Graph Clustering Algorithms

Instructor:
Dr. Anand Mishra

Presented By:

Kishore Kumar Kalathur Chenchu (M21CS058)
Prabhala Sandhya Gayatri (M21CS060)
Tejaswee A (M21CS064)

Objective

There exist many universally accepted graph clustering algorithms with the best strategies but usage of each algorithm depends on the use case. Given the constraints, optimizing these graph clustering algorithms is an NP-hard problem. Therefore, among the available best strategic graph clustering algorithms, our task is to explore a few of them that have a wider range of applications.

Graph Clustering Measures

The quality measures of a clustered graph is identified as follows:

1. Combination of less inter-cluster edges and more intra-cluster edges gives better and higher quality
2. Inseparable Cliques
3. Connected Clusters
4. Disjoint Cliques approaching maximum quality

Graph Clustering Approaches

The Graph Clustering Approaches are majorly classified as:

1. **Top-down:** In this approach starting with one-cluster, we split to form smaller clusters in iterative manner.
2. **Bottom-up**: It starts with singleton set of individual data points and merge them to form clusters.
3. **Local Optimization:** Starting with random clustering, we migrate towards the nodes.

Graph Clustering Algorithms

Top down:

- k-means - minimizing the distance squared
- k-center - minimizing the maximum distance
- k-median - minimizing the average distance

Bottom up:

- Greedy Agglomeration
- Markov Clustering and Random Walks

Local Optimization :

- Local Moving and Multilevel algorithm

Others:

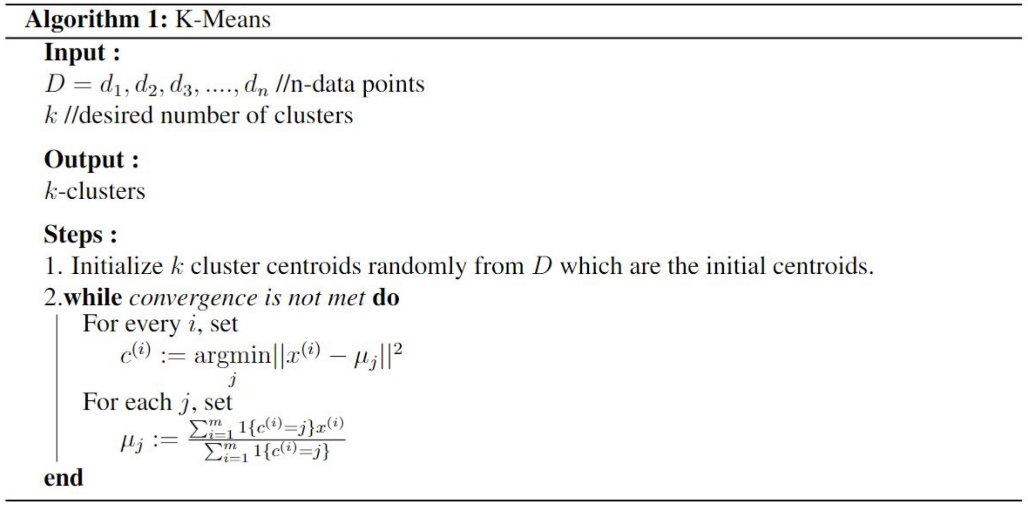
- Clique Percolation

TOP-DOWN APPROACH

k-Means Clustering

- *k*-Means stores the k-centroids that are useful to define the clusters.
- *k*-Means works for a set of samples having features with no labels mentioned. Hence, grouping the samples into clusters is used to find the related sample data.
- For finding centroids using *k*-Means, we iteratively choose the nearest data points by calculating Euclidean distance with the centroid and assigning such data points to a cluster by calculating the mean.
- Clustering the data points terminate when either the number of centroids are not changing further or it has reached the maximum number of iterations.

