SCALABLE METHODS FOR REPRESENTING LARGE SCALE GRAPHS

GROUP 4

16CS10058: Lovish Chopra

16CS30044: Sarthak Chakraborty

16ME10069: Partho Roychoudhury

Mentor: Ayan Kumar Bhowmick

2,410,000,000+

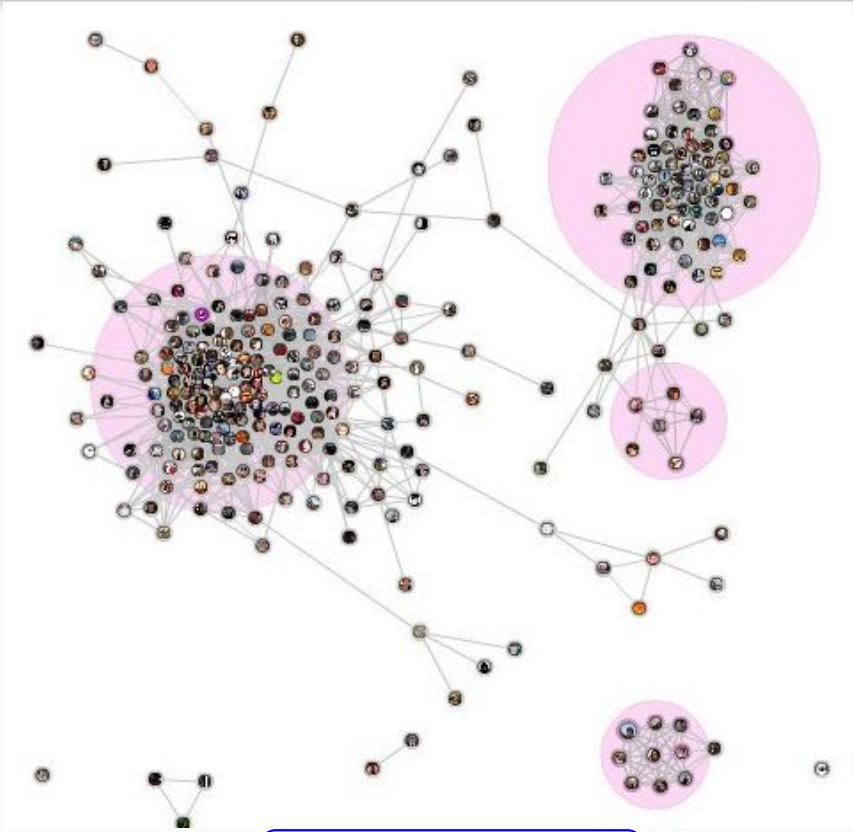
monthly active accounts on facebook

Whoa! That’s a big number!

338

average number of friends per facebook account

Graphs in Action - Link Recommendation, Node Classification



Search Recommendation

Friend Suggestion

Product Recommendation

Graph Algorithms

Require |V| x |V| size adjacency matrix for representing a graph G(V,E)

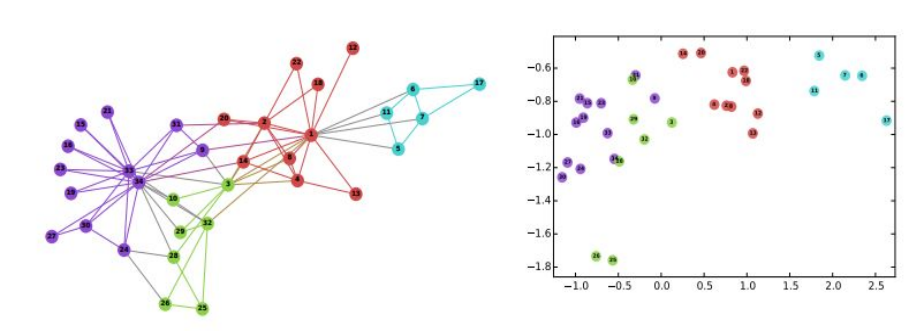
Issues: Scalable?

Cannot be applied on large graphs as the number of nodes in the graph can be extremely high. Also, adjacency matrix may be a sparse matrix.

What we need?

Low-dimensional representation of the graph can be useful: Embed each node with a d-dimensional vector (d << |V|), yet preserving the higher order structural features of the graph

DeepWalk



2D representation of Karate Graph

- Initialize a random Embedding vector
- For each node in G(V,E), generate sequence of reachable nodes using Random Walk algorithm for t timesteps
- Apply SkipGram algorithm on the node to improve the embedding vector

Similar Algorithms: Node2Vec, GraRep, NetMF

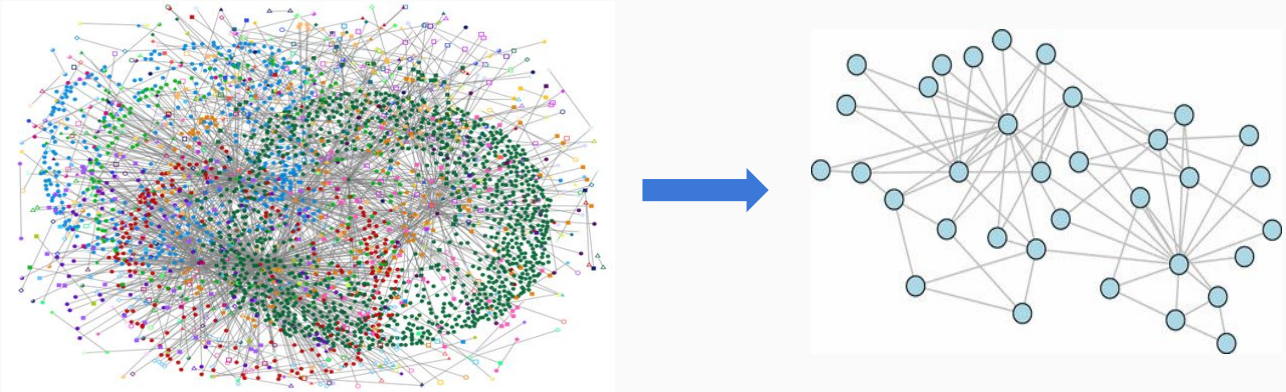
Image Source: DeepWalk Online Learning of Social Representations

Where do these algorithms fail?

- These algorithms focus on local structure and thus, implicitly ignore long-distance global relationships.
- Most of them rely on a non-convex optimization goal solved using stochastic gradient descent which can get stuck in a local minima.
- If |V| is very large, we need to generate random walks for each node and hence can take a lot of time to converge

OBJECTIVE

To develop a fast and efficient network embedding framework for learning representations of nodes in large-scale graphs with nodes and edges in the order of billions, which preserves the higher order structure of the graph



Previous Works

- **HARP:**
  Hierarchical Representation Learning for Networks

- **MILE:**
  A Multi-Level Framework for Scalable Graph Embedding

Basic Steps: Hierarchical Approach

- Repetitively coarse the graph by combining structurally similar nodes to form a supernode
- Apply Node embedding algorithms on the coarsened graph
- Prolongate the embeddings to the original graph

