UNIT-7 Introduction to community discovery

Introduction to community discovery

- Many real world problems can be effectively modeled as complex relationship networks where nodes represent entities of interest and edges mimic the interactions or relationships among them.
- Discovering communities in complex networks means **grouping nodes similar to each other, to uncover latent information about them**. There are hundreds of different algorithms to solve the community detection task, each with its own understanding and definition of what a "community" is.
- Community discovery is **the problem of extracting all the communities in a given network.**
- **C**ommunity identification is the problem of identifying the community to which a given set of nodes from the network belong.

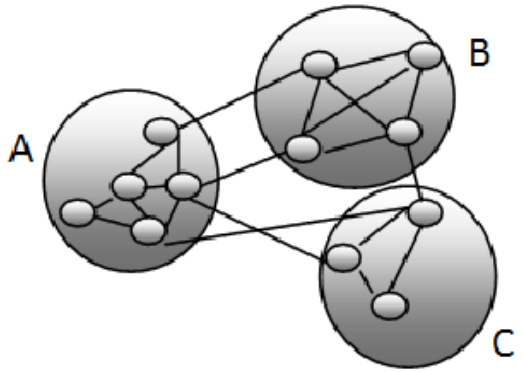
- Complex networks commonly have community structures, in which pairs of nodes belonging to the same community are closely connected, while pairs of nodes belonging to different communities are relatively sparsely connected.
- Community detection aims at grouping nodes in accordance with the relationships among them to form strongly linked sub graphs from the entire graph.
- The study of complex relationship networks, recently referred to as network science, can provide insight into their structures, properties and emergent behaviors.
- Find rigorous methods for uncovering and understanding important network or sub-network (community).
- Extracting such community structure and leveraging them for predicting the emergent, critical, and causal nature of such networks in a dynamic setting is of growing importance.

- **Why Community Detection?**
  - When analyzing different networks, it may be important to discover communities inside them.
  - Community detection techniques are useful for social media algorithms to discover people with common interests and keep them tightly connected.
  - Community detection can be used detect groups with similar properties and extract groups for various reasons. For example, this technique can be used to discover manipulative groups inside a social network or a stock market.
  - Communities can be implicit or explicit. Explicit communities are those, in which a grouping is predefined and members joining the group form a community. In this case Communities are directly visible, for example whatsapp group.
  - Implicit communities on the other hand do not have any predefined classification. We have to analyze the activities of the individuals to form the community. Community detection is used for implicit communities only.

Issues in Community detection:

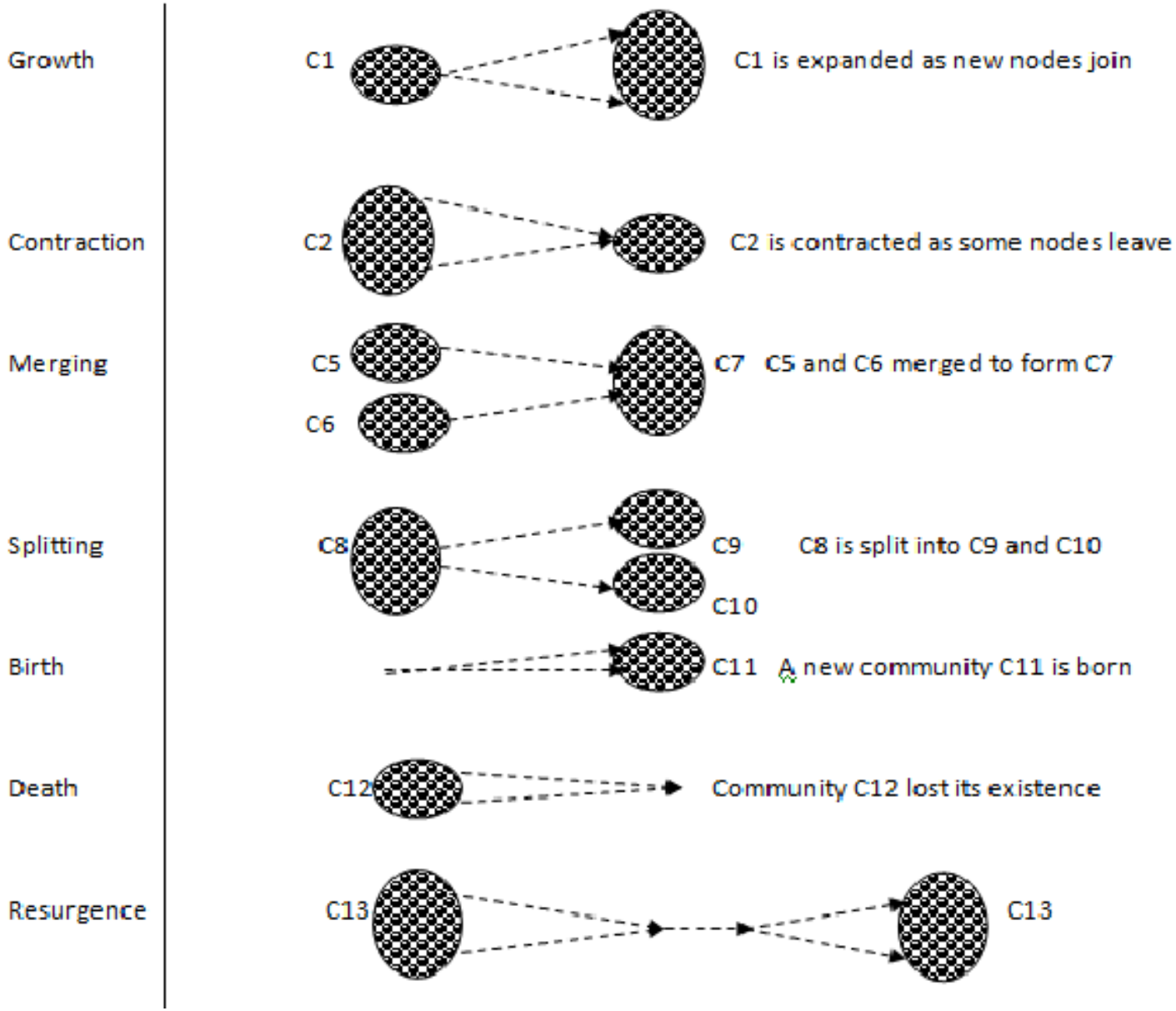
No Precise Definition:

- According to Kernighan –Lin algorithm ,”Communities are those parts of graph that have less ties with rest of the graph”.
- Wasserman et al. considered Community as maximal subgraph, that cannot be extended by addition of more vertices without loosing its property.
- But in very simple words, a community can be defined as a subgraph with more intra cluster edges than inter cluster edges. It means they are group of nodes which have more interactions among themselves than others.

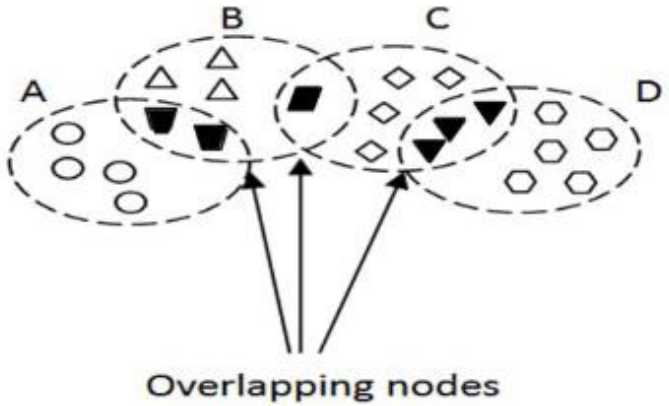


- In spite of community detection being one of the strongest fields of research in social media mining, a proper definition of this problem does not exist.
- Due to lack of proper definition there are different views of community by different researchers.
- These different views are not disjoint, they can sometime lead to common result.
- Challenges Involved
  - i. Analyzing which view of community can be beneficial under what conditions.
  - ii. Selecting a particular viewpoint for community to start community detection

- **Dynamic Nature of Communities**
- Most of the work in community detection is performed on a static network. These cannot be applied on Complex Network which is ever changing like Social Media. Dynamic communities keep on changing with time.
- As dynamic communities change with time, different situations are faced in detection of such communities.
  - Growth: New nodes can be included in a community with time.
  - Contraction: Some nodes can leave the community, making the community smaller.
  - Merging: Different communities can combine with time, resulting in a bigger merged community.
  - Splitting: Two or more communities may be formed by splitting one community.
  - Birth: A new community can emerge which was not existent at an earlier time interval.
  - Death: A community can entirely disappear at any time.
  - Resurgence: Community can be dormant for certain period and then reappear as if nothing happened.


Growth
C1
C1 is expanded as new nodes join
Contraction
C2
C2 is contracted as some nodes leave
Merging
C5
C6
C7
C5 and C6 merged to form C7
Splitting
C8
C9
C10
C8 is split into C9 and C10
Birth
C11
A new community C11 is born
Death
C12
Community C12 lost its existence
Resurgence
C13
C13

- **Communities Overlap an Issue**
  - Most of the early research in community detection assumes community to be disjoint group of densely connected nodes. But in real world a person can be member of more than one community.
  - This also applies in Social Media. It means generally the communities are not disjoint, they are overlapping in nature.