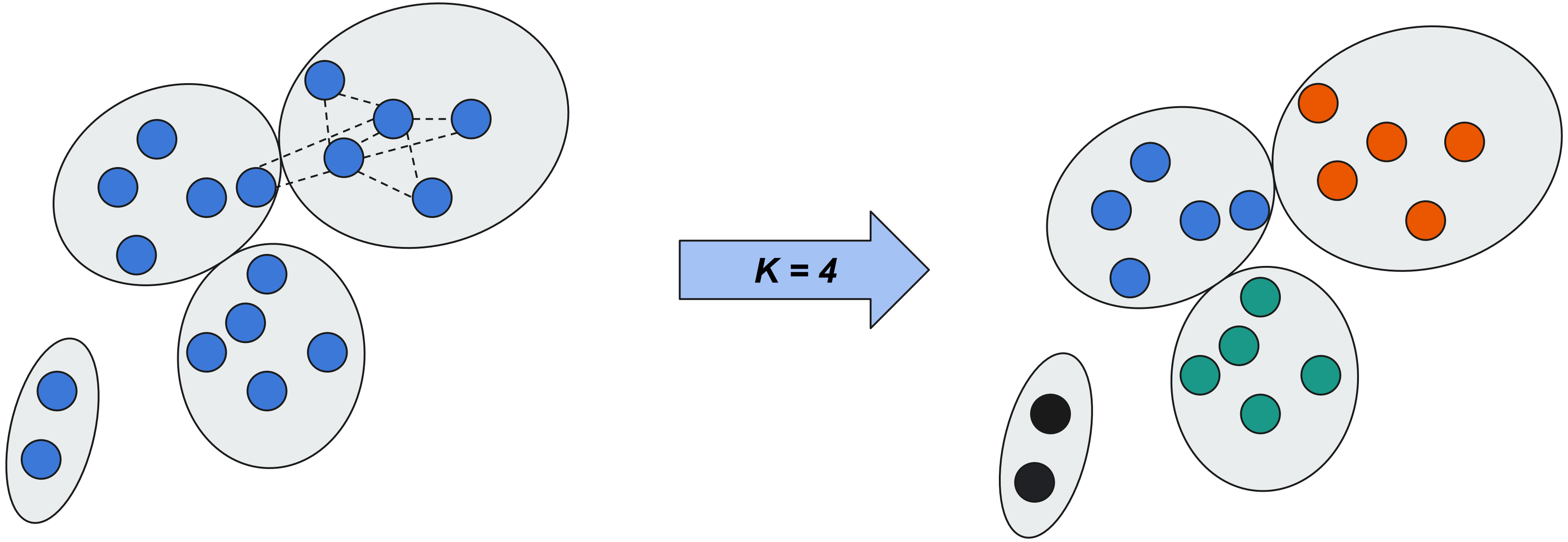
k-Means Algorithm



Algorithm 1: K-Means

```
Input :
$D = d_1, d_2, d_3, ...., d_n$ //n-data points
$k$ //desired number of clusters

Output :
$k$-clusters

Steps :
1. Initialize $k$ cluster centroids randomly from $D$ which are the initial centroids.
2.while convergence is not met do
    For every $i$, set
        $c^{(i)} := \underset{j}{\text{argmin}}||x^{(i)} - \mu_j||^2$
    For each $j$, set
        $\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)}=j\}x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)}=j\}}$
end
```

Example



k-Center Clustering

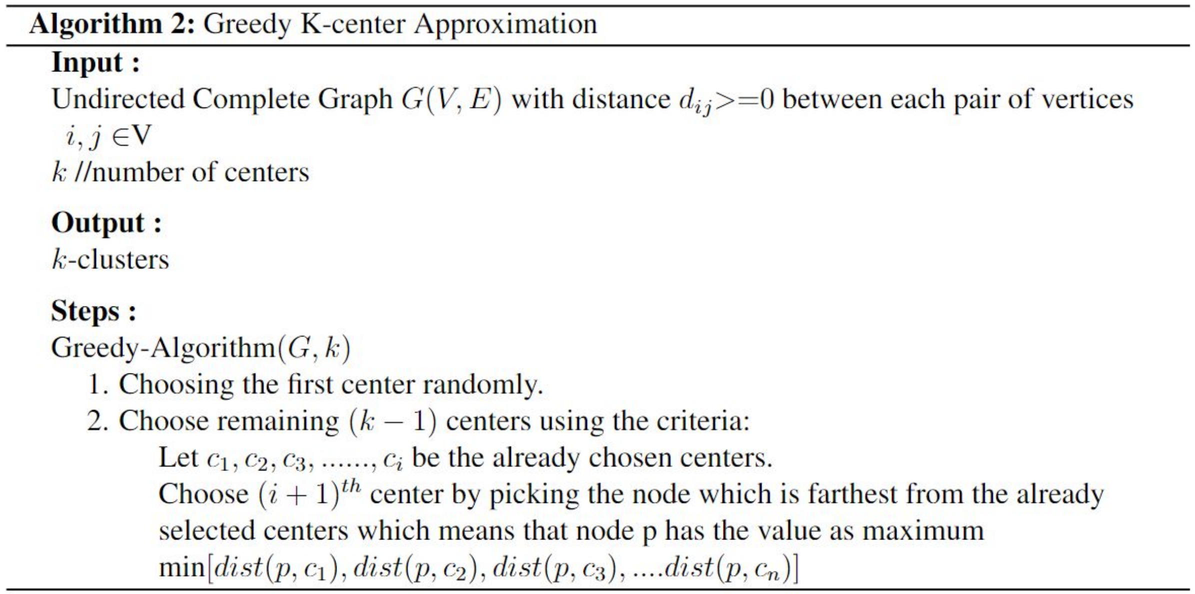
- *k*-Center Clustering, for each cluster, considers the farthest sample data from the centroid.

- On the clusters level, it takes into consideration of the worst cluster whose data point has the maximum distance from the centroid in comparison to other clusters.

- *k*-Center clustering is NP-Hard problem and solving them in polynomial time is highly unlikely to achieve.

- Therefore, approximation algorithms such Greedy algorithms, Local search are preferred.

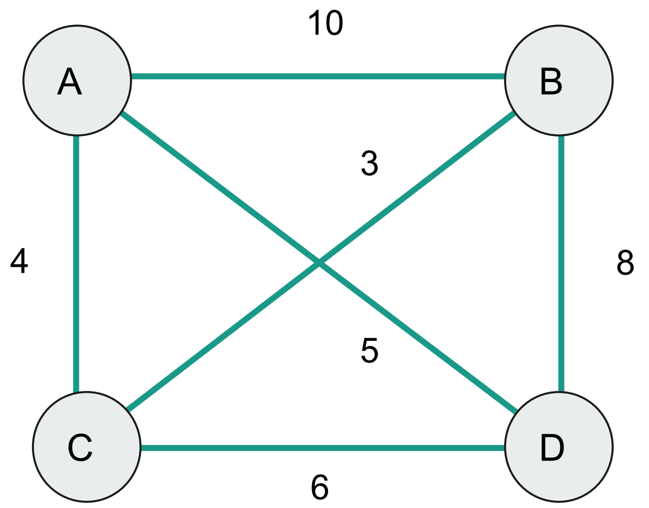
Greedy *k* center Algorithm



```
Algorithm 2: Greedy K-center Approximation
  Input :
  Undirected Complete Graph $G(V, E)$ with distance $d_{ij}$>=0 between each pair of vertices
   $i, j$ ∈V
  $k$ //number of centers
  Output :
  $k$-clusters
  Steps :
  Greedy-Algorithm$(G, k)$
    1. Choosing the first center randomly.
    2. Choose remaining $(k - 1)$ centers using the criteria:
         Let $c_1, c_2, c_3, ......, c_i$ be the already chosen centers.
         Choose $(i + 1)^{th}$ center by picking the node which is farthest from the already
         selected centers which means that node p has the value as maximum
         min$[dist(p, c_1), dist(p, c_2), dist(p, c_3), ....dist(p, c_n)]$
```

Example

Let us consider a hostel block where the rooms are distance (in meters) apart as shown in the graph below. The authorities decide to establish Wi-Fi connection with routers placed at specific distances so that every occupant is equally benefited. Our aim is to find minimum numbers of routers to be installed.