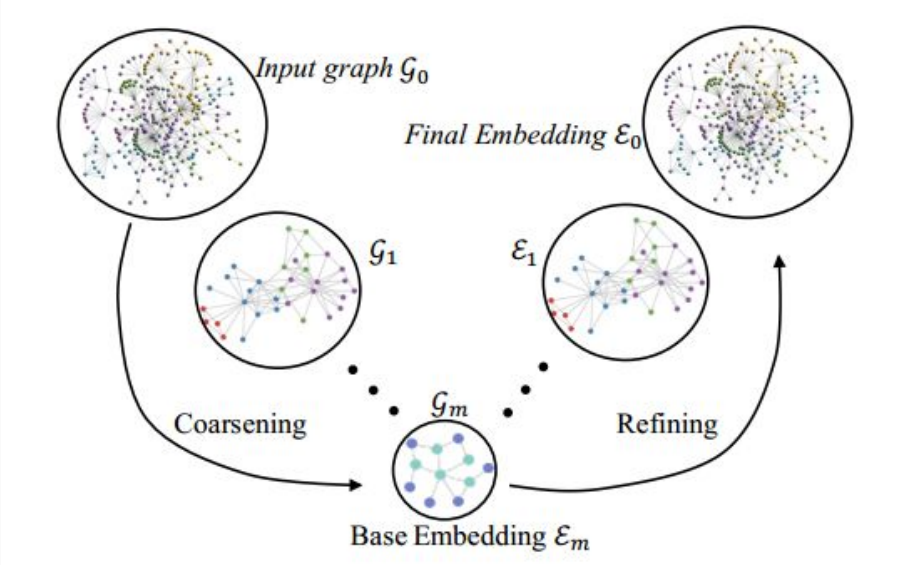
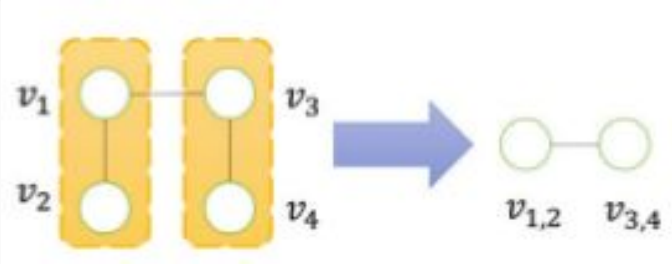


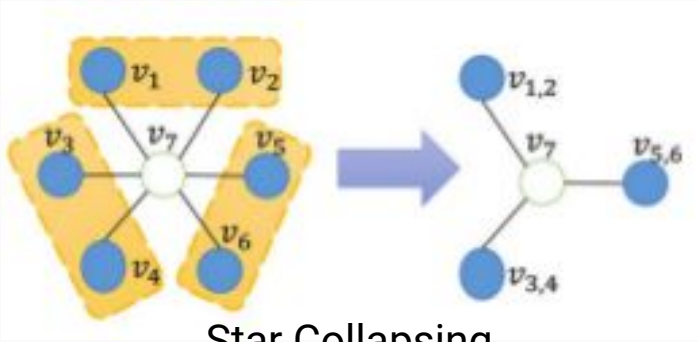
Image Source: MILE: A Multi-Level Framework for Scalable Graph Embedding

HARP

- **Graph Coarsening:** Star Collapsing followed by Edge Collapsing iteratively
- **Graph Embedding on Coarsest Graph:** Get high quality representation on the coarsest graph. Mainly using DeepWalk
- **Prolongation and Refinement:** Prolong the graph embeddings backwards to the coarsed graphs



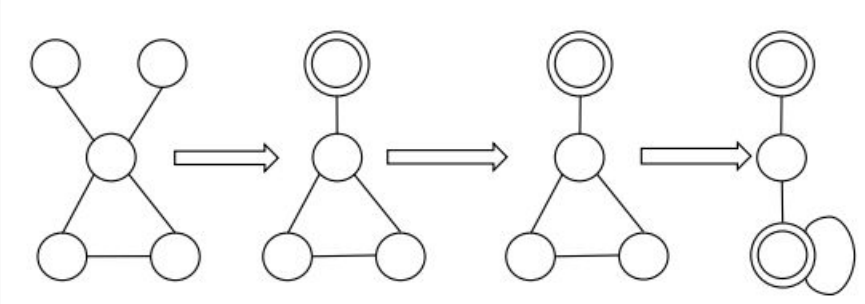
Edge Collapsing



Star Collapsing

MILE

- **Graph Coarsening:** Hybrid Matching Method (Structurally Equivalence Matching, Normally Heavy Edge Matching)
- **Base Embeddings:** DeepWalk, Node2Vec, GraRep, NetMF
- **Embedding Refinement:** Uses Graph Convolution Network to project embeddings to original graph



Shortcomings of HARP and MILE

HARP:

- Uses learned embedding from previous level as initial embedding for next level, thus resulting in computation overhead.
- Cannot be easily extended to different embedding techniques

MILE:

- Complex GCN model is used for refinement which is not easily interpretable and takes time to train
- Experiments show that the algorithm is not fast on large datasets

What to aim for?

- A graph coarsening algorithm that can coalesce nodes based on its content
- Fast, efficient and simple embedding refinement method
- Method should be scalable and give accurate results on large graph datasets

THANK YOU!

SCALABLE METHODS FOR REPRESENTING LARGE SCALE GRAPHS

Group 4

27/09/2019

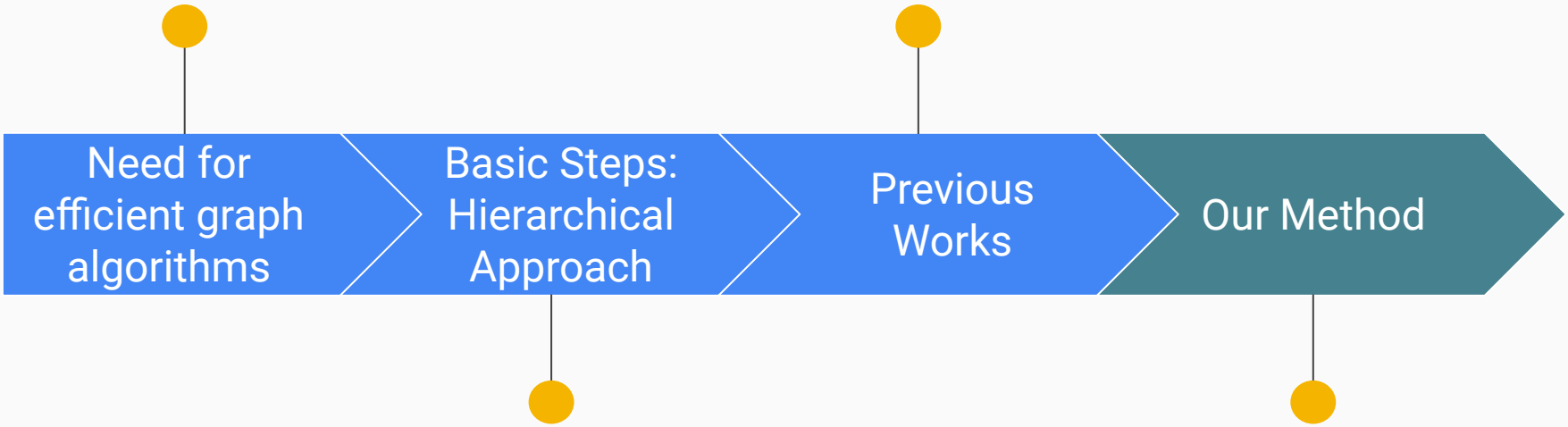
16CS10058: Lovish Chopra

16CS30044: Sarthak Chakraborty

16ME10069: Partho Roychoudhury

Mentor: Ayan Kumar Bhowmick

Progress So Far...



LouvainNE

- Based on the notion of community structure present in real nodes
- Nodes in a single community are densely connected among themselves and hence, are of similar type. Also, repeatedly identifying smaller sub-communities will help to preserve the local proximity between nodes, capturing the neighbourhood property of node-pairs. We are using Louvain for identifying the sub-communities.