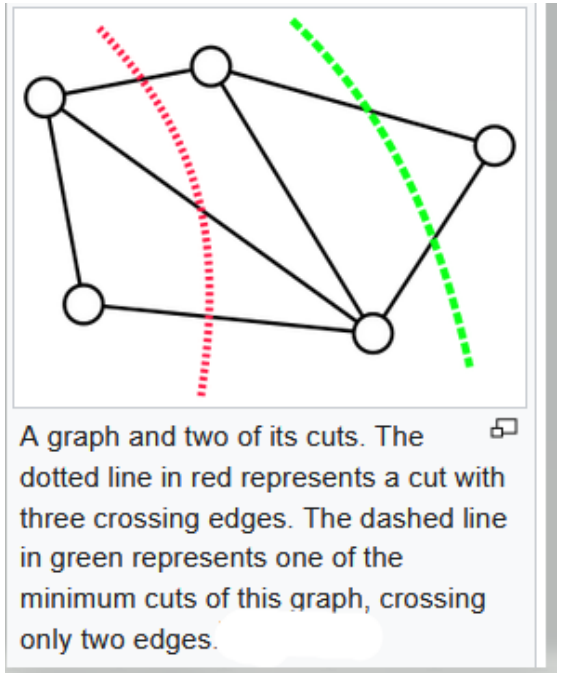


Core Methods

- Informally, a community in a network is a group of nodes with greater ties internally than to the rest of the network.
- This intuitive definition has been formalized in a number of competing ways, usually by way of a quality function, which quantifies the goodness of a given division of the network into communities.
- Some of these quality metrics, such as Normalized Cuts, Modularity, etc., but none has gained universal acceptance since no single metric is applicable in all situations.
  - The normalized cut criterion measures both the total dissimilarity between the different groups as well as the total similarity within the groups.
  - Modularity is a measure of community detection which measures the strength of division of a network into communities. Networks with a high modularity have a dense connection between the nodes within modules but a spread connections between nodes in different modules.

- Algorithms for community discovery vary on a number of important dimensions, including their approach to the problem as well as their performance characteristics.
- An important dimension on which algorithms vary in their approaches is whether or not they explicitly optimize a specific quality metric.
  - ✓ Spectral methods, the Kernighan-Lin algorithm are all examples of algorithms which explicitly try to optimize a specific quality metric,
  - ✓ Other algorithms, such as Markov Clustering (MCL) and clustering via shingling do not do so.
- Another dimension on which algorithms vary is in how (or even whether) they let the user control the granularity of the division of the network into communities.
  - ✓ Some algorithms (such as spectral methods) are mainly meant for bi-partitioning the network, but this can be used to recursively subdivide the network into as many communities as desired.
  - ✓ Other algorithms such as agglomerative clustering or MCL allow the user to indirectly control the granularity of the output communities through certain parameters.
- Another important characteristic differentiating community discovery algorithms is the importance they attach to a balanced division of the network.
- Algorithms also vary in their scalability to big networks, with multi-level clustering algorithms scaling better than many other approaches.

- Given a graph G = (V, E) and a subset S of V , the cut δ(S) induced by S is the subset of edges (i, j) ∈ E such that |{i, j} ∩ S| = 1.
- That is, δ(S) consists of all those edges with exactly one endpoint in S.
- In graph theory, a cut is a set of edges removal of which divides a connected graph into two non-overlapping (disjoint) subsets.
- The minimum cut (or min-cut) is defined as the minimum number of edges, when removed from a graph, divide the graph into two disjoint sets.



A graph and two of its cuts. The dotted line in red represents a cut with three crossing edges. The dashed line in green represents one of the minimum cuts of this graph, crossing only two edges.

Quality Functions

- A variety of quality functions or measures have been proposed in the literature to capture the goodness of a division of a graph into clusters.
- Let A denotes the adjacency matrix of the network or graph, with $A(i, j)$ representing the edge weight or affinity between nodes i and j, and V denotes the vertex or node set of the graph or network.

The normalized cut of a group of vertices $S \subset V$ is defined as[

$$Ncut(S) = \frac{\sum_{i \in S, j \in \bar{S}} A(i,j)}{\sum_{i \in S} degree(i)} + \frac{\sum_{i \in S, j \in \bar{S}} A(i,j)}{\sum_{j \in \bar{S}} degree(j)}$$

- In words, the normalized cut of a group of nodes S is the sum of weights of the edges that connect S to the rest of the graph, normalized by the total edge weight of S and that of the rest of the graph $\bar{S}$.
- The groups with low normalized cut make for good communities, as they are well connected amongst themselves but are sparsely connected to the rest of the graph.

- **The *conductance*** of a group of vertices $S \subset V$ is defined as

$$Conductance(S) = \frac{\sum_{i \in S, j \in \bar{S}} A(i,j)}{\min(\sum_{i \in S} degree(i), \sum_{i \in \bar{S}} degree(i))}$$

- The normalized cut (or conductance) of a division of the graph into k clusters $V1, \ldots, Vk$ is the sum of the normalized cuts (or conductances) of each of the clusters $Vi\{i = 1, \ldots, k\}$

- The *Kernighan-Lin (KL) objective* looks to minimize the edge cut under the constraint that all clusters be of the same size (making the simplifying assumption that the size of the network is a multiple of the number of clusters)

$$KLObj(V_1, \ldots, V_k) = \sum_{i \neq j} A(V_i, V_j) \text{subject to} |V_1| = |V_2| = \ldots = |V_k|$$

- Here $A(Vi, Vj)$ denotes the sum of edge affinities between vertices in Vi and Vj, i.e. $A(Vi, Vj) = u \in Vi, v \in Vj\ A(u, v)$

- ***Modularity*** :
- Measures the goodness of a clustering of a graph.
- Advantages of modularity is that it is independent of the number of clusters that the graph is divided into.
- The intuition behind the definition of modularity is that the farther the subgraph corresponding to each community is from a random subgraph the better or more significant the discovered community structure is.
- The modularity *Q* for a division of the graph into *k* clusters *{V1, . . . , Vk}* is given by:

$$Q = \sum_{c=1}^{k} \left[\frac{A(V_i, V_i)}{m} - \left(\frac{degree(V_i)}{2m} \right)^2 \right]$$

- In the above, the *Vi*s are the clusters, *m* is the number of edges in the graph and *degree*(*Vi*) is the total degree of the cluster *Vi*.
- For each cluster, we take the difference between the fraction of edges internal to that cluster and the fraction of edges that would be expected to be inside a random cluster with the same total degree.

The Kernighan-Lin(KL) algorithm

- The KL algorithm is one of the classic graph partitioning algorithms which optimizes the KL objective function i.e**. minimize the edge cut while keeping the cluster sizes balanced.**
- The algorithm is iterative in nature and starts with an initial bipartition of the graph.
- At each iteration, the algorithm searches for a subset of vertices from each part of the graph such that swapping them will lead to a reduction in the edge cut.
- The ***gain gv* of a vertex *v*** is the reduction in edge-cut if vertex ***v*** is moved from its current partition to the other partition.
- The KL algorithm repeatedly selects from the larger partition the vertex with the largest gain and moves it to the other partition;
- A vertex is not considered for moving again if it has already been moved in the current iteration.
- After a vertex has been moved, the gains for its neighboring vertices will be updated in order to reflect the new assignment of vertices to partitions.

- Problem: Divide a weighted graph with 2n nodes into two parts, each of size n, to minimize the sum of the weights crossing the two parts.

Divide the network into 2 parts A, B of equal size arbitrarily.

Repeat until no more vertices are left:

Select $a_i \in$ A, $b_i \in$ B, such that the reduction in cost is as large as possible and neither a_i, b_i has been chosen before

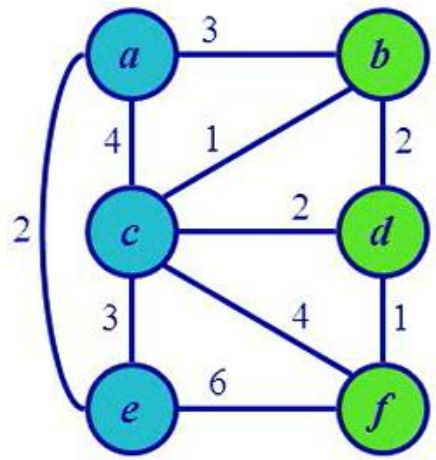
Swap a_i and b_i

Let C_i be the cost of the partition after swapping a_i,b_i

Return (A′,B′) corresponding to the smallest C_i observed.

While cost continues to be reduced

Call using the returned partition as the new starting point



Given:

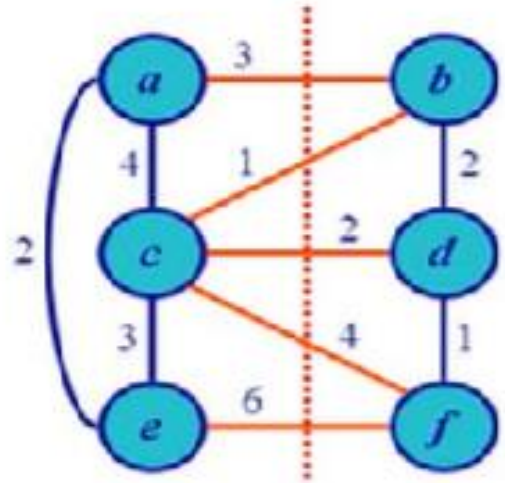
Initial weighted graph *G* with

$V(G) = \{ a, b, c, d, e, f \}$

Start with any partition of V(G) into X and Y, say

$X = \{ a, c, e \}$

$Y = \{ b, d, f \}$



cut-size = 3+1+2+4+6 = 16

$X = \{ a, c, e \}$
$Y = \{ b, d, f \}$

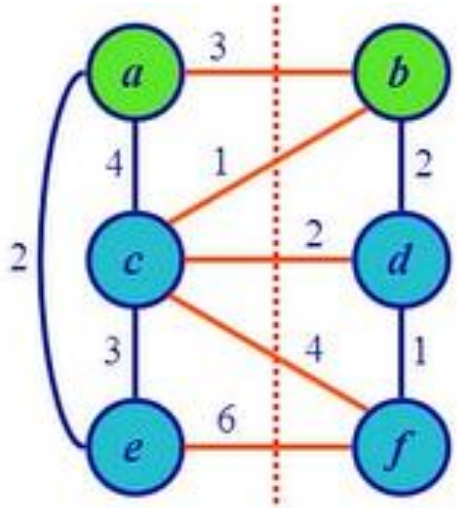
Compute the gain values of moving node x to the others set:

