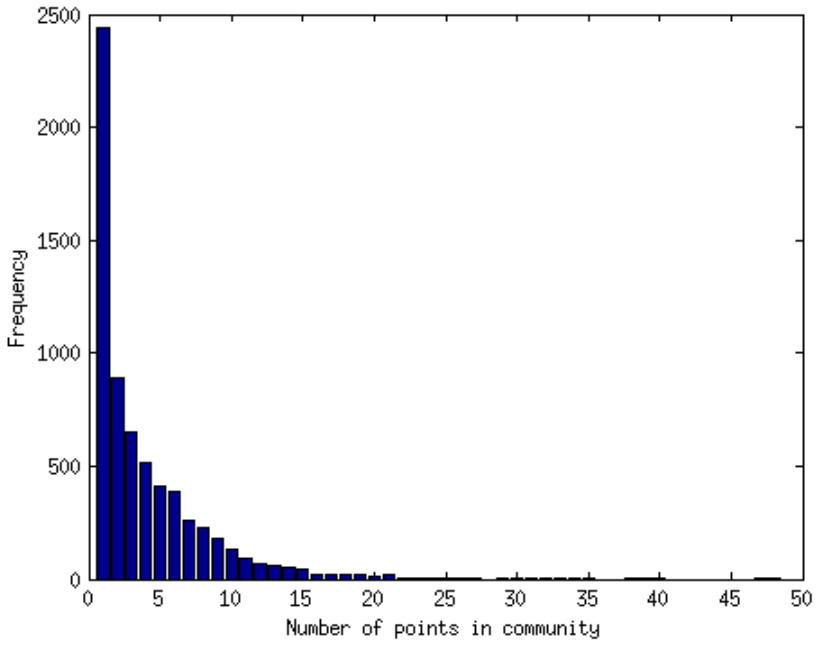



Fig. 16: Common communities (intersection) with top communities : i = 4

– Inflation value : 6
  15 iterations, 159258 clusters

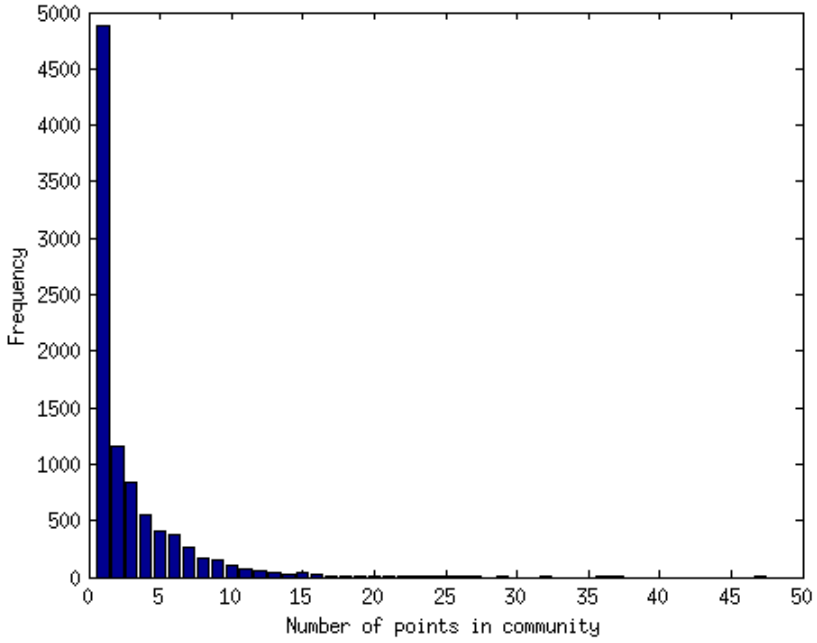



Fig. 17: Common communities (intersection) with top communities : i = 6

– Inflation value : 30
  30: 8 iterations, 210145 clusters

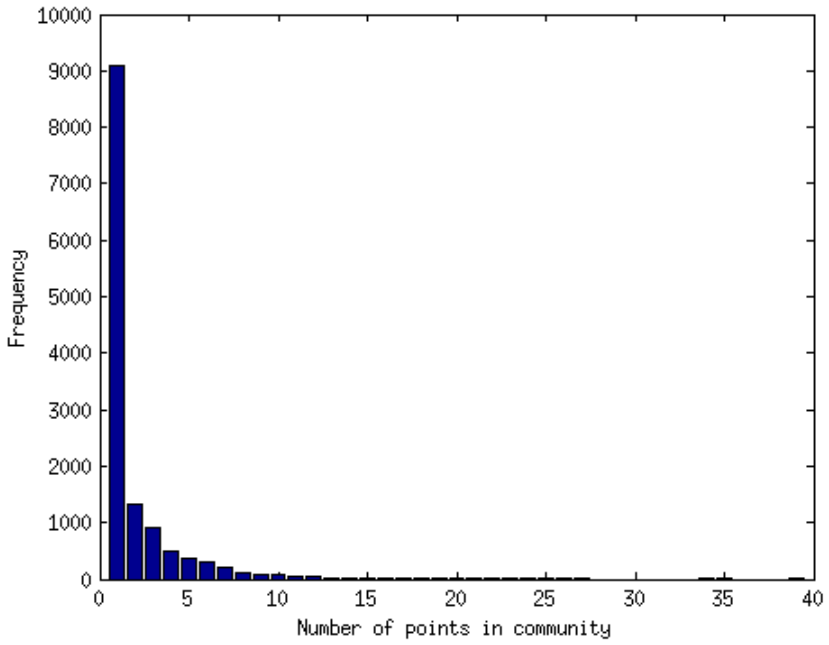



Fig. 18: Common communities (intersection) with top communities : i = 30

Here, for both the datasets, from graph it can be observed that top community and all communities behave in similar way as per the histogram. So, as we can see, for values 4 and 6, the number of singleton community increases, which does not signify anything.

For value inflation value 2, we get shape similar to top ground truth communities, slightly shifted left by 2. That means, the community retrieved at 2 is similar to real communities. So, we may find some hidden communities in the result set, which was not visible in top communities.

Histogram also shows that with less inflation value, we need more iterations, and the convergence is slow. Moreover, the communities contain many entities, which makes the community weak.

So for both the cases inflation value 2 gives best result, as it is a point of transition. because, as can be viewed in the histograms, decreasing inflation create bigger communities, but they are weak, and on the other hand, the higher inflation makes weak communities, but they are too small, e.g. for i = 6 or 30, because, for those, histogram is concentrated at 1.

Clusters extracted with parameter of Inflation as 2 keeps the community strength moderate, and is most similar to the actual top ground truth community in shape.

**Note: For processing the files to find intersection of communities and Python script was used which can be found under Code folder.**