Notations and Basic Problem Definition

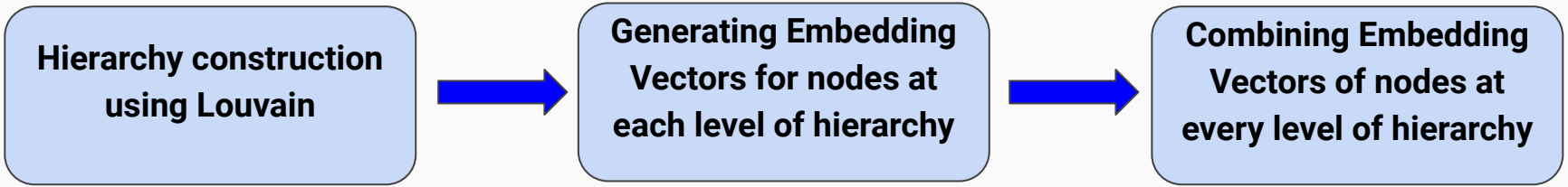
Graph: G (V, E) is given, which is essentially a very large graph with a large number of nodes

Objective: To find a mapping function f: $v \rightarrow y_v$, where y_v is a low-dimensional embedding of the node v and dimension of $y_v < |V|$. The mapped function should preserve the network centric properties in the embedding space.

What are those network centric properties?

- Preserving the neighborhood proximity between connected node pairs
- Keeping the similar albeit disconnected nodes relatively close to each other
- Ensuring that dissimilar and disconnected nodes are placed far away
- Representing the graph topology

Steps involved in the Method



Steps involved in the Method

1. **Hierarchy construction using Louvain**: Repeatedly identifying sub-communities from the communities to generate a hierarchy of nodes.

2. **Generating Embedding Vectors**: Using standard embedding algorithms like Node2Vec and DeepWalk to generate embeddings for nodes at each level of the hierarchy.

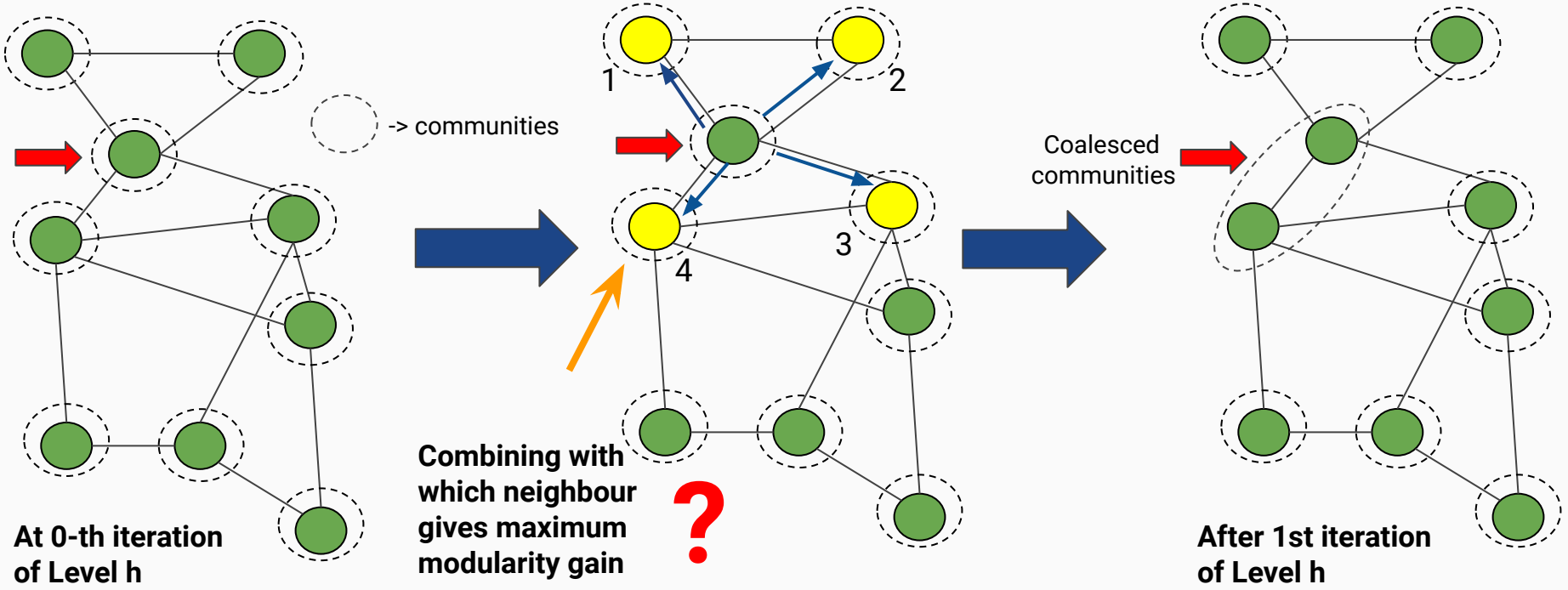
3. **Combining Embeddings at Different Levels**: Compute the final embedding vectors for each of the nodes using the generated embedding vectors.

Louvain Community Detection

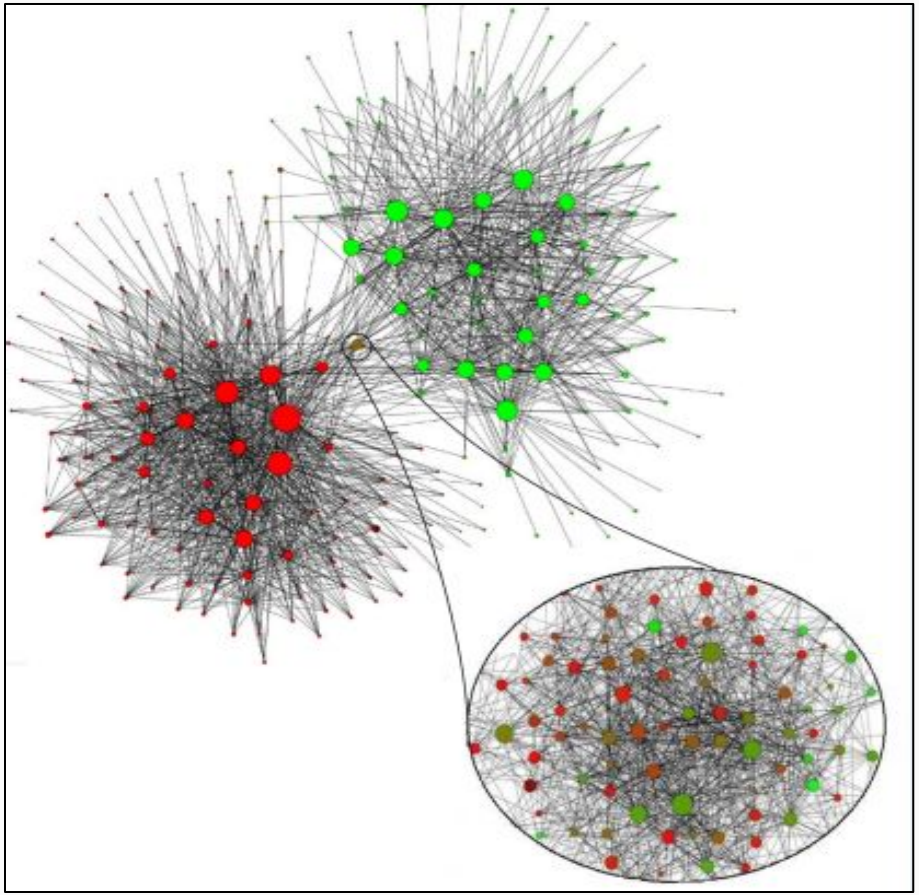
Problem Definition: A very large graph G (V, E) is given and we need to identify communities of nodes in those graphs which can be combined to form a supernode.