Given *k* = *2*, the most optimal solution is to place the routers near rooms *C* and *D* where the maximum distance becomes 6.

k- Median Clustering

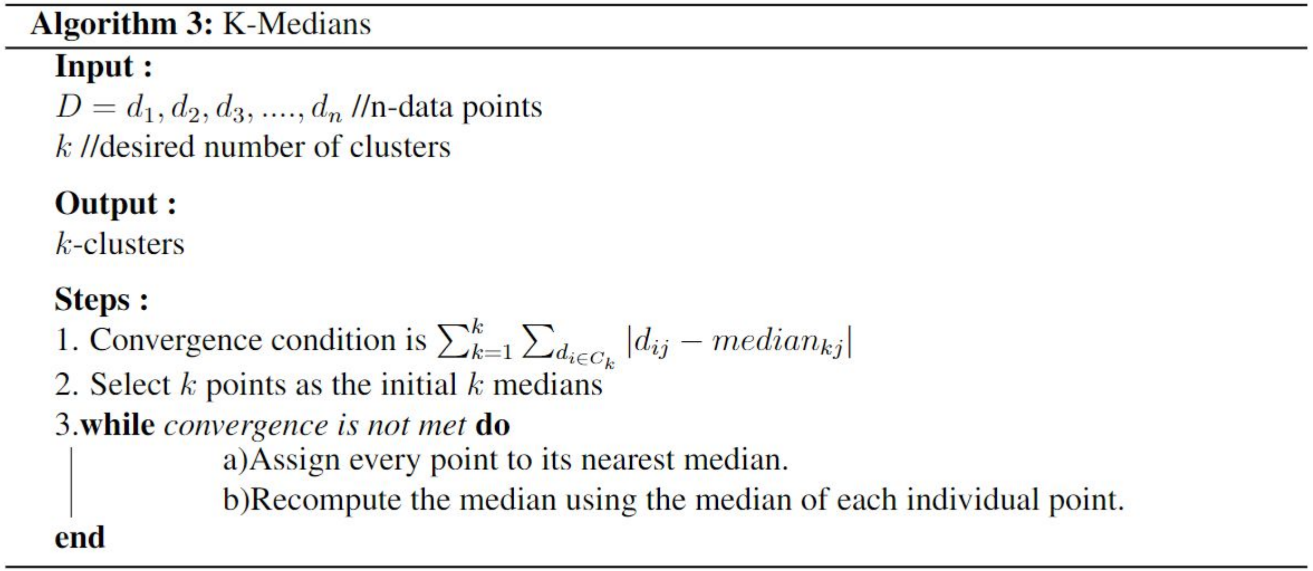
- Generally Medians are less sensitive to the outliers when compared to the Means.

- K-medians approach aims at minimizing the 1-norm distances (Manhattan-distance) of every data point and its closest center to the cluster.

- Using median as the statistic to determine the clusters' centers, it minimizes the distance between the data points to the center to form appropriate clusters.

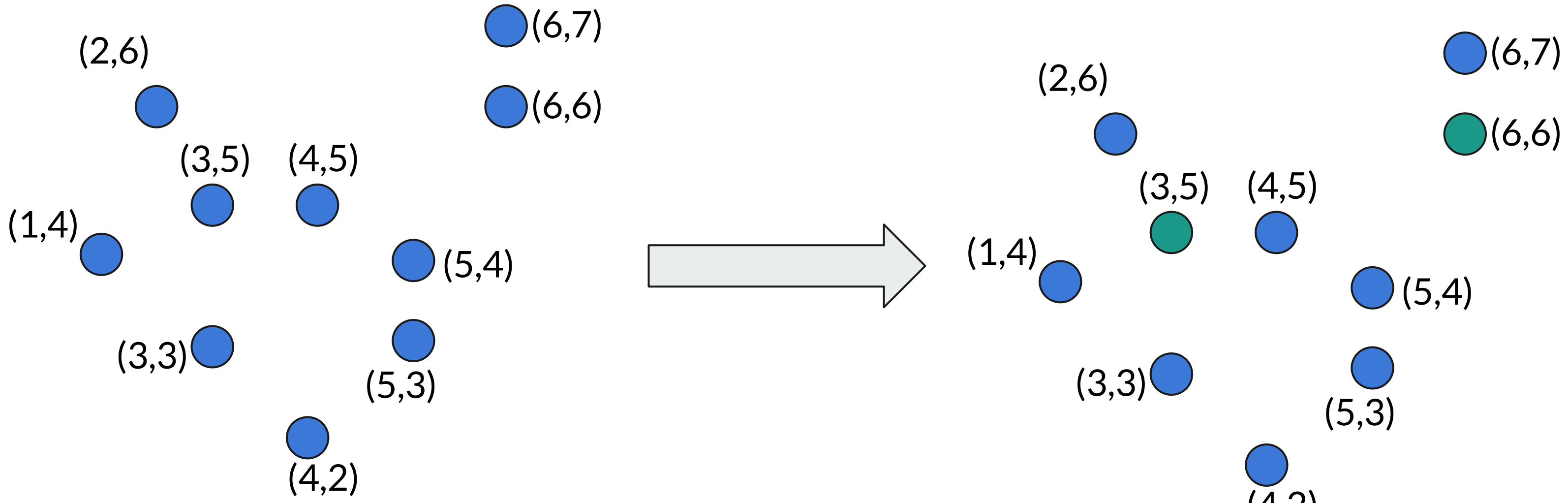
- Mean is highly vulnerable to outliers, even a single outlier can change the value of mean making it distant from other data points whereas median is highly resistant to outliers. This may require more than 50% of the data points to be differed in order to change the value of median.