$G_x = E_x - I_x$

E_x = cost of edges connecting node x with the other group (extra)

I_x = cost of edges connecting node x within its own group (intra)

$G_a = E_a - I_a = -3 \quad (= 3 - 4 - 2)$
$G_c = E_c - I_c = \ \ 0 \quad (= 1 + 2 + 4 - 4 - 3)$
$G_e = E_e - I_e = +1 \quad (= 6 - 2 - 3)$
$G_b = E_b - I_b = +2 \quad (= 3 + 1 - 2)$
$G_d = E_d - I_d = -1 \quad (= 2 - 2 - 1)$
$G_f = E_f - I_f = +9 \quad (= 4 + 6 - 1)$



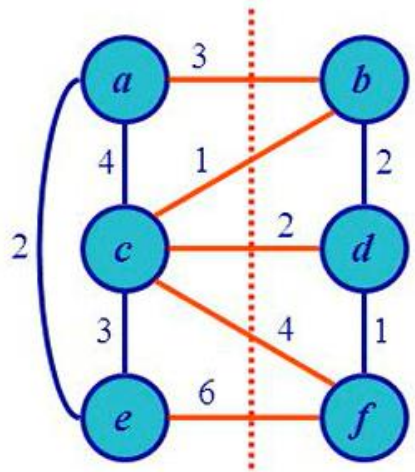
Cost saving when exchanging a and b is essentially $G_a + G_b$

However, the cost saving 3 of the direct edge was counted twice. But this edge still connects the two groups

Hence, the real "gain" (i.e. cost saving) of this exchange is $g_{ab} = G_a + G_b - 2c_{ab}$

$X = \{ a, c, e \}$
$Y = \{ b, d, f \}$

$G_a = E_a - I_a = -3 \quad (= 3 - 4 - 2)$
$G_b = E_b - I_b = +2 \quad (= 3 + 1 - 2)$
$g_{ab} = G_a + G_b - 2c_{ab} = -7 \quad (= -3 + 2 - 2 \cdot 3)$



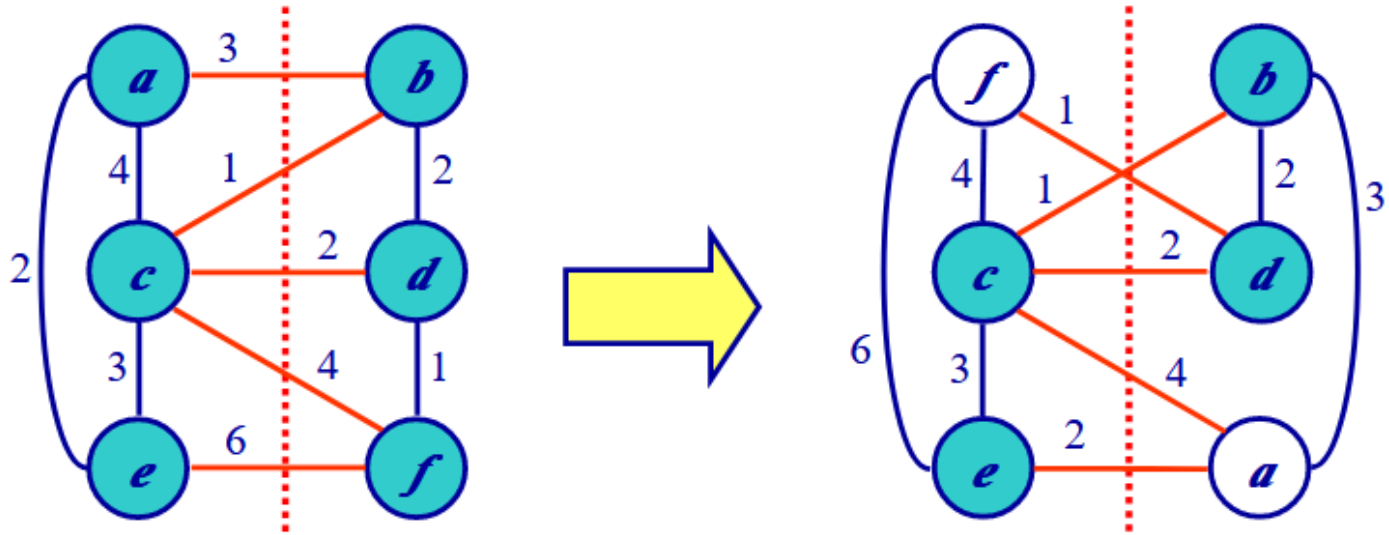
cut-size = 16

Pair with maximum gain

$G_a = -3$ $G_b = +2$
$G_c = 0$ $G_d = -1$
$G_e = +1$ $G_f = +9$

Compute all the gains

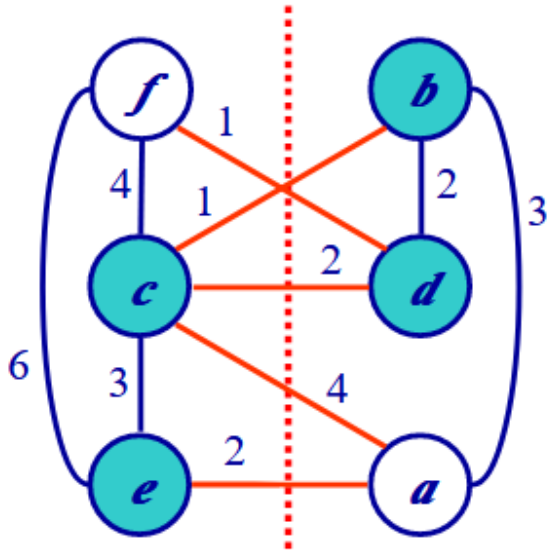
$g_{ab} = G_a + G_b - 2w_{ab} = -3 + 2 - 2\cdot 3 = -7$
$g_{ad} = G_a + G_d - 2w_{ad} = -3 - 1 - 2\cdot 0 = -4$
$g_{af} = G_a + G_f - 2w_{af} = -3 + 9 - 2\cdot 0 = +6$
$g_{cb} = G_c + G_b - 2w_{cb} = 0 + 2 - 2\cdot 1 = 0$
$g_{cd} = G_c + G_d - 2w_{cd} = 0 - 1 - 2\cdot 2 = -5$
$g_{cf} = G_c + G_f - 2w_{cf} = 0 + 9 - 2\cdot 4 = +1$
$g_{eb} = G_e + G_b - 2w_{eb} = +1 + 2 - 2\cdot 0 = +3$
$g_{ed} = G_e + G_d - 2w_{ed} = +1 - 1 - 2\cdot 0 = 0$
$g_{ef} = G_e + G_f - 2w_{ef} = +1 + 9 - 2\cdot 6 = -2$



Exchange nodes *a* and *f*

Then lock up nodes *a* and *f*

$$g_{af} = G_a + G_f - 2c_{af} = -3 + 9 - 20 = +6$$



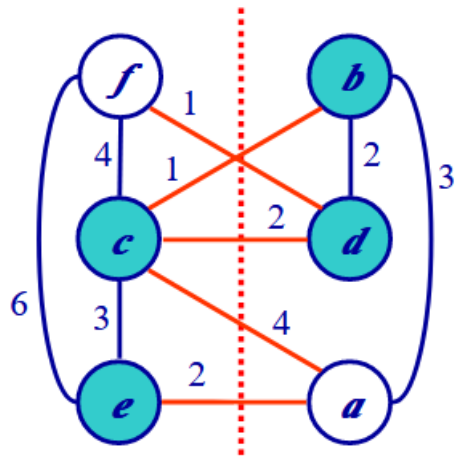
cut-size = 10

$G_a = -3$	$G_b = +2$
$G_c = 0$	$G_d = -1$
$G_e = +1$	$G_f = +9$

$X' = \{c, e\}$
$Y' = \{b, d\}$

Update the G-values of unlocked nodes

$$G'_c = G_c + 2c_{ca} - 2c_{cf} = 0 + 2(4 - 4) = 0$$
$$G'_e = G_e + 2c_{ea} - 2c_{ef} = 1 + 2(2 - 6) = -7$$
$$G'_b = G_b + 2c_{bf} - 2c_{ba} = 2 + 2(0 - 3) = -4$$
$$G'_d = G_d + 2c_{df} - 2c_{da} = -1 + 2(1 - 0) = 1$$



$G'_c = 0$ $G'_b = -4$
$G'_e = -7$ $G'_d = +1$

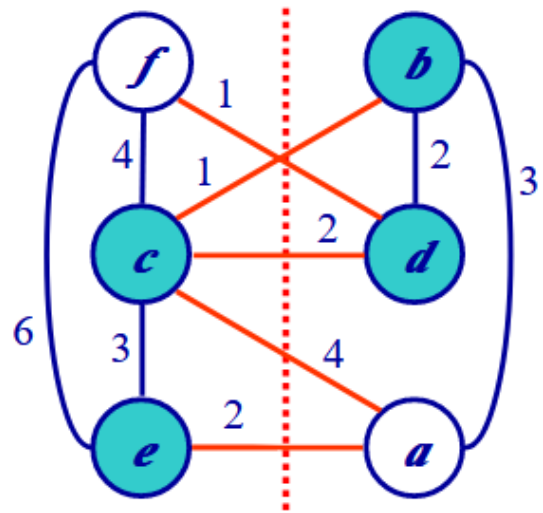
$X' = \{c, e\}$
$Y' = \{b, d\}$

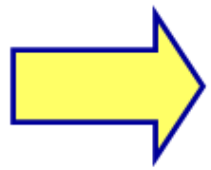
cut-size = 10

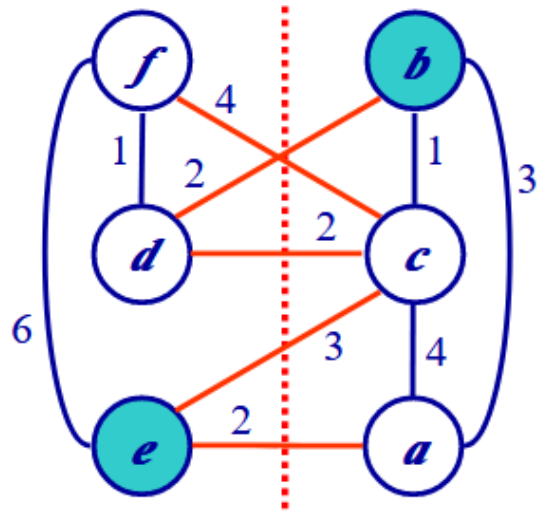
Compute the gains

$g'_{cb} = G'_c + G'_b - 2c_{cb} = 0 - 4 - 21 = -6$
$g'_{cd} = G'_c + G'_d - 2c_{cd} = 0 + 1 - 22 = -3$
$g'_{eb} = G'_e + G'_b - 2c_{eb} = -7 - 4 - 20 = -11$
$g'_{ed} = G'_e + G'_d - 2c_{ed} = -7 + 1 - 20 = -6$