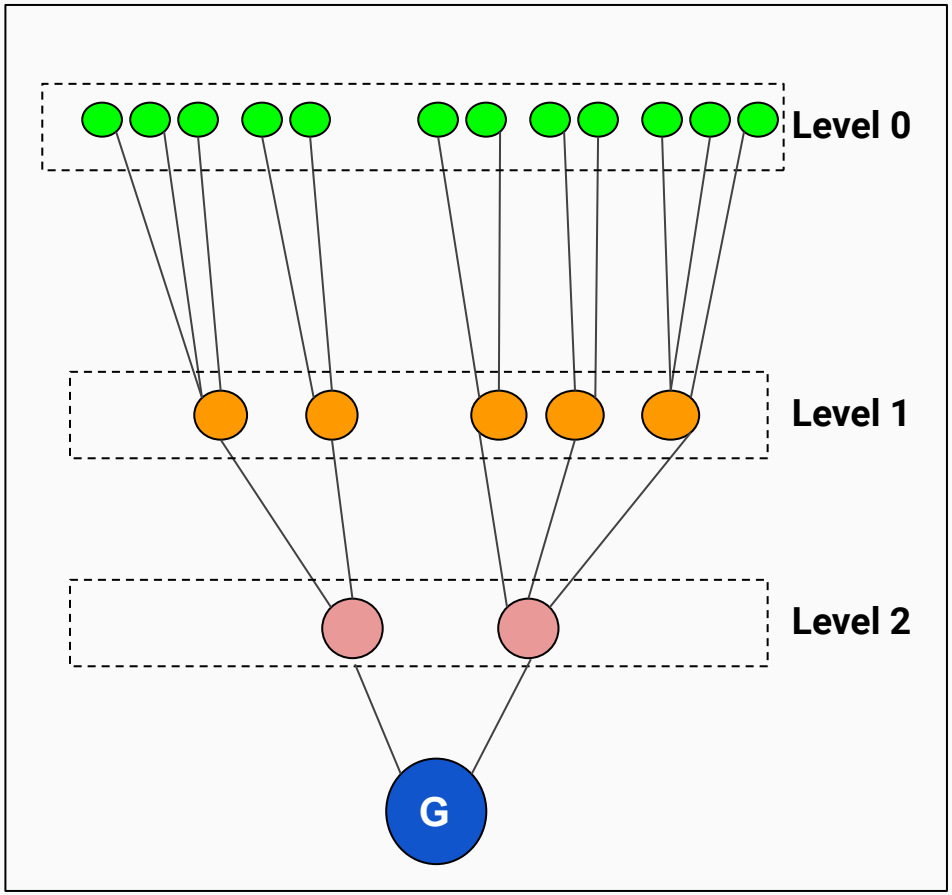
Algorithm: For each node *i* we consider the neighbours *j* of *i* and we place *i* in that *j-th* community which would give a maximum gain in network modularity. This process is repeated for every node and this gives rise to a higher level network.

The Algorithm



Community Detection Visually Explained





Advantages of Louvain Community Detection

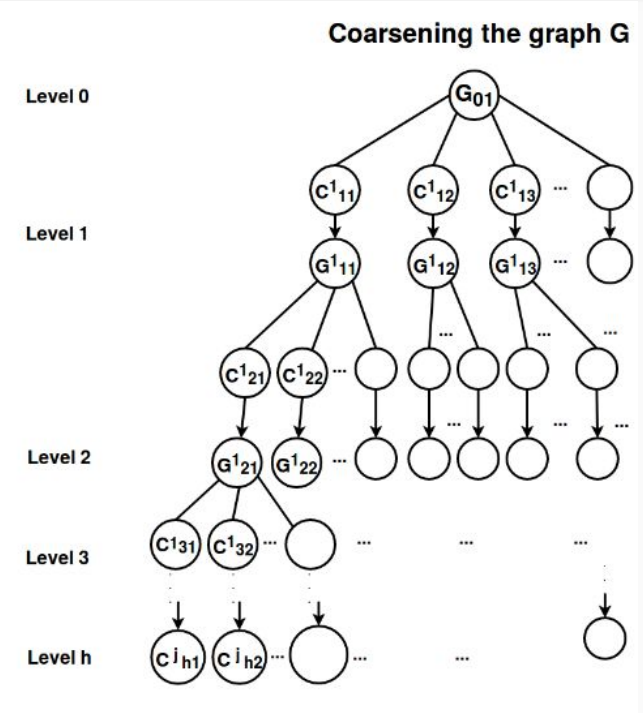
1) Its steps are intuitive and easy to implement, and the outcome is unsupervised.
2) The algorithm works extremely fast.
3) The so-called resolution limit problem of modularity optimisation has been partially dealt with in this method due to the multi-level nature of our algorithm.

Steps involved: Step 1

Hierarchy construction:

- As discussed, we are using Louvain Community Detection method to create a hierarchy of subgraphs from a given graph

- From each graph, sub-communities were found using Louvain. These sub-communities were further divided using Louvain iteratively, thus creating a tree based hierarchy of communities.



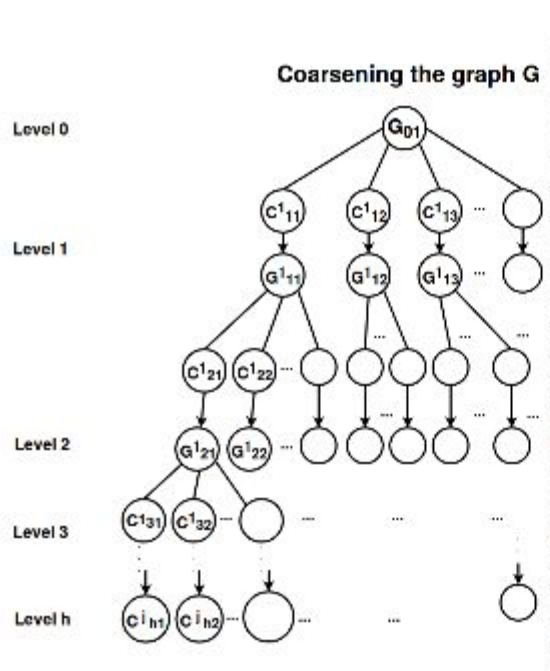
Steps involved: Step 1

Notation:

G_{01} : Complete graph as a single community

C_{ij}^{k} : j^{th} community at the i^{th} level (obtained from the k^{th} community at (i - 1)th level)

G_{ij}^{k} : subgraph corresponding to community C_{ij}^{k}



Steps involved: Step 2

Generating Embedding Vectors:

- Then, we generated embedding vectors for each of the coarsened subgraphs in the hierarchy of nodes.
- There are multiple ways of generating the embeddings and we have tried the following methods:
  - Stochastic Embedding
  - Standard Embedding:
    - DeepWalk
    - node2vec

Steps involved: Step 3

Combining embeddings for a node obtained at each level of hierarchy:

- Finally, to compute the embeddings of the nodes, we use the following method:

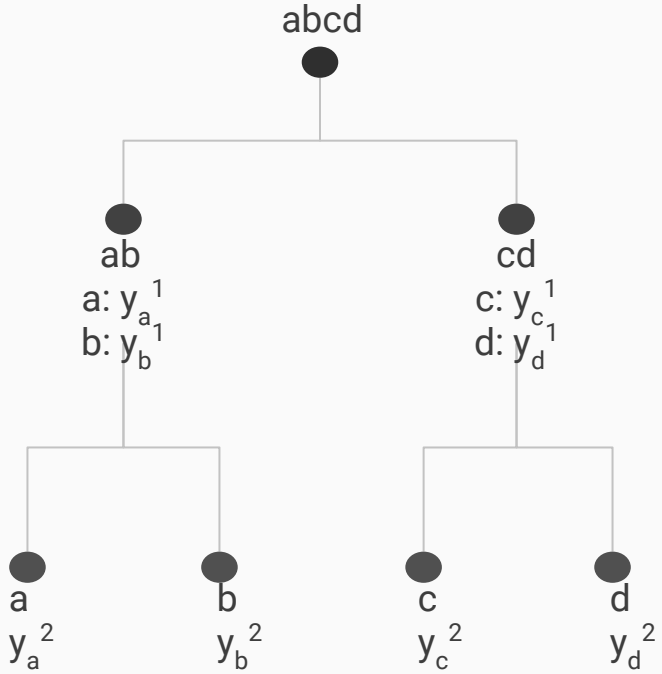
  Consider a node **v** with y_v^1, y_v^2, …, y_v^h be the h embeddings obtained for it at different levels t = 1, …, h.