k-Median Clustering

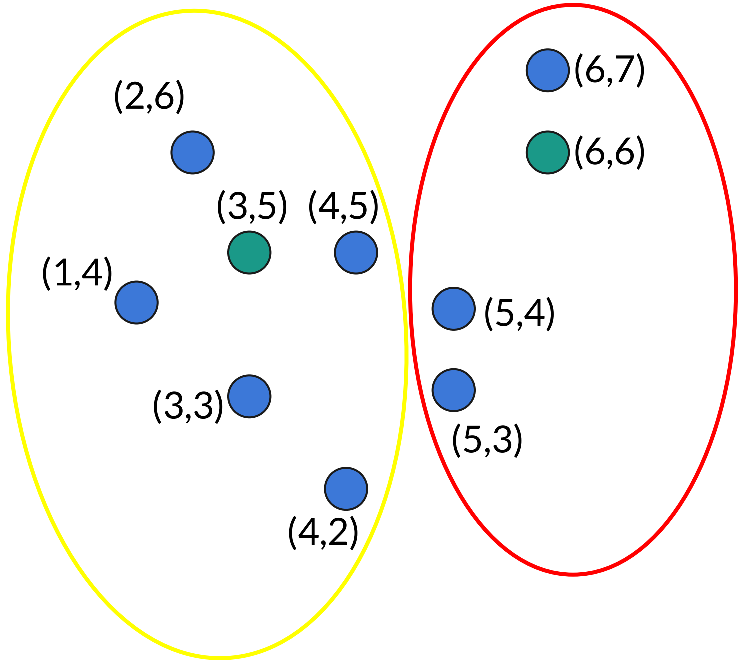


```
Algorithm 3: K-Medians
Input :
$D = d_1, d_2, d_3, ...., d_n$ //n-data points
$k$ //desired number of clusters

Output :
$k$-clusters

Steps :
1. Convergence condition is $\sum_{k=1}^{k} \sum_{d_i \in C_k} |d_{ij} - median_{kj}|$
2. Select $k$ points as the initial $k$ medians
3.while convergence is not met do
        a)Assign every point to its nearest median.
        b)Recompute the median using the median of each individual point.
end
```

Example

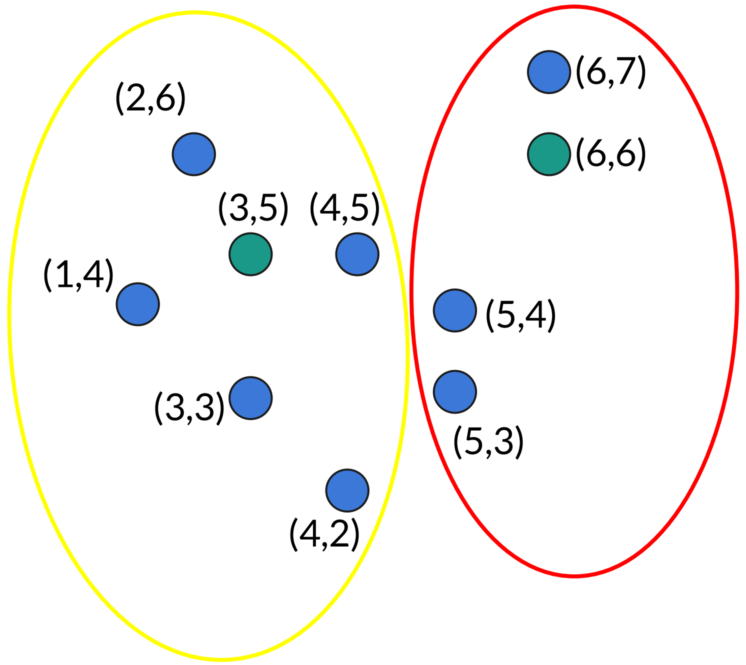


Randomly choosing c_1 as (3,5) and c_2 as (6,6)



Total Cost = 20

Points	Dissimilarity from c1	Dissimilarity from c2
(1,4)	3	7
(2,6)	2	4
(3,3)	2	6
(3,5)	-	-
(4,2)	4	6
(4,5)	1	3
(5,3)	4	4
(5,4)	3	3
(6,6)	-	-
(6,7)	5	1



Swap Cost = New cost - Prev Cost
= 21 - 20
= 1

Total Cost = 21

Randomly choosing c_1 as (3,3) and c_2 as (5,4)

Points	**Dissimilarity from c1**	**Dissimilarity from c2**
(1,4)	3	4
(2,6)	4	5
(3,3)	-	-
(3,5)	2	3
(4,2)	2	3
(4,5)	3	2
(5,3)	2	1
(5,4)	-	-
(6,6)	6	3
(6,7)	7	4

- New calculated cost is more than what we had with the medians as (3,5) and (6,6) hence, we terminate the process and the clusters obtained are shown in the graph otherwise the process would continue with the new means..