Pair with maximum gain (can also be neative)







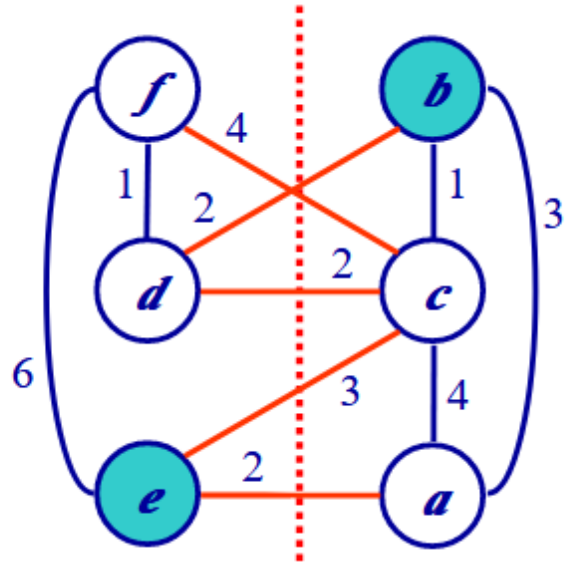
cut-size = 10

cut-size = 10 – (–3) = 13

Exchange nodes *c* and *d*

Then lock up nodes *c* and *d*

$$g'_{cd} = G'_c + G'_d - 2c_{cd} = 0 + 1 - 22 = -3$$



cut-size = 13

$G'_c = 0$ $G'_b = -4$
$G'_e = -7$ $G'_d = +1$

$X'' = \{e\}$
$Y'' = \{b\}$

Update the G-values of unlocked nodes

$G''_e = G'_e + 2c_{ed} - 2c_{ec} = -7 + 2(0 - 3) = -1$
$G''_b = G'_b + 2c_{bd} - 2c_{bc} = -4 + 2(2 - 1) = -2$

Compute the gains

Pair with max. gain is (e, b)

$g''_{eb} = G''_e + G''_b - 2c_{eb} = -1 - 2 - 20 = -3$

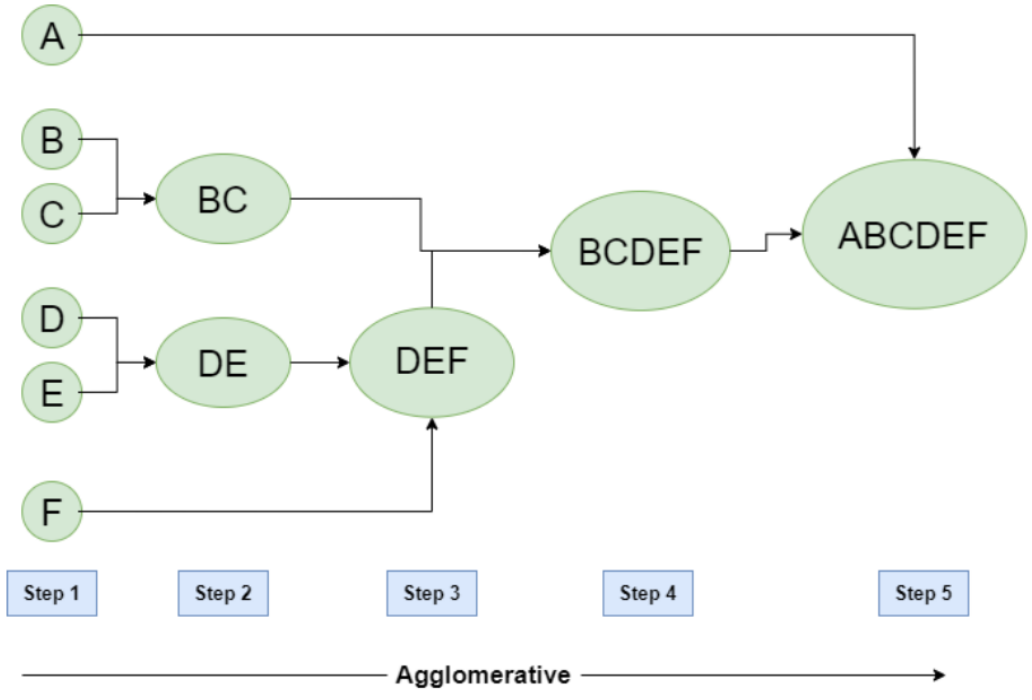
- **Summary of the Gains..**
  - $g = +6$
  - $g + g' = +6 - 3 = +3$
  - $g + g' + g'' = +6 - 3 - 3 = 0$
- Maximum Gain = $g = +6$
- Exchange only nodes *a* and *f*.
- End of 1 pass.
- ***Repeat the Kernighan-Lin***

Agglomerative/Divisive Algorithms

- Agglomerative algorithms begin with each node in the social network in its own community, and at each step merge communities that are deemed to be sufficiently similar, continuing until either the desired number of communities is obtained or the remaining communities are found to be too dissimilar to merge any further.
- Divisive algorithms operate in reverse; they begin with the entire network as one community, and at each step, choose a certain community and split it into two parts.
- Both kinds of hierarchical clustering algorithms often output a *dendrogram* which is a binary tree , where the leaves are nodes of the network, and each internal node is a community.
- In the case of divisive algorithms, a parent-child relationship indicates that the community represented by the parent node was divided to obtain the communities represented by the child nodes.
- In the case of agglomerative algorithms, a parent-child relationship in the dendrogram indicates that the communities represented by the child nodes were agglomerated (or merged) to obtain the community represented by the parent node.

- Agglomerative clustering is the more common of the two paradigms. The standard
- greedy algorithm proceeds as follows:

1. Start with all data points in their own clusters
2. Repeat until only one cluster remains:
   -Find 2 clusters (C1, C2) that are most similar (that have the smallest pairwise cluster dissimilarity d(C1, C2))
   -Merge C1, C2 into one cluster

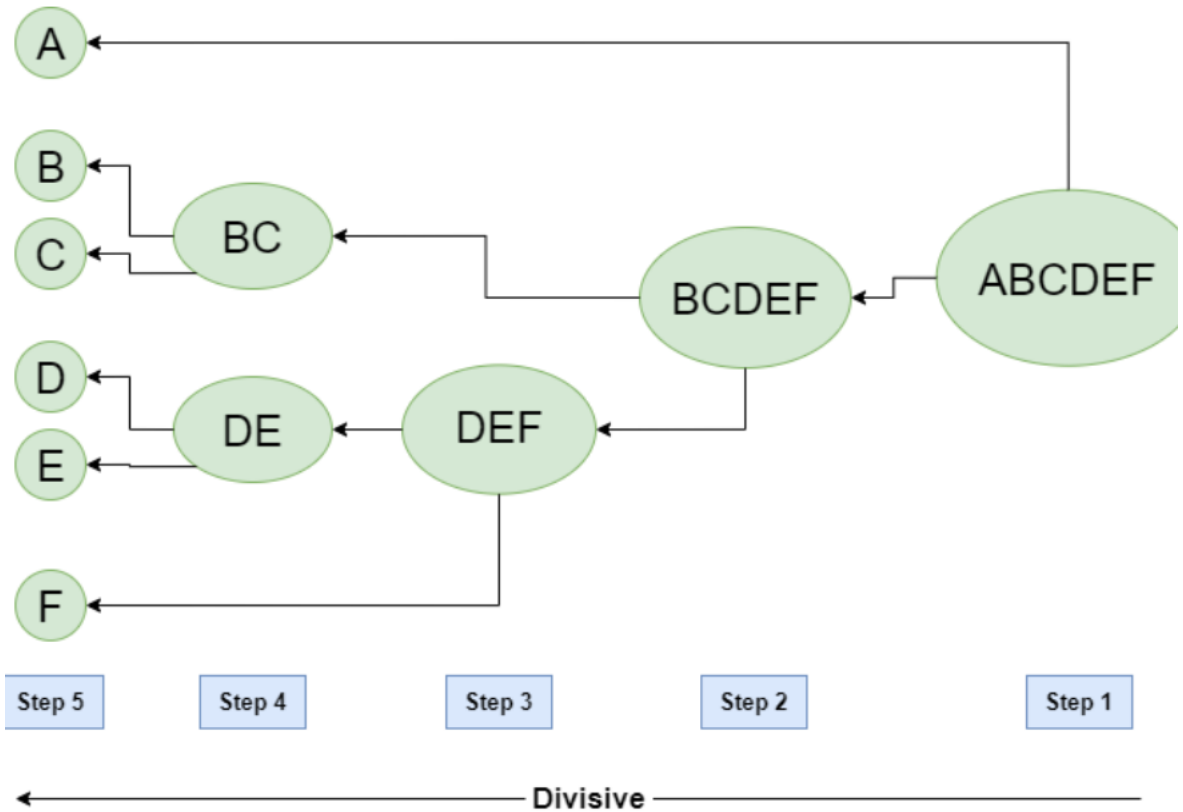


Hierarchical Agglomerative Clustering

Steps:

- Consider each alphabet as a single cluster and calculate the distance of one cluster from all the other clusters.
- In the second step, comparable clusters are merged together to form a single cluster. Let's say cluster (B) and cluster (C) are very similar to each other therefore we merge them in the second step similarly to cluster (D) and (E) and at last, we get the clusters [(A), (BC), (DE), (F)]
- We recalculate the proximity according to the algorithm and merge the two nearest clusters([(DE), (F)]) together to form new clusters as [(A), (BC), (DEF)]
- Repeating the same process; The clusters DEF and BC are comparable and merged together to form a new cluster. We're now left with clusters [(A), (BCDEF)].
- At last, the two remaining clusters are merged together to form a single cluster [(ABCDEF)].