  Then, the final embedding y_v is given by:

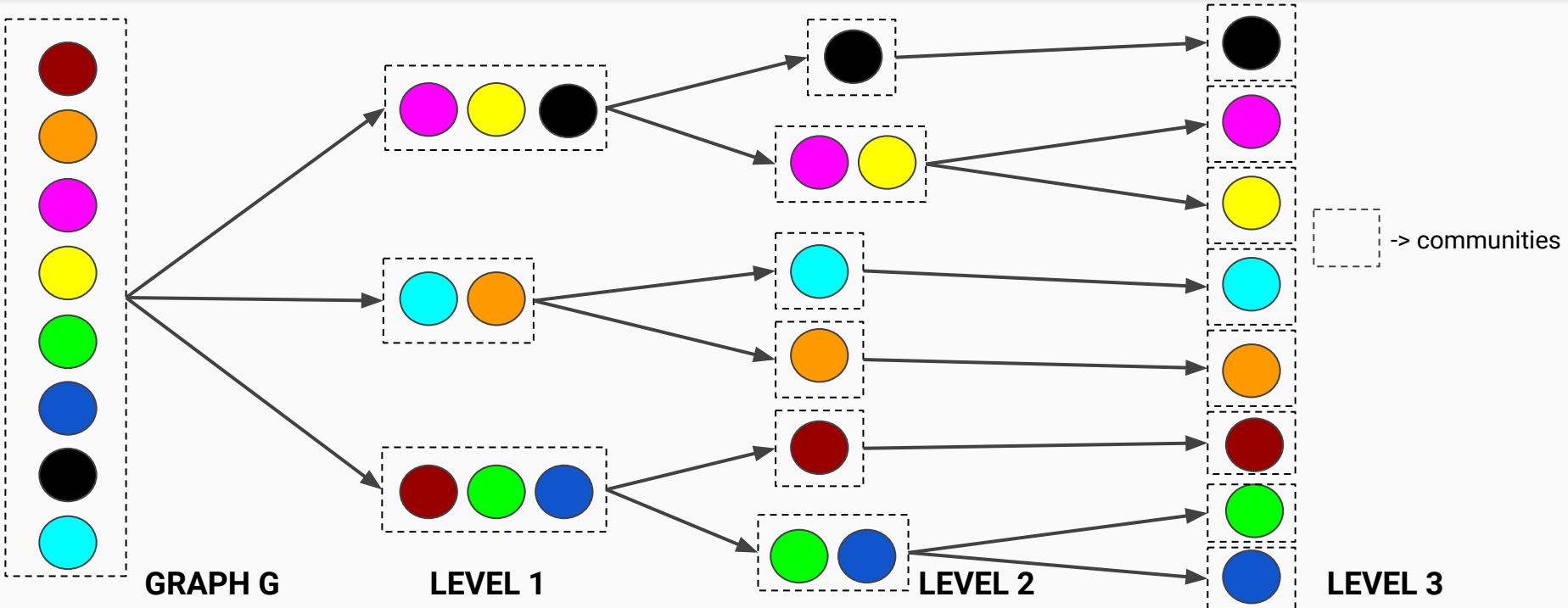
$$\mathbf{y_v} = \sum_{t=1}^{h} \alpha^{t-1} \mathbf{y_v^t}$$

Example

Consider the hierarchy tree shown beside. For ease of understanding, we have used a simple notation. At level 1, we apply Node2Vec/DeepWalk on **ab** and **cd** separately, getting embeddings for a, b and c, d respectively. At level 2, we have single nodes in each community so each of a, b, c and d get some random embeddings.

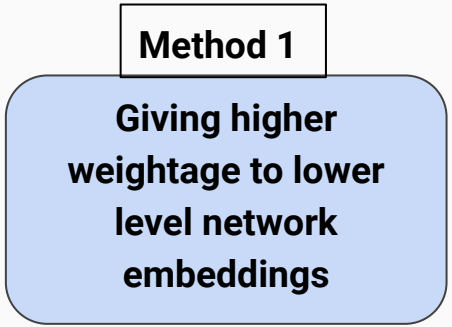


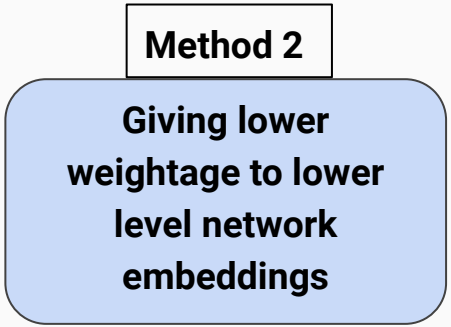
Step 3: Combining Embedding vectors



Step 3: Combining the NE of different levels

At this point, we have all the the NE of any single node at different levels of the graph hierarchy. To combine all the NE of any single node to generate its one unique NE, we have 2 options:





Step 3

Let a particular node be present in the levels starting from **1** to ***t***.
Let its NE at any level ***j*** be y_j and its combined network embedding be *y*.

Method 1

Giving higher weightage to lower level network embeddings

$$y = \alpha^1 y_1 + \alpha^2 y_2 + \alpha^3 y_3 + ... + \alpha^t y_t$$

Method 2

Giving lower weightage to lower level network embeddings

$$y = \alpha^t y_1 + \alpha^{t-1} y_2 + \alpha^{t-2} y_3 + ... + \alpha^1 y_t$$

Step 3: Combining Embedding vectors

Let us the embedding vector of node of level j be y_j. We assume a parameter α to combine the embeddings in this fashion:

$y = \alpha^1 y_1 + \alpha^2 y_2 + \alpha^3 y_3$

$y = \alpha^1 y_1 + \alpha^2 y_2 + \alpha^3 y_3$

$y = \alpha^1 y_1 + \alpha^2 y_2 + \alpha^3 y_3$

$y = \alpha^1 y_1 + \alpha^2 y_2 + \alpha^3 y_3$

$y = \alpha^1 y_1 + \alpha^2 y_2 + \alpha^3 y_3.$

$y = \alpha^1 y_1 + \alpha^2 y_2 + \alpha^3 y_3$

$y = \alpha^1 y_1 + \alpha^2 y_2 + \alpha^3 y_3$

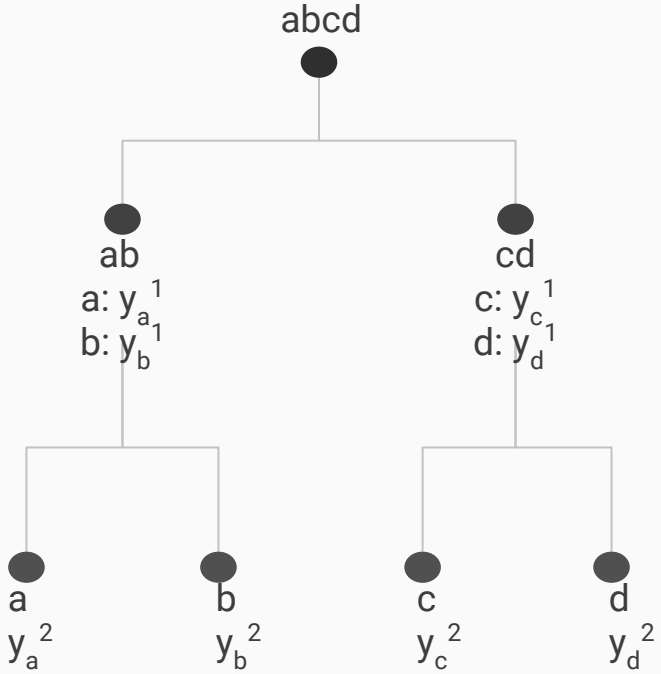
$y = \alpha^1 y_1 + \alpha^2 y_2 + \alpha^3 y_3$

Example

Now, we combine the obtained embeddings as follows:

$y_a = \alpha^1 y_a^1 + \alpha^2 y_a^2$

and similarly for node b, c and d.