BOTTOM-UP APPROACH

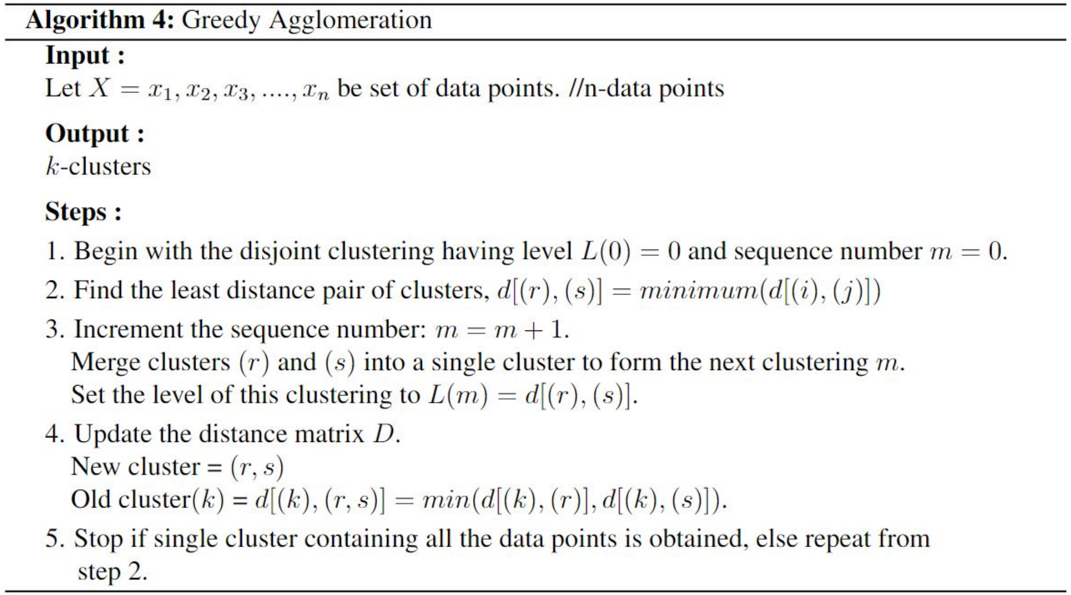
Greedy Agglomeration

- There is no requirement to specify the number of clusters required in advance.
- Algorithms under bottom-up approach consider each data point as a singleton cluster and iteratively agglomerates to form a single cluster containing all the sample points.
- Similar pair of clusters are grouped together as they go up in hierarchy.
- Based on the distance measurement generated, we use linkage function to group the clusters.

Characteristics of Greedy Agglomeration

- No prior information about the number of clusters is requited.
- With *n* number of data points, time complexity shoots up to having atleast $O(n^2 \log n)$.
- Operations cannot be undone once performed.
- Results in best output in some cases and easier to implement.
- It is sensitive to outliers, noise and having different sized clusters becomes unmanageable at times.

Greedy Agglomeration



```
Algorithm 4: Greedy Agglomeration
  Input :
  Let $X = x_1, x_2, x_3, ...., x_n$ be set of data points. //n-data points
  Output :
  $k$-clusters
  Steps :
  1. Begin with the disjoint clustering having level $L(0) = 0$ and sequence number $m = 0$.
  2. Find the least distance pair of clusters, $d[(r),(s)] = minimum(d[(i),(j)])$
  3. Increment the sequence number: $m = m + 1$.
     Merge clusters $(r)$ and $(s)$ into a single cluster to form the next clustering $m$.
     Set the level of this clustering to $L(m) = d[(r),(s)]$.
  4. Update the distance matrix $D$.
     New cluster = $(r,s)$
     Old cluster($k$) = $d[(k),(r,s)] = min(d[(k),(r)], d[(k),(s)])$.
  5. Stop if single cluster containing all the data points is obtained, else repeat from
      step 2.
```

Example

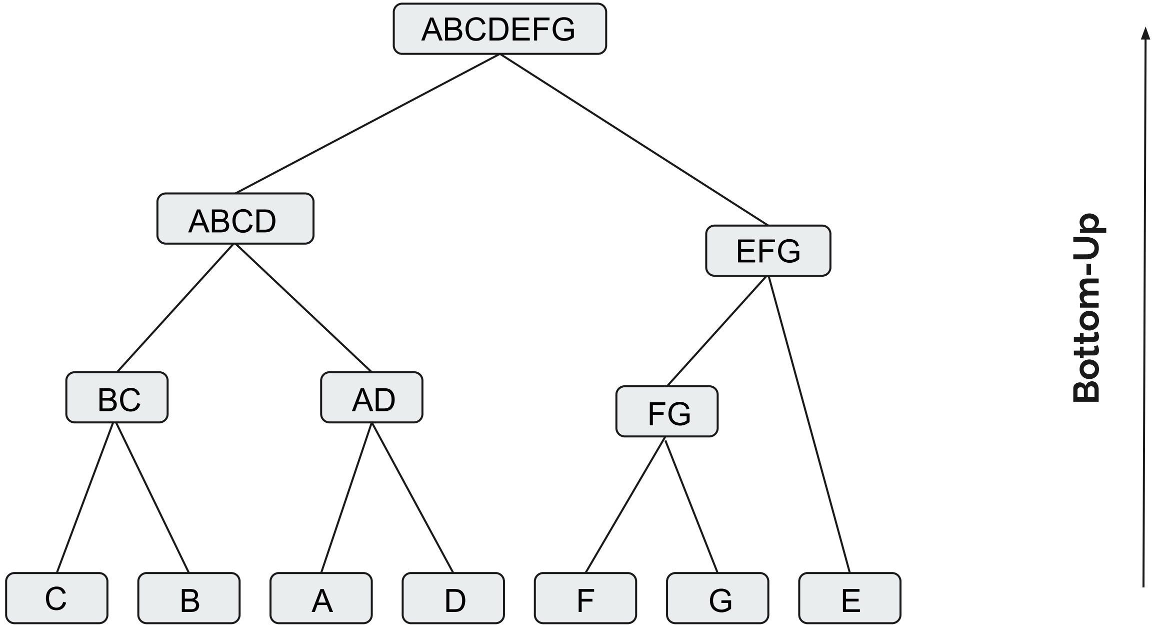


Figure: Bottom-Up Greedy Agglomeration to group a pair of clusters and form a single entity

Markov Clustering and Random Walks

Markov Clustering:

Markov Clustering Algorithm is based on Random Walks.