Content:https://www.geeksforgeeks.org/ml-hierarchical-clustering-agglomerative-and-divisive-clustering/



Hierarchical Divisive clustering

Computing Distance Matrix

While merging two clusters we check the distance between two every pair of clusters and merge the pair with the least distance/most similarity.

There are different ways of defining Inter Cluster distance/similarity. Some of them are:

1. Min Distance: Find the minimum distance between any two points of the cluster.
2. Max Distance: Find the maximum distance between any two points of the cluster.
3. Group Average: Find the average distance between every two points of the clusters.

Girvan and Newman’s divisive algorithm:

- Newman and Girvan proposed a divisive algorithm for community discovery, using ideas of *edge betweenness*.
- **Edge betweenness measures are defined in a way that edges with high betweenness scores are more likely to be the edges that connect different communities.**
- That is, **inter-community edges are designed to have higher edge betweenness scores** than intra-community edges do.
- Hence, by identifying and discarding such edges with high betweenness scores, one can disconnect the social network into its constituent communities.

Girvan and Newman's divisive algorithm:

- ***Shortest path betweenness*** is one example of an edge betweenness measure: the intuitive idea here is that since there will only be a few inter-community edges, shortest paths between nodes that belong to different communities will be constrained to pass through those few inter-community edges.
- In *random-walk betweenness*, the choice of path connecting any two nodes is the result of random walk instead of geodesic as in the case of *shortest path*.
- The *current-flow betweenness* definition is motivated by the circuit theory.
- First the network is virtually transformed into a resistance network where each edge is replaced by a unit resistance and two nodes are chosen as unit current source and sink.
- Then the betweenness of each edge is computed as the sum of absolute values of the currents flowing on it with all possible selections of node pairs.

- General idea: “If two communities are joined by only a few inter-community edges, then all paths through the network from vertices in one community to vertices in the other must pass along one of those few edges.”
- Community structure can be revealed by removing edges that with high betweenness
- Algorithm is based on a divisive clustering idea

➢ Betweenness of an edge e is defined as the number of shortest paths that include e

➢ Edges that lie between communities tend to have high betweenness

$$B(e) = \frac{\text{Shortest path including } e}{\text{Number of total shortest paths}}$$

The general form of their algorithms is as follows:

1 Calculate betweenness score for all edges in the network using any measure.

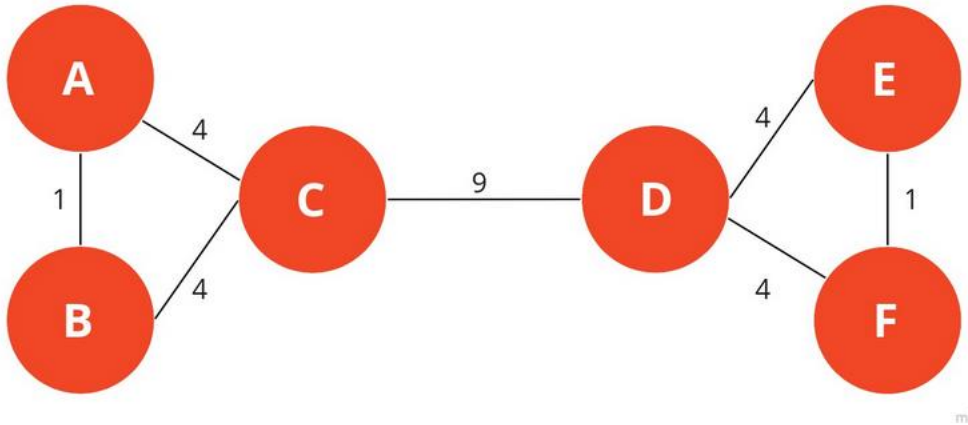
2 Find the edge with the highest score and remove it from the network.

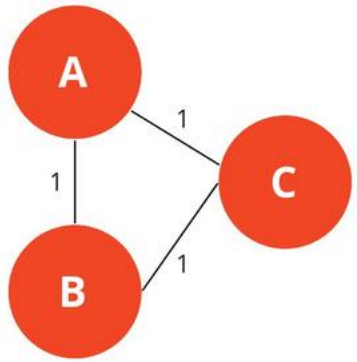
3 Recalculate betweenness for all remaining edges.

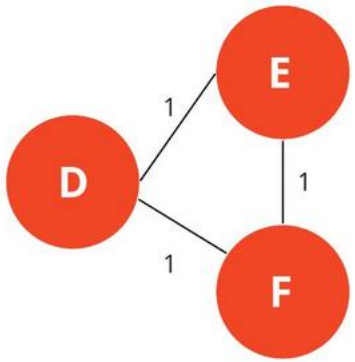
4 Repeat from step 2.

- The above procedure is continued until a sufficiently small number of communities are obtained, and a hierarchical nesting of the communities is also obtained as a natural by-product.

- Convergence criteria can be
  - No more edges
  - Desired modularity

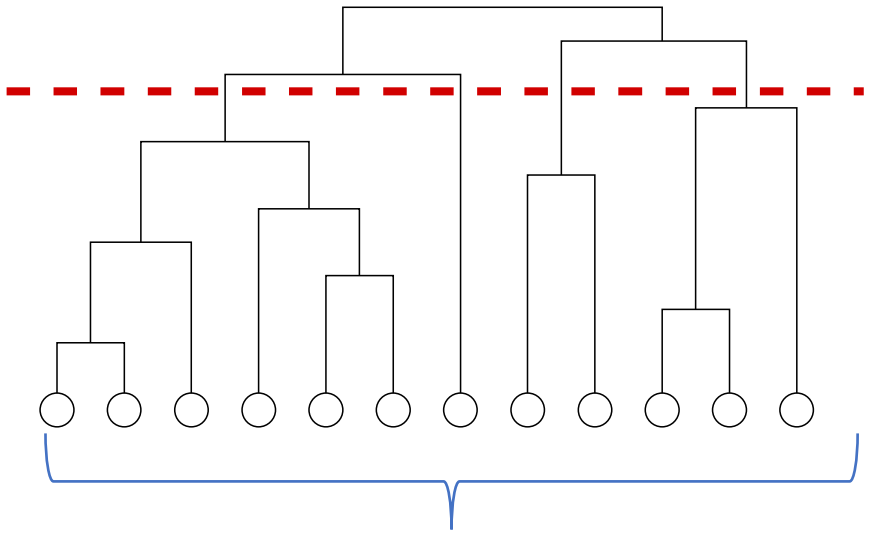






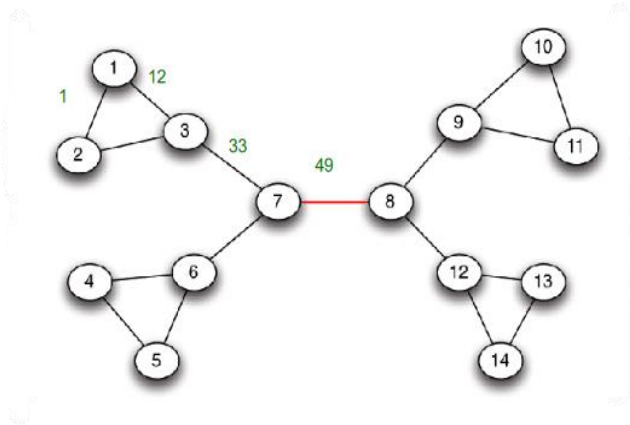
Girvan-Newman algorithm as a hierarchical clustering algorithm