Choice of α

- At each level in the hierarchy graph, we are changing the value of $\boldsymbol{\alpha}$ by a constant multiplicative factor less than 1.
- α gradually decreases as we move down the hierarchy. This is because at lower levels in the hierarchy, the nodes would automatically be placed close to each other in the embedding space, whereas at nodes in the same community at higher levels might have extremely different embeddings at lower level. Hence, embeddings at higher level are given larger weightage so that the global as well as the local structure can be preserved.
- α was varied from 0.005 to 0.9 and was plotted against the AUC scores. It was observed that AUC decreased as α was increased, hence **0.005** was chosen.

Results

Datasets Used

- **Karate Dataset:**
  - 34 nodes, 78 edges
  - Average degree: 4
- **BlogCatalog3 Dataset:**
  - 10312 nodes, 333983 edges
  - Average degree: 65

Network Reconstruction Task

A good network embedding algorithm should ensure that the embedding vectors can preserve the original network topology, such that they can be used to reconstruct the network.

In network reconstruction, we aim to predict the links in the original network by ranking the node pairs based on the similarity of their learned embedding vectors. Larger similarity means greater probability of connection between the nodes.

Network Reconstruction Task

a) **Precision @ K**: Here, we take all possible pairs of nodes and rank them based on the Euclidean distance between them. The top K node pairs are considered and their accuracy is checked based on the actual links in the graph.

b) **Area Under Curve (AUC)**: Here, we consider all connected node pairs and randomly choose equal number of disconnected node pairs in the actual graph. The similarity between the selected node pairs is computed by taking the inner product between embedding vectors. AUC is the fraction of times the connected node pairs have greater similarity than the disconnected ones.

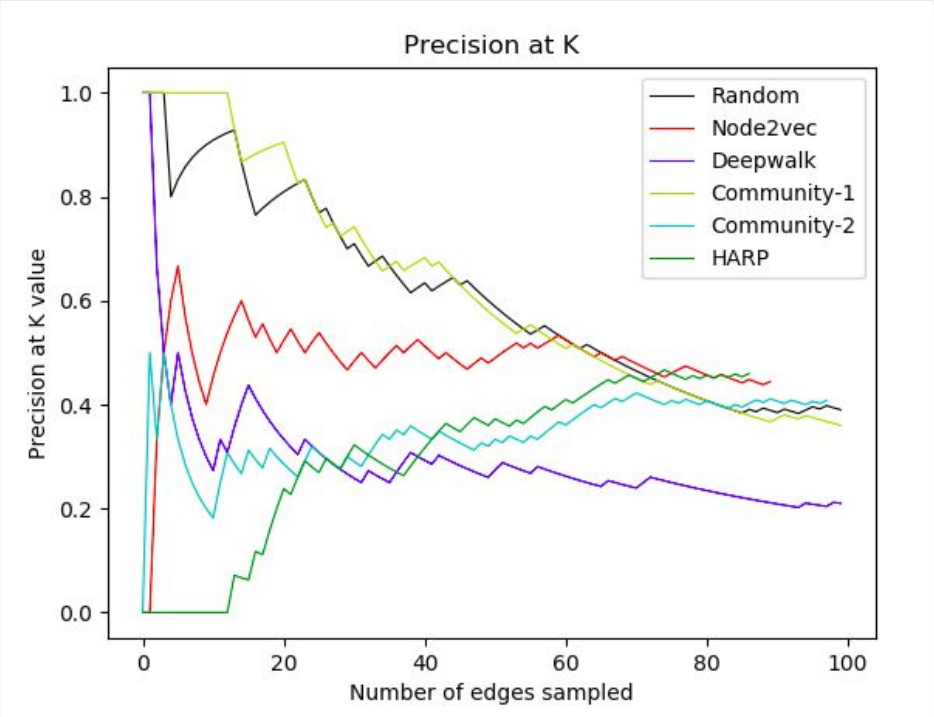
AUC Score: Comparison of Performance of Different Embedding Methods

	Louvain (Node2Vec)	Louvain (Stochastic)	Louvain (Community Embedding) [Method 1]	Louvain (Community Embedding) [Method 2]	Louvain (Community Embedding) [Method 3]	HARP (Node2Vec)
karate	0.675	0.619	0.699	0.754	0.436	0.770
BlogCatalog3	0.657	0.602	0.623	0.669	0.368	0.544
facebook	0.884	0.485	0.819	0.779	0.602	0.975
enrom	0.839	0.427	0.363	0.925	0.317	0.973
fb-cmu	0.697	0.727	0.621	0.650	0.493	0.859

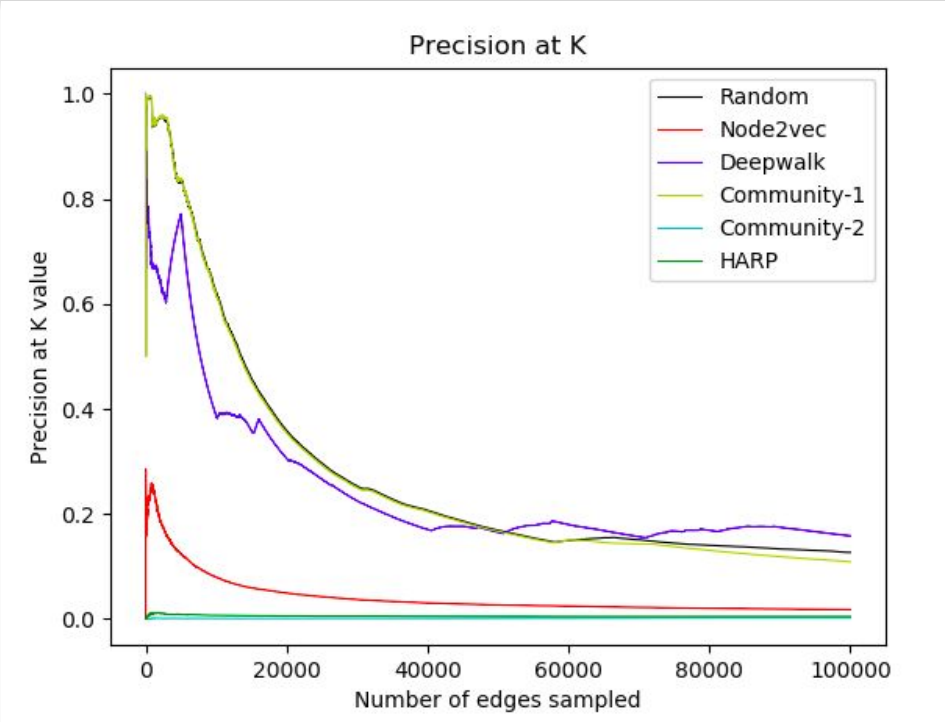
AUC

	Louvain (DeepWalk)	Louvain (Stochastic)	Louvain (Community Embedding) [Method 1] Deepwalk	Louvain (Community Embedding) [Method 2] DeepWalk	Louvain (Community Embedding) [Method 3] Deepwalk	HARP (Deep walk
karate	0.657	0.619	0.664	0.735	0.419	0.758
BlogCatalog3	0.627	0.602	0.566	0.651	0.507	0.527
facebook	0.813	0.485	0.722	0.829	0.733	0.945
enron	0.790	0.427	0.402	0.888	0.521	0.962
fb-cmu	0.683	0.727	0.566	0.693	0.594	0.871