Random Walks:

Given a graph with clusters, we find more edges within a cluster, and less edges between clusters. This implies that, if we start our random walk at a node, then we are more likely to stay within a cluster to which the node belongs, than travelling between clusters. To know the where the flow trends gather, we use these random walks on graphs, which results in finding the clusters. These random walks are calculated using Markov Chains.

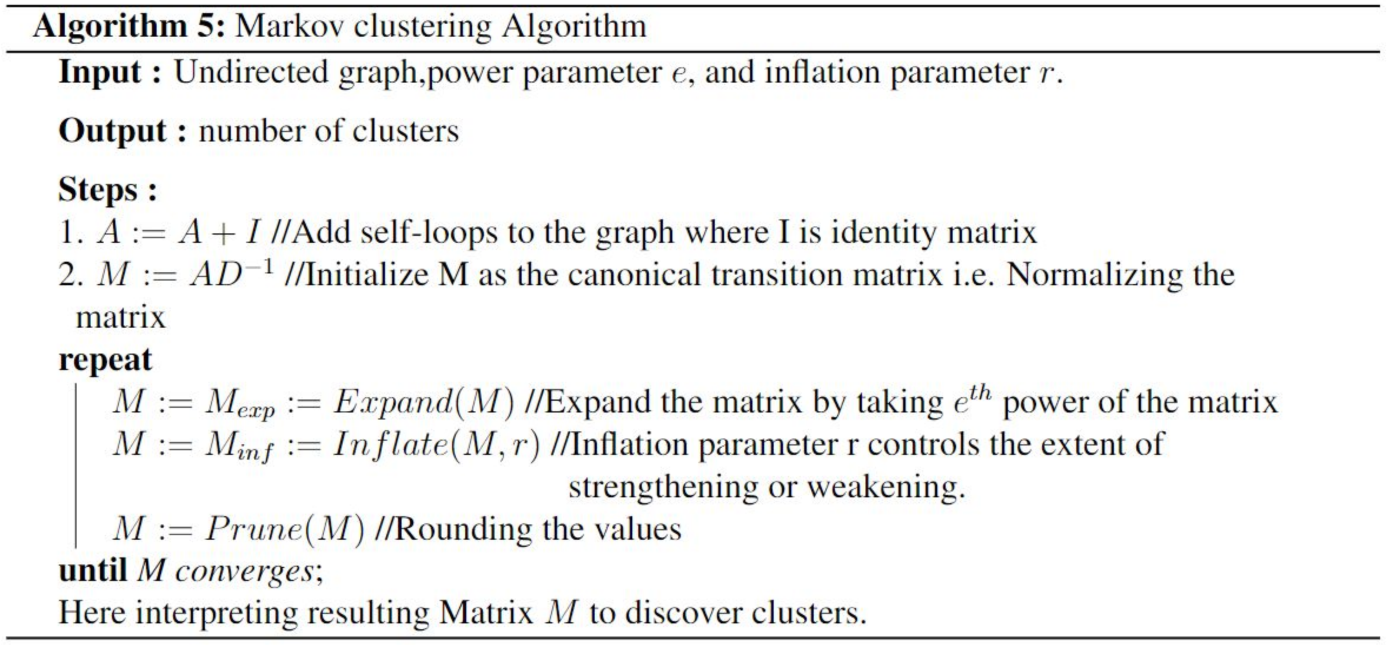
Markov Clustering and Random Walks

- In MCL, the following two processes are alternated repeatedly:
  a. Expansion (taking the Markov Chain transition matrix powers)
  b. Inflation
- The expansion operator is responsible for allowing flow to connect different regions of the graph.
- Expansion is carried out by taking the Markov Chain powers.
- The inflation operator is responsible for both strengthening and weakening of current, parameter r is used to control the same.
- Inflation is re-normalizing a single column when expanded.
- Edge weights connecting two vertices within clusters will be higher in the initial phase while having lower values connecting a pair of clusters.

Markov Clustering Algorithm

1. Input parameters: Undirected graph, power e and inflation parameter r .
2. Initialize a matrix with the corresponding edge weights and add self loops to each node.
3. Normalize the matrix column-wise.
4. Multiply the matrix by self for e times.
5. Inflate the matrix by r parameter.
6. Repeat above two steps (4 and 5) until we find two successive matrices the same (convergence is achieved).
7. Find for clusters so formed.

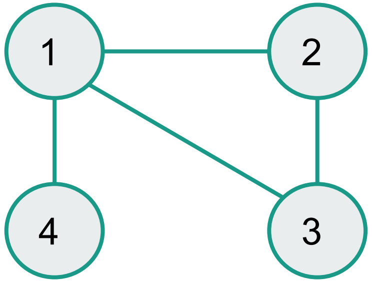
Markov Clustering Algorithm - Pseudocode



Algorithm 5: Markov clustering Algorithm

```
Input : Undirected graph,power parameter $e$, and inflation parameter $r$.
Output : number of clusters
Steps :
1. $A := A + I$ //Add self-loops to the graph where I is identity matrix
2. $M := AD^{-1}$ //Initialize M as the canonical transition matrix i.e. Normalizing the
   matrix
repeat
    $M := M_{exp} := Expand(M)$ //Expand the matrix by taking $e^{th}$ power of the matrix
    $M := M_{inf} := Inflate(M, r)$ //Inflation parameter r controls the extent of
                                      strengthening or weakening.
    $M := Prune(M)$ //Rounding the values
until $M$ converges;
Here interpreting resulting Matrix $M$ to discover clusters.
```

Example



$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \xrightarrow{\text{Adding edges of self loops}} \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \xrightarrow{\text{Normalizing}} \begin{pmatrix} 1/4 & 1/3 & 1/2 & 1/3 \\ 1/4 & 1/3 & 0 & 1/3 \\ 1/4 & 0 & 1/2 & 0 \\ 1/4 & 1/3 & 0 & 1/3 \end{pmatrix}$$

Example Contd.