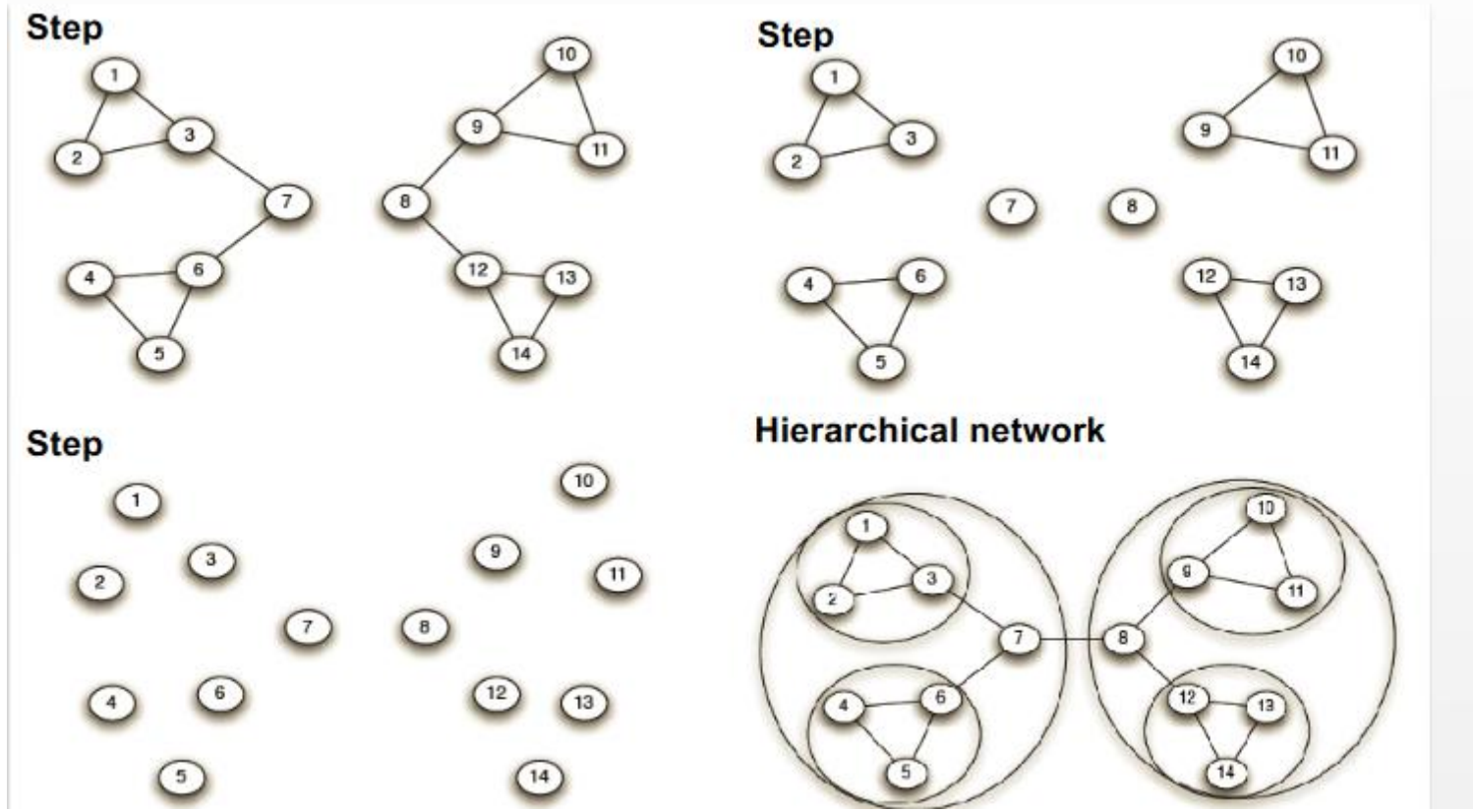
- One can view this algorithm as a top-down (divisive) hierarchical clustering algorithm
- The root of the dendrogram groups all nodes into one community
- Each branch of the tree represents the order of splitting the network as edges are removed



Graph vertices

The main disadvantage of this approach is the high computational cost: simply computing the betweenness for all edges takes O(|V ||E|) time, and the entire algorithm requires O(|V | ^{3}) time.


1
12
1
33
49
1
2
3
4
5
6
7
8
9
10
11
12
13
14


Step
Step
Step
Hierarchical network

Newman’s greedy optimization of modularity:

- Newman proposed a greedy agglomerative clustering algorithm for optimizing modularity.
- The basic idea of the algorithm is that at each stage, groups of vertices are successively merged to form larger communities such that the modularity of the resulting division of the network increases after each merge.
- At the start, each node in the network is in its own community, and at each step one chooses the two communities whose merger leads to the biggest increase in the modularity.
- We only need to consider those communities which share at least one edge, since merging communities which do not share any edges cannot result in an increase in modularity - hence this step takes O(|E|) time.
- An additional data structure which maintains the fraction of shared edges between each pair of communities in the current partition is also maintained, and updating this data structure takes worst-case O(|V |) time.
- There are a total of $|V| - 1$ iterations (i.e. mergers), hence the algorithm requires $O(|V|^2)$ time.

Spectral Algorithm

- Spectral methods generally refer to algorithms that **assign nodes to communities based on the eigenvectors of matrices,** such as the adjacency matrix of the network itself or other related matrices.
- The top k eigenvectors define an embedding of the nodes of the network as points in a k-dimensional space, and one can subsequently use classical data clustering techniques such as K-means clustering to derive the final assignment of nodes to clusters.
- The main idea behind spectral clustering is that the low-dimensional representation, induced by the top eigenvectors, exposes the cluster structure in the original graph with greater clarity.

- The main matrix that is used in spectral clustering applications is the Laplacian matrix L.
- If A is the adjacency matrix of the network, and D is the diagonal matrix with the degrees of the nodes along the diagonal, then the unnormalized Laplacian L is given as L = D −A.

  - Graph $G=\{V, E\}$ where V is the vertex set and E is the edge set
  - D: Degree matrix of a graph
  - A: Adjacency matrix of a graph
  - L: Graph Laplacian

  - The unnormalized graph Laplacian is |V|X|V| matrix

$$L = D - A$$

- The Laplacian (or the normalized Laplacian) $\boldsymbol{L}$ is given by

$$\boldsymbol{L} = D^{-1/2}(D - A)D^{-1/2}$$

$$= I - D^{-1/2} A D^{-1/2}.$$

It can be verified that both L and $\boldsymbol{L}$ are symmetric and positive definite, and therefore have real and positive eigenvalues.

- The eigenvector corresponding to the smallest non-zero eigenvalue of L is known as the Fiedler vector, and usually forms the basis for bi-partitioning the graph.

Disadvantages:

- The main disadvantage of spectral algorithms lies in their computational complexity.
- Most modern implementations for eigenvector computation use iterative algorithms such as the Lanczos algorithm, where at each stage a series of matrix vector multiplications are performed to obtain successive approximations to the eigenvector currently being computed.
- The complexity for computing the top eigenvector is O(kM(m)), where k is the number of matrix-vector multiplications and M(m) is the complexity of each such multiplication, dependent primarily on the number of non-zeros m in the matrix.
- k depends on the specific properties of the matrix at hand - such as the spectral gap i.e. the difference between the current eigenvalue and the next eigenvalue;
- the smaller this gap, the more number of matrix-vector multiplications are required for convergence.
- In practice, spectral clustering is hard to scale up to networks with more than tens of thousands of vertices without employing parallel algorithms.

Multi-level Graph Partitioning

- The main idea here is to shrink or coarsen the input graph successively so as to obtain a small graph, partition this small graph and then successively project this partition back up to the original graph, refining the partition at each step along the way.
- Multi-level graph partitioning methods include multi-level spectral clustering, Metis (which optimizes the KL objective function), Graclus (which optimizes normalized cuts and other weighted cuts.

- The main components of a multi-level graph partitioning strategy are:
- **Coarsening:**

✓ The goal here is to produce a smaller graph that is similar to the original graph.

✓This step may be applied repeatedly to obtain a graph that is small enough to be partitioned quickly and with high quality.

✓ A popular coarsening strategy is to first construct a matching on the graph, where a matching is defined as a set of edges no two of which are incident on the same vertex.

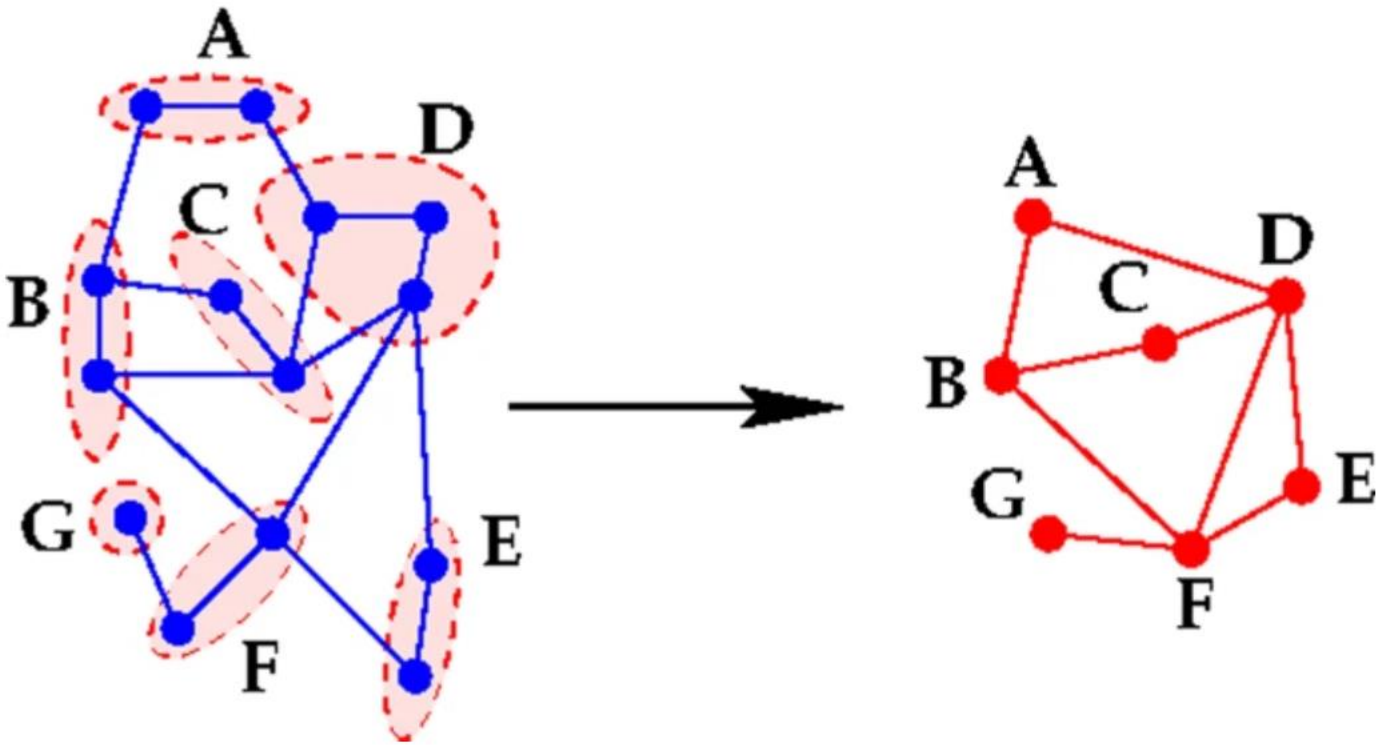
✓For each edge in the matching, the vertices at the ends of the edge are collapsed together and are represented by a single node in the coarsened graph.

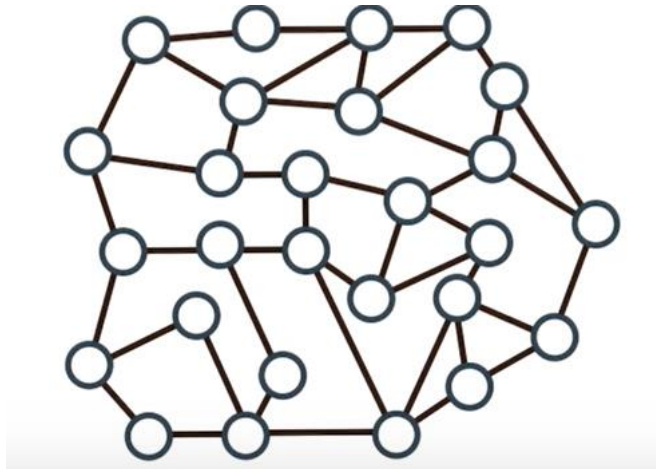
- **2 Initial partitioning:**
  - ✓In this step, a partitioning of the coarsest graph is performed.
  - ✓Since the graph at this stage is small enough, one may use strategies like spectral partitioning which are slow but are known to give high quality partitions.
- **3 Uncoarsening:**
  - ✓ In this phase, the partition on the current graph is used to initialize a partition on the finer (bigger) graph.
  - ✓The finer connectivity structure of the graph revealed by the uncoarsening is used to refine the partition, usually by performing local search. This step is continued until we arrive at the original input graph.

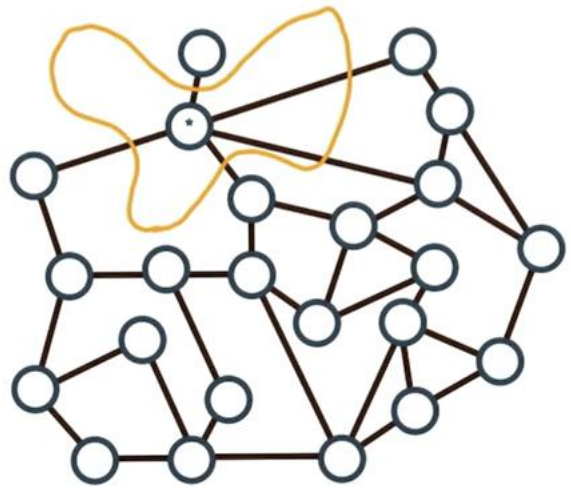
At a finer level, Metis uses a variant of the KL algorithm in its uncoarsening phase to refine the partition obtained from previous steps.

Graclus, uses weighted kernel k-means for refining the partition.

Coarsening:







- If we want to partition G(N,E), but it is too big to do efficiently, what can we do?
  - 1) Replace G(N,E) by a coarse approximation $G_c(N_c,E_c)$, and partition G_c instead
  - 2) Use partition of G_c to get a rough partitioning of G, and then iteratively improve it
- What if G_c still too big?
  - Apply same idea recursively

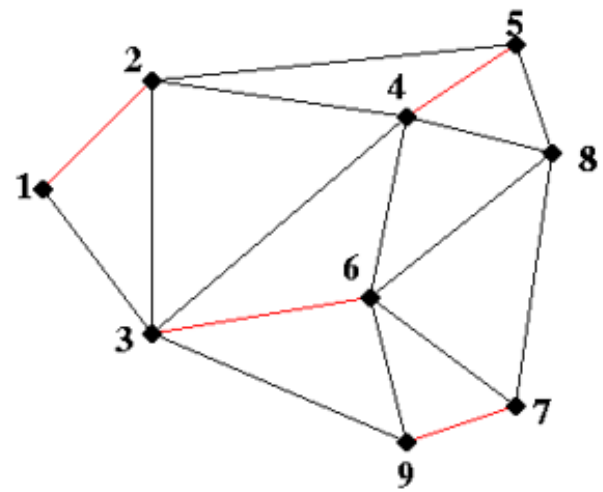
Multilevel Kernighan-Lin

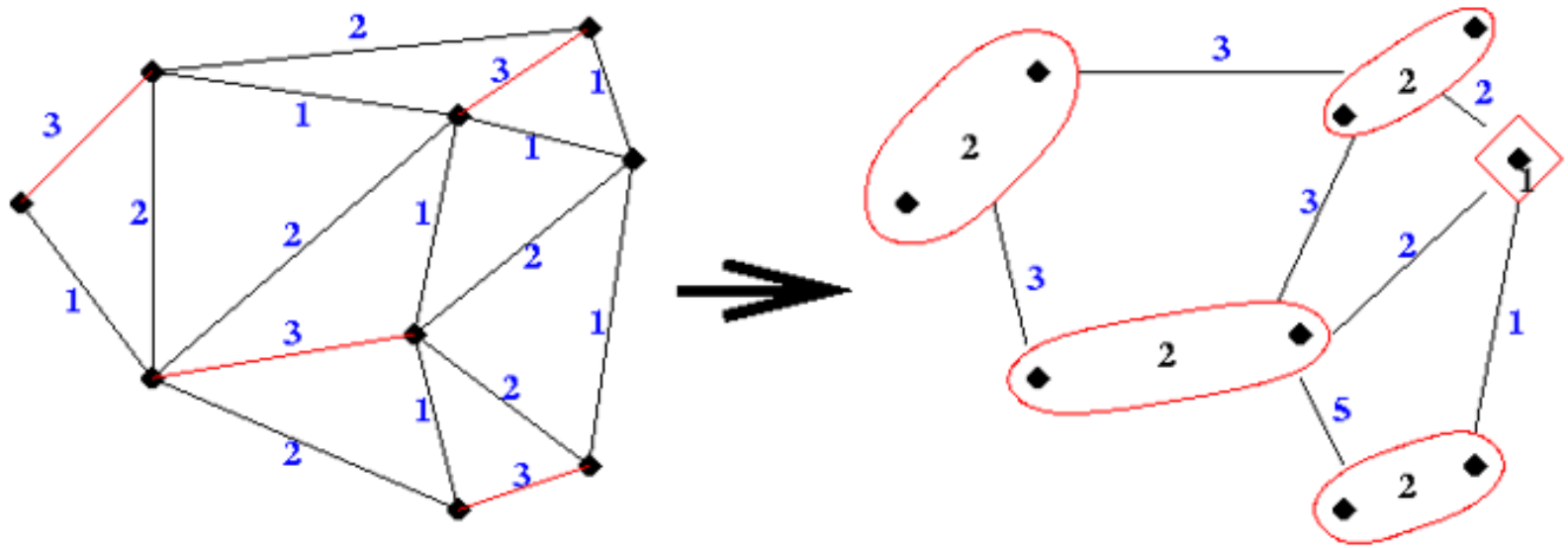
- Coarsen graph and expand partition using maximal matchings
- Improve partition using Kernighan-Lin

Maximal Matching:

- *Definition*: A matching of a graph G(N,E) is a subset E_m of E such that no two edges in E_m share an endpoint
- *Definition:* A maximal matching of a graph G(N,E) is a matching E_m to which no more edges can be added and remain a matching
- A simple greedy algorithm computes a maximal matching

```
Em = { }
for i = 1 to |E|
    if i-th edge shares no endpoint with edges in Em,
        add i-th edge to Em
end
```


1
2
3
4
5
6
7
8
9



$G = (N, E)$

E_m is shown in red

Edge weights shown in blue

Node weights are all one

$G_c = (N_c , E_c)$

N_c is shown in red

Edge weights shown in blue

Node weights shown in black

1) Construct a maximal matching E_m of G(N,E)

for all edges e=(j,k) in E_m **2) collapse matched nodes into a single one**

Put node n(e) in N_c

W(n(e)) = W(j) + W(k)

for all nodes n in N not incident on an edge in E_m **3) add unmatched nodes**

Put n in N_c … do not change W(n)