Karate Dataset: Precision @ K Graph



BlogCatalog3 Dataset: Precision @ K Graph



Link Prediction: AUC Score (60% pruning)

	HARP	Louvain (Node2Vec)	Louvain (Deepwalk)	Louvain (Stochastic)	Louvain (Community Embedding) [Method 2]	Louvain M2 Deepwalk
karate	0.7634	0.7236	0.6521	0.4868	0.6053	0.5791
BlogCatalog3	0.8363	0.6426	0.6345	0.5225	0.6994	0.6783
facebook	0.7748	0.7578	0.6702	0.6529	0.7169	0.6912
enron	0.7859	0.7261	0.7016	0.6637	0.6785	0.6551
fb-cmu	0.6901	0.6711	0.6351	0.5764	0.654	0.5764

Node Classification: BlogCatalog

	Louvain (Node2Vec)	Louvain (Deepwalk)	Louvain (Stochastic)	Louvain (Community Embedding) [Method 1]	Louvain (Community Embedding) [Method 2]	Louvain M2 Deepwalk	Louvain M1 Deepwalk	Louvain M3 Node2vec	Louvain M3 Deepwalk	HARP
Micro-F1	0.427	0.344	0.350	0.304	0.434	0.391	0.332	0.353	0.306	0.304
Macro-F1	0.209	0.172	0.151	0.145	0.219	0.179	0.123	0.214	0.170	0.134

Node Classification: fb-cmu

	Louvain (Node2Vec)	Louvain (Deepwalk)	Louvain (Stochastic)	Louvain (Community Embedding) [Method 1]	Louvain (Community Embedding) [Method 2]	Louvain M2 Deepwalk	Louvain M1 Deepwalk	Louvain M3 Node2vec	Louvain M3 Deepwalk	HARP
Micro-F1	0.535	0.533	0.560	0.548	0.547	0.539	0.564	0.535	0.548	0.555
Macro-F1	0.275	0.248	0.257	0.237	0.278	0.258	0.240	0.243	0.273	0.244

Next Step...

1. Comparison of running time of baseline algorithms with our method.
2. Optimisations in Node2Vec and DeepWalk to make them faster.
3. Minimise the file handling part in the code.
4. Trying Hybrid Embeddings: Node2Vec at higher level communities and random at lower level communities.
5. Improving the quality of embeddings using inter-community edge detection so that connected nodes belonging to different communities have similar embeddings.
6. Greedy Hyperparameter tuning to generate efficient embeddings.
7. Test the embeddings on other network tasks like link prediction, node classification, etc.

THANK YOU!

SCALABLE METHODS FOR REPRESENTING LARGE SCALE GRAPHS

GROUP 4

Date: 04/11/2019

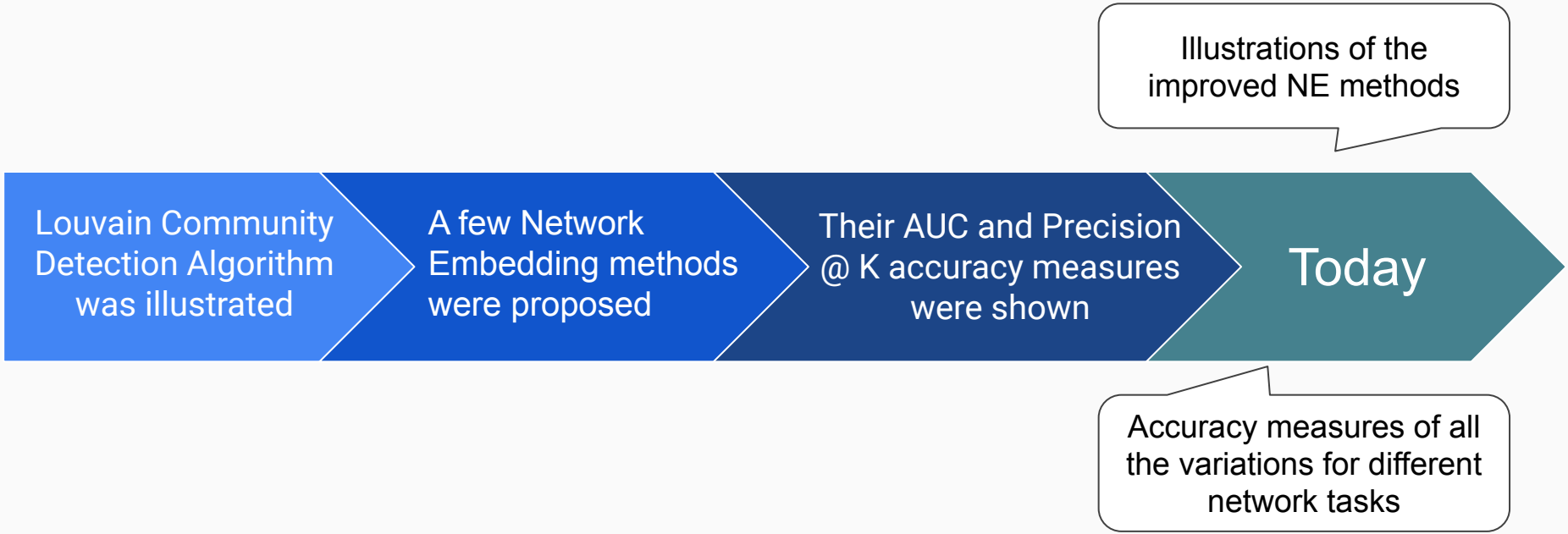
16CS10058: Lovish Chopra

16CS30044: Sarthak Chakraborty

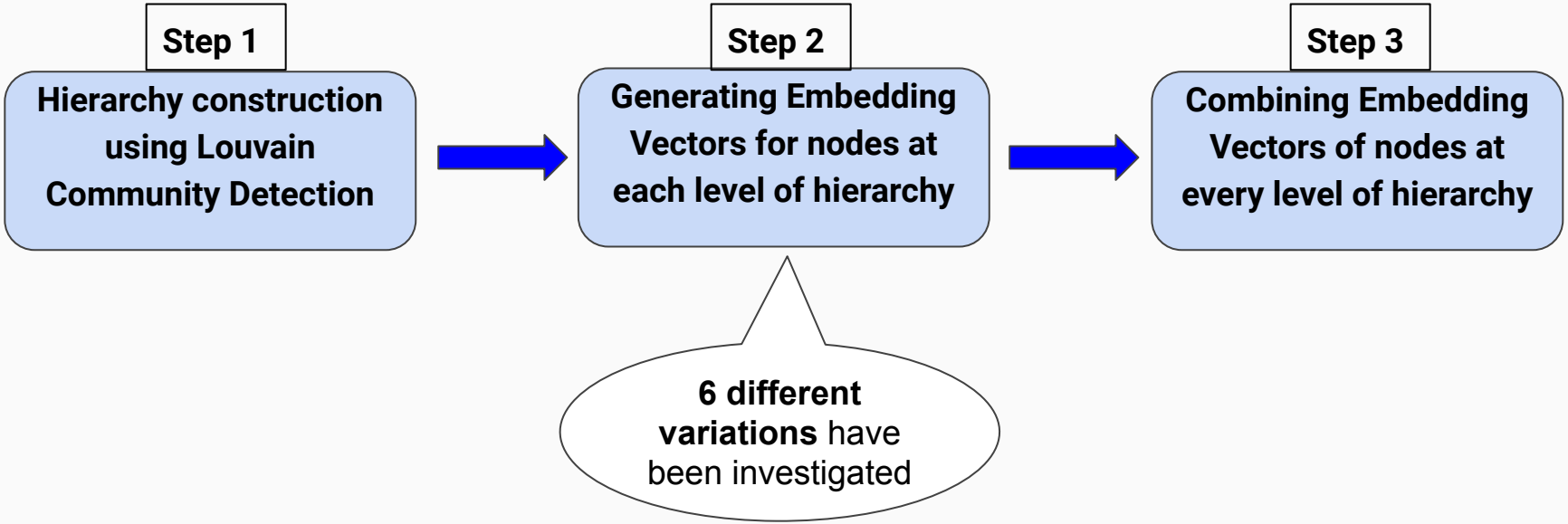
16ME10069: Partho Roychoudhury

Mentor: Ayan Kumar Bhowmick

A brief recap...



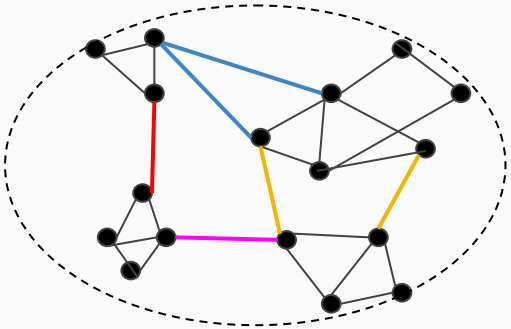
Basic Steps involved in our Approach



Step 1
Louvain Community Detection

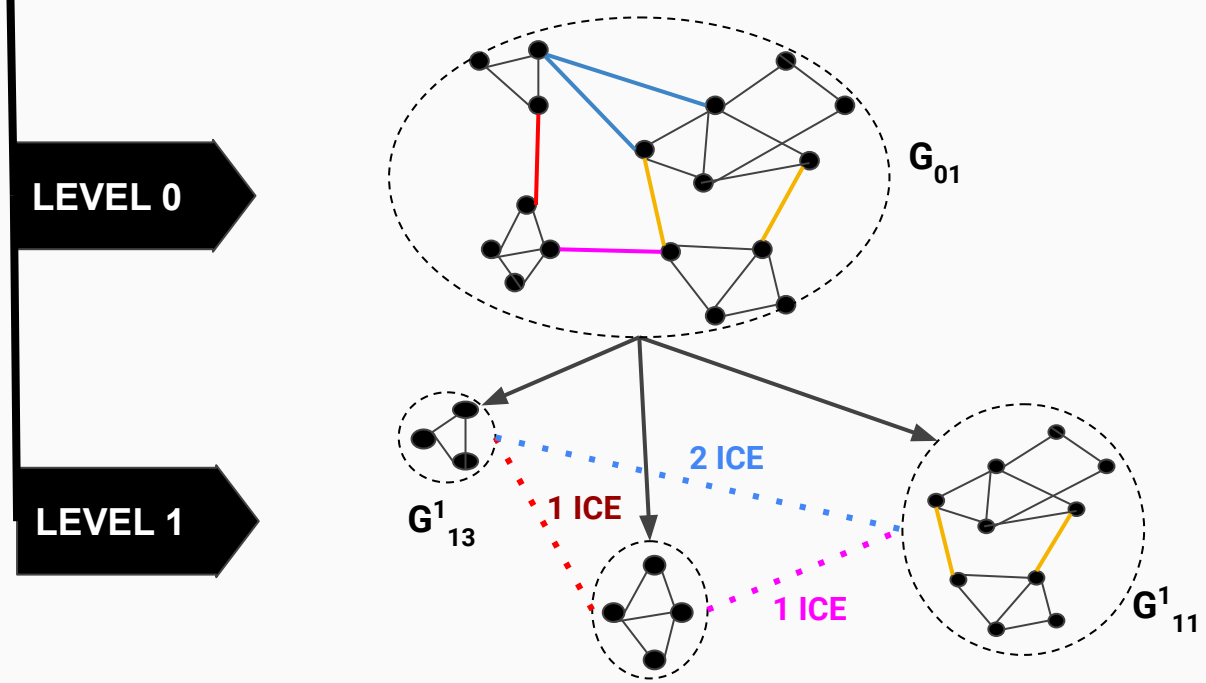
Step 1

LEVEL 0



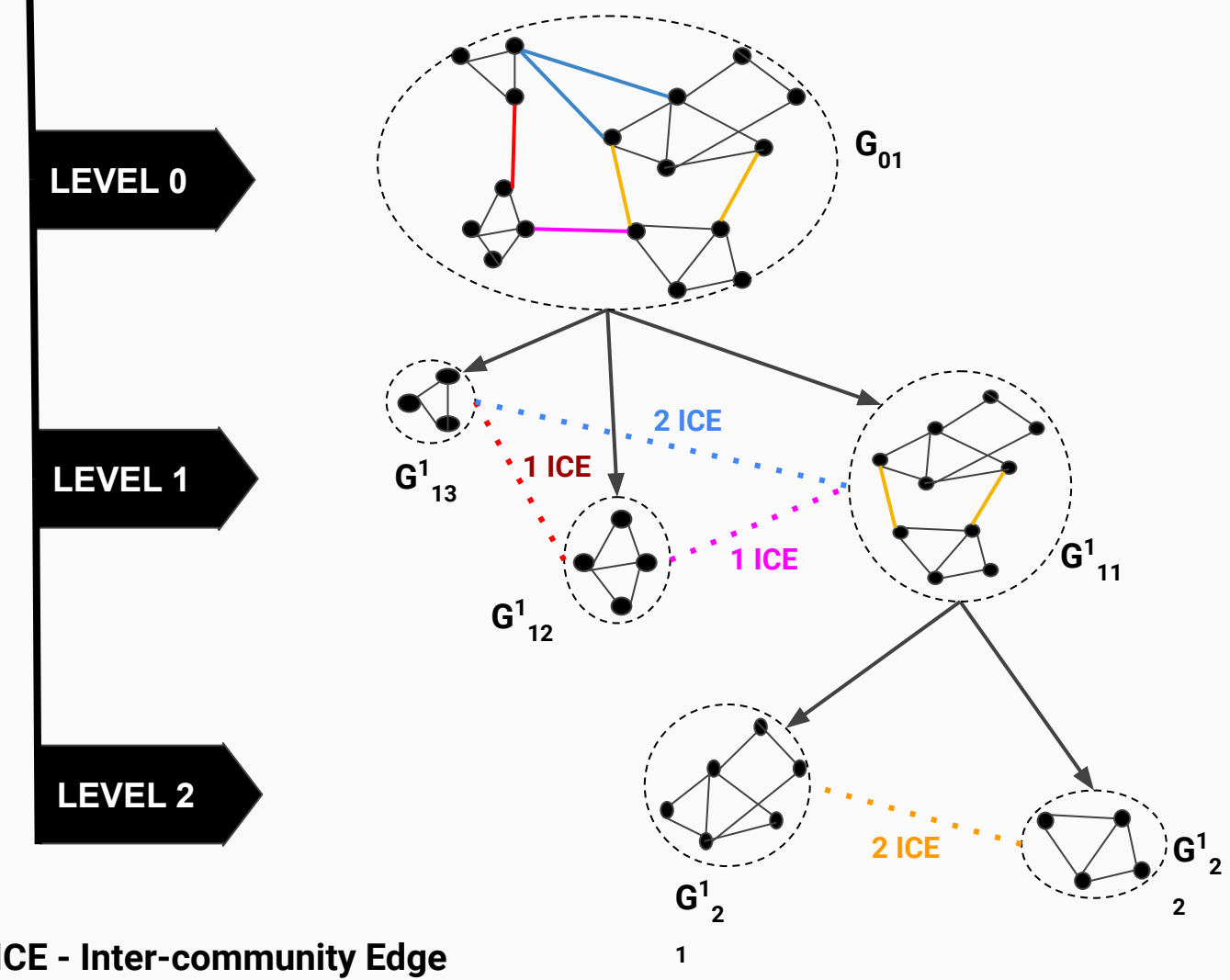
G_{01}

Step 1



ICE - Inter-community Edge