$$\begin{pmatrix} 1/4 & 1/3 & 1/2 & 1/3 \\ 1/4 & 1/3 & 0 & 1/3 \\ 1/4 & 0 & 1/2 & 0 \\ 1/4 & 1/3 & 0 & 1/3 \end{pmatrix}^2 \overset{\text{Power of 2}}{\underset{\text{Inflation of 2}}{=}} \begin{pmatrix} 0.35 & 0.31 & 0.38 & 0.31 \\ 0.23 & 0.31 & 0.13 & 0.31 \\ 0.19 & 0.08 & 0.38 & 0.08 \\ 0.23 & 0.31 & 0.13 & 0.31 \end{pmatrix}^2 \overset{\text{Power of 2}}{\underset{\text{Inflation of 2}}{=}} \ldots\ldots = \begin{pmatrix} 1 & 0.33 & 0.50 & 0.33 \\ - & 0.33 & - & 0.33 \\ - & - & 0.50 & - \\ - & 0.33 & - & 0.33 \end{pmatrix}$$

LOCAL OPTIMIZATION APPROACH

Local Moving and Multilevel Algorithm

- Given an input graph, coarsener, refiner and the reduction factor, the Multilevel Clustering Algorithm operates in 2 phases:
  1. Coarsening Phase
  2. Refinement Phase

- In the Coarsening Phase, we may use any of the Cluster Joining (CJ) or Vertex Moving (VM) algorithms, which are known as Local Moving Approaches. These help to find the best move for each vertex by repeatedly iterating through all vertices in a randomized fashion.

- In the Refinement Phase we stick to use VM approach, since CJ algorithms may not find joinable clusters resulting to an optimal clustering.

Local Moving and Multilevel Algorithm

Coarsening Phase:

- A sequence of graphs called the coarsening levels are produced by the Coarsening Phase, in which the first coarsening level is the input graph.Initially each single vertex is a cluster for the coarsener and it starts clustering.

- The coarsener runs until
  1. It terminates
  2. The number of clusters decrease by a certain percentage, as compared with the start, where the percentage is called reduction factor

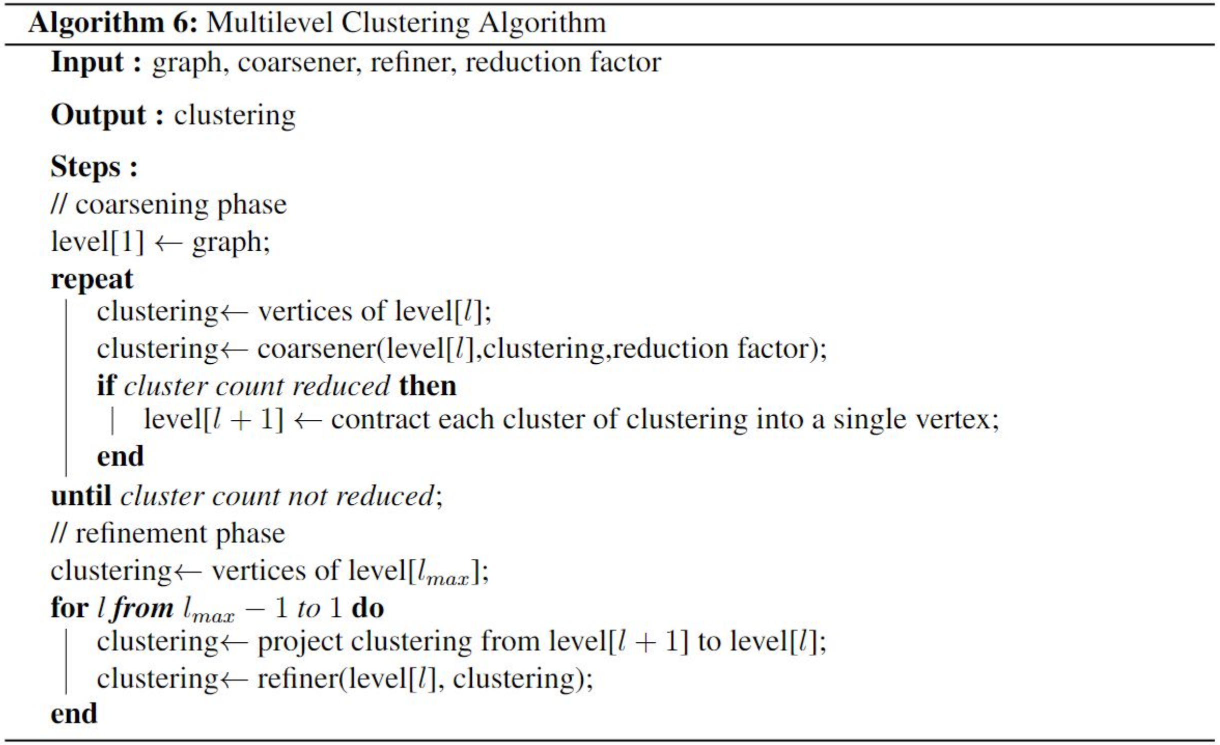
- The next coarsening level is formed where each cluster is contracted to result into a single vertex. When there are no further changes in clusters in 2 subsequent layers, the coarsening phase is known to reach a fixed point and it ends.

Local Moving and Multilevel Algorithm

Refinement Phase:

- The algorithm then visits all the coarsening levels in a reverse order, from the coarsest graph to the original graph.
- It computes clustering at each level. This phase is called the Refinement Phase.
- The final clustering of a level is projected to its next subsequent level and this forms the initial clustering of the subsequent level, on which a refiner is applied to compute the final clustering.
- Here the refiner is preferred to be a VM algorithm, since CJ algorithms may not find joinable clusters resulting to an optimal clustering.