… Now each node r in N is "inside" a unique node n(r) in N_c

… **4) Connect two nodes in Nc if nodes inside them are connected in E**

for all edges e=(j,k) in E_m

for each other edge e'=(j,r) or (k,r) in E

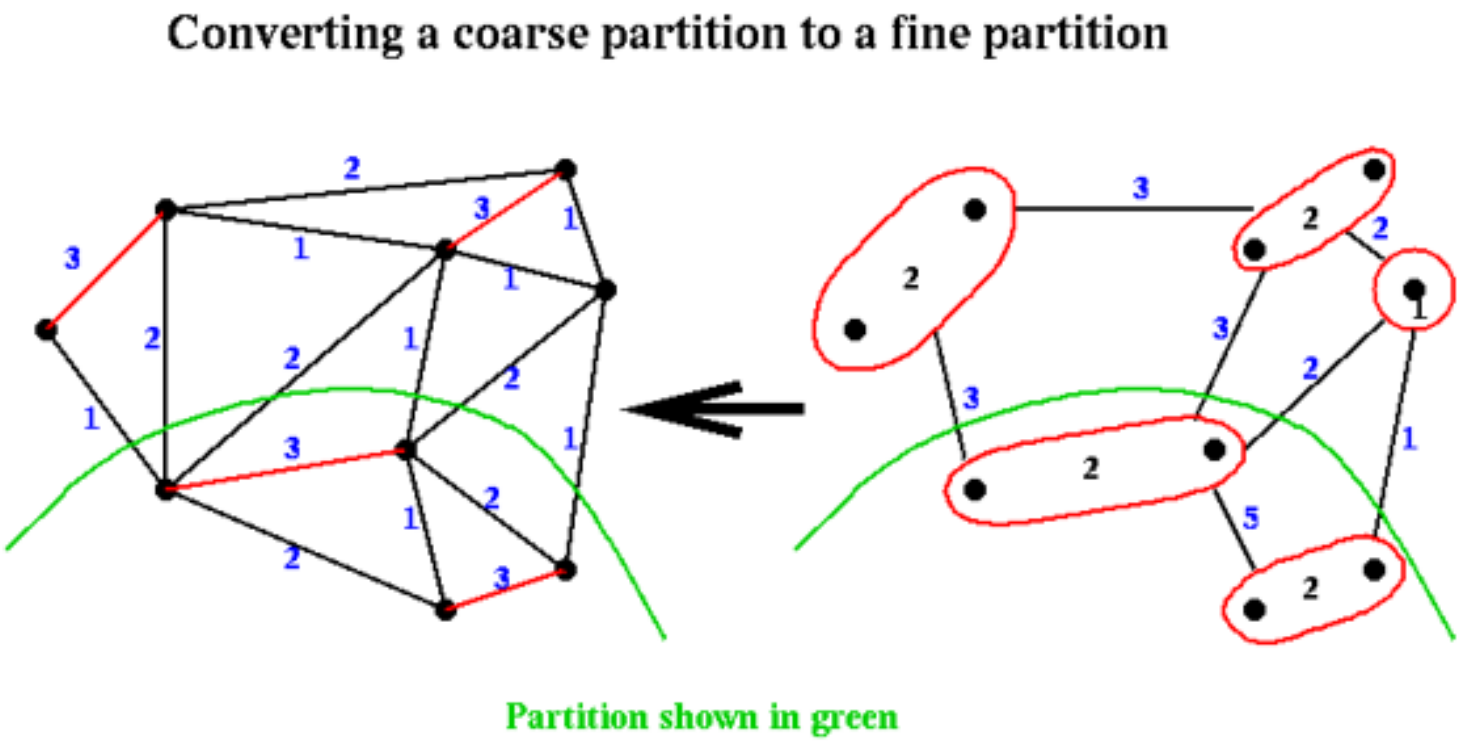
Put edge ee = (n(e),n(r)) in E_c

W(ee) = W(e')

If there are multiple edges connecting two nodes in N_c, collapse them,

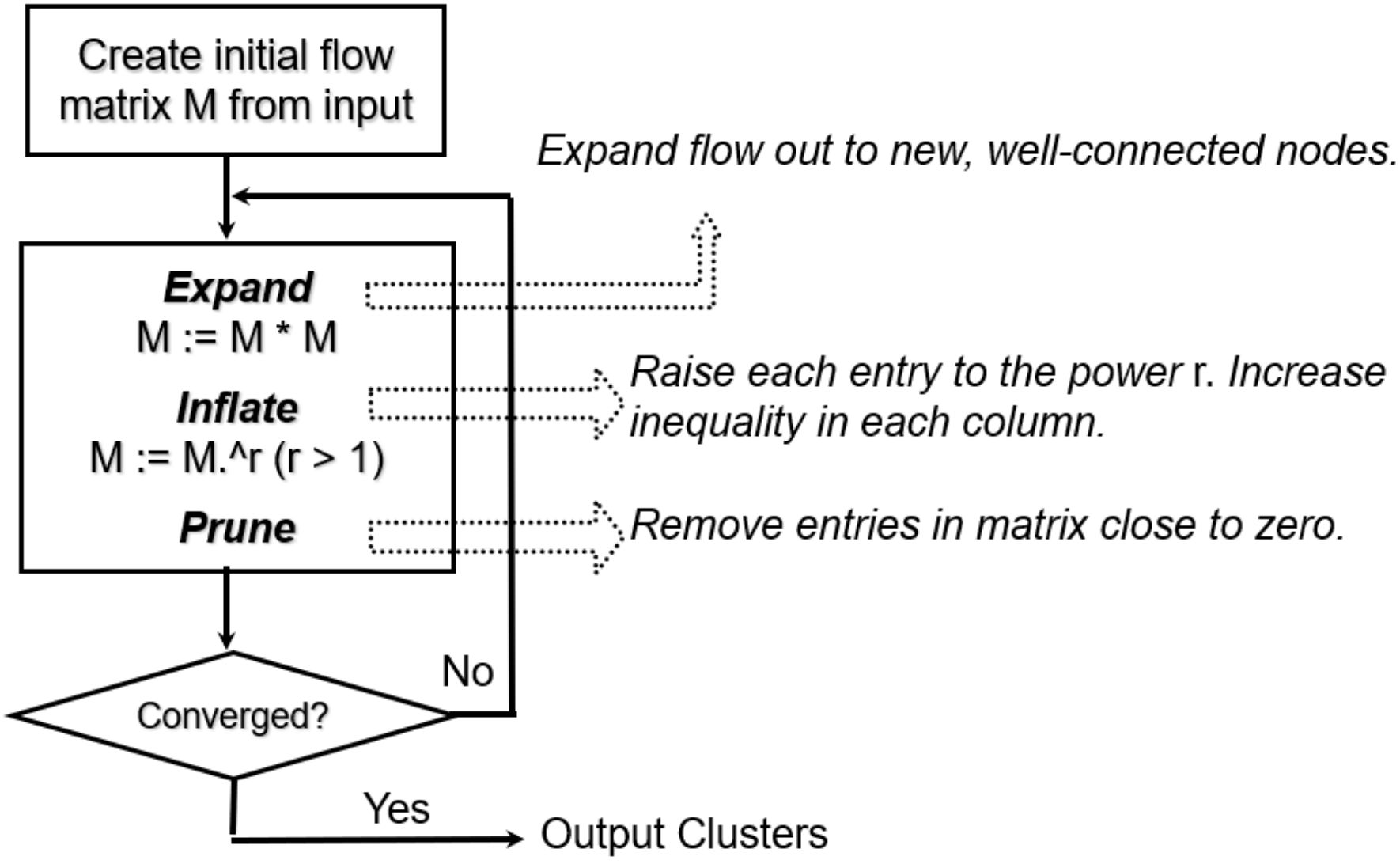
adding edge weights

Expanding a partition of G_c to a partition of G

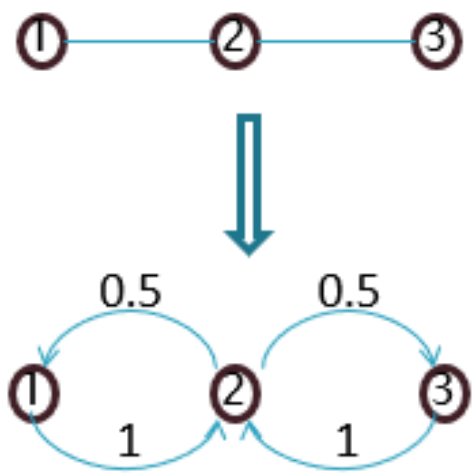


Markov Clustering

- Stijn van Dongen's Markov Clustering algorithm (MCL) clusters graphs via manipulation of the **stochastic matrix or transition probability matrix** corresponding to the graph.
- The transition probability between two nodes is also referred to as **stochastic flow.**
- The MCL process consists of two operations on stochastic matrices, **Expand and Inflate.**
- Expand(M) is simply $M * M$, and Inflate(M,r) raises each entry in the matrix M to the inflation parameter r (> 1, and typically set to 2) followed by re-normalizing the columns to sum to 1.
- These two operators are applied in alternation iteratively until convergence, starting with the initial transition probability matrix.
- The expand step spreads the stochastic flow out of a vertex to potentially new vertices and also enhances the stochastic flow to those vertices which are reachable by multiple paths.
- The inflation step introduces a non-linearity into the process, with the purpose of strengthening intra-cluster stochastic flow and weakening inter-cluster stochastic flow.


Create initial flow matrix M from input
Expand
M := M * M
Inflate
M := M.^r (r > 1)
Prune
Converged?
No
Yes
Output Clusters
Expand flow out to new, well-connected nodes.
Raise each entry to the power r. Increase inequality in each column.
Remove entries in matrix close to zero.

- **Flow**: Transition probability from a node to another node.
- **Flow matrix**: Matrix with the flows among all nodes; i^{th} column represents flows out of i^{th} node. Each column sums to 1.



Flow Matrix

	1	2	3
1	0	0.5	0
2	1.0	0	1.0
3	0	0.5	0

Other Approaches

- **Local Graph Clustering:**
- A local algorithm is one that finds a solution containing or near a given vertex (or vertices) without looking at the whole graph.
- Local algorithms are interesting in the context of large graphs since their time complexity depends on the size of the solution rather than the size of the graph to a large extent.
- However, if the clusters need to cover the whole graph, then it is not possible to be independent of the size of the graph.
- The main intuition is that random walks simulated from inside a group of internally well-connected nodes will not mix well enough/soon enough, as the cluster boundary acts a bottleneck that prevents the probability from seeping out of the cluster easily.

- **Flow-Based Post-Processing for Improving Community Detection**:

  ✓Algorithms for computing the maximum flow in flow networks can be used to post-process or improve existing partitions of the graph.

- **Community Discovery via Shingling:**
- Broder et al. introduced the idea of clustering web documents through the use of shingles and fingerprints.
- For example, a length-s shingle of a graph node contains s outgoing links of the node.
- A sketch is a constant-size subset of all shingles with a specific length, with the remarkable property that the similarity between sets of two objects' sketches approximates the similarity between the objects.
- Gibson et al. attempt to extract dense communities from large-scale graphs via a recursive application of shingling.