Multilevel Clustering Algorithm



Algorithm 6: Multilevel Clustering Algorithm

```
Input : graph, coarsener, refiner, reduction factor
Output : clustering
Steps :
// coarsening phase
level[1] ← graph;
repeat
    clustering← vertices of level[$l$];
    clustering← coarsener(level[$l$],clustering,reduction factor);
    if cluster count reduced then
        level[$l + 1$] ← contract each cluster of clustering into a single vertex;
    end
until cluster count not reduced;
// refinement phase
clustering← vertices of level[$l_{max}$];
for $l$ from $l_{max} - 1$ to 1 do
    clustering← project clustering from level[$l + 1$] to level[$l$];
    clustering← refiner(level[$l$], clustering);
end
```

Clique Percolation

- In social networking graphs, it is important to find the communities which consist of similar type of people having similar though process. Hence, the key task become sto find the strongly connected subgraphs in a graph. These subgraphs are called cliques

- It is so possible to have a single person belonging to multiple communities i.e a node in a community may be shared with certain other number of communities.

- Therefore, clique percolation algorithms aims at identifying the overlapping communities.

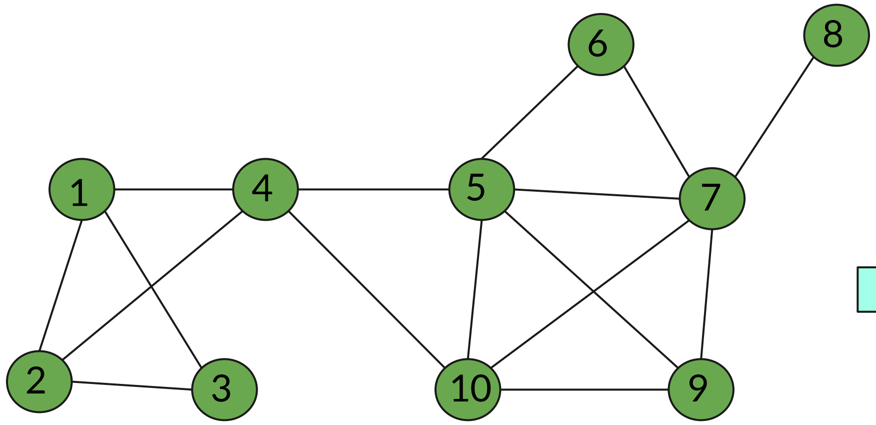
- It initiates with identifying k-cliques and maintaining the adjacent clique (having k-1 nodes) in a community.

- Further in this process, k-cliques are considered to be different entities when it is not possible to group furthermore as a community.

Clique Percolation

```
Algorithm 7: Clique Percolation Method
  Input : Complex Graph $G$
  Output : Candidate community set $C_L$
  Steps :
  1. UF $\leftarrow$ Union Find data structure
  2. Dict $\leftarrow$ Empty Dictionary
  for each k clique $c_k \in G$ do
      S $\leftarrow \phi$
      for each $(k-1)$clique $c_{k-1} \subset c_k$ do
          if $c_{k-1} \in Dict.keys()$ then
              $P \leftarrow$ UF.Find(Dict[$c_{k-1}$])
          end
          else
              $P \leftarrow$ UF.MakeSet()
              Dict[$c_{k-1}$] $\leftarrow p$
          end
          $S \leftarrow S \cup \{p\}$
      end
      UF.Union($S$)
  end
```

Example



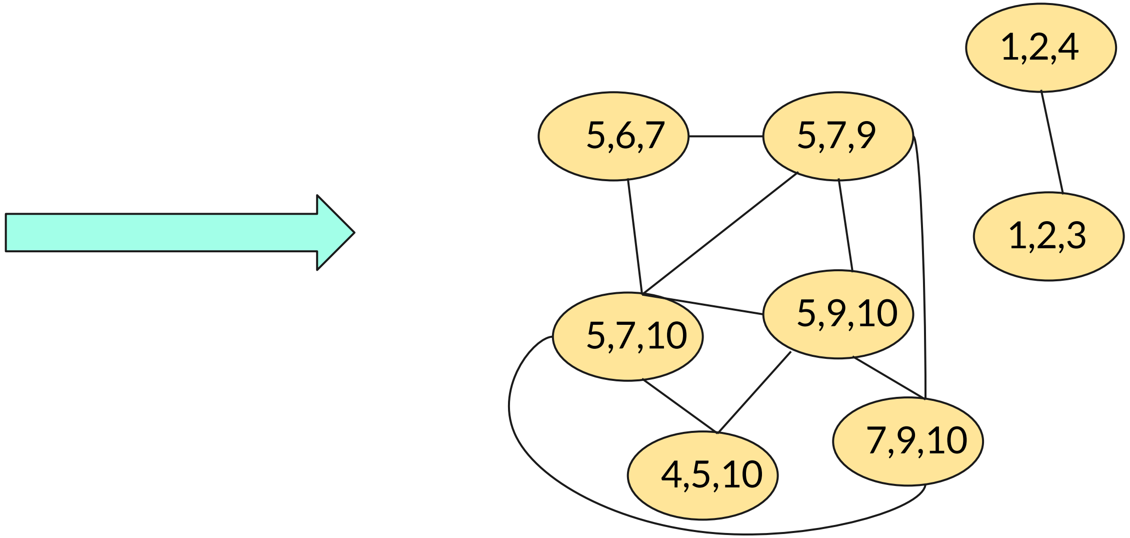
Cliques of size k=3:

- {1,2,4}
- {1,2,3}
- {4,5,10}
- {5,6,7}
- {5,7,9}
- {5,7,10}
- {5,9,10}
- {7,9,10}

Example contd.

Cliques of size k=3:

- {1,2,4}
- {1,2,3}
- {4,5,10}
- {5,6,7}
- {5,7,9}
- {5,7,10}
- {5,9,10}
- {7,9,10}



K-Clique Communities:

- {1,2,3,4}
- {4,5,6,7,9,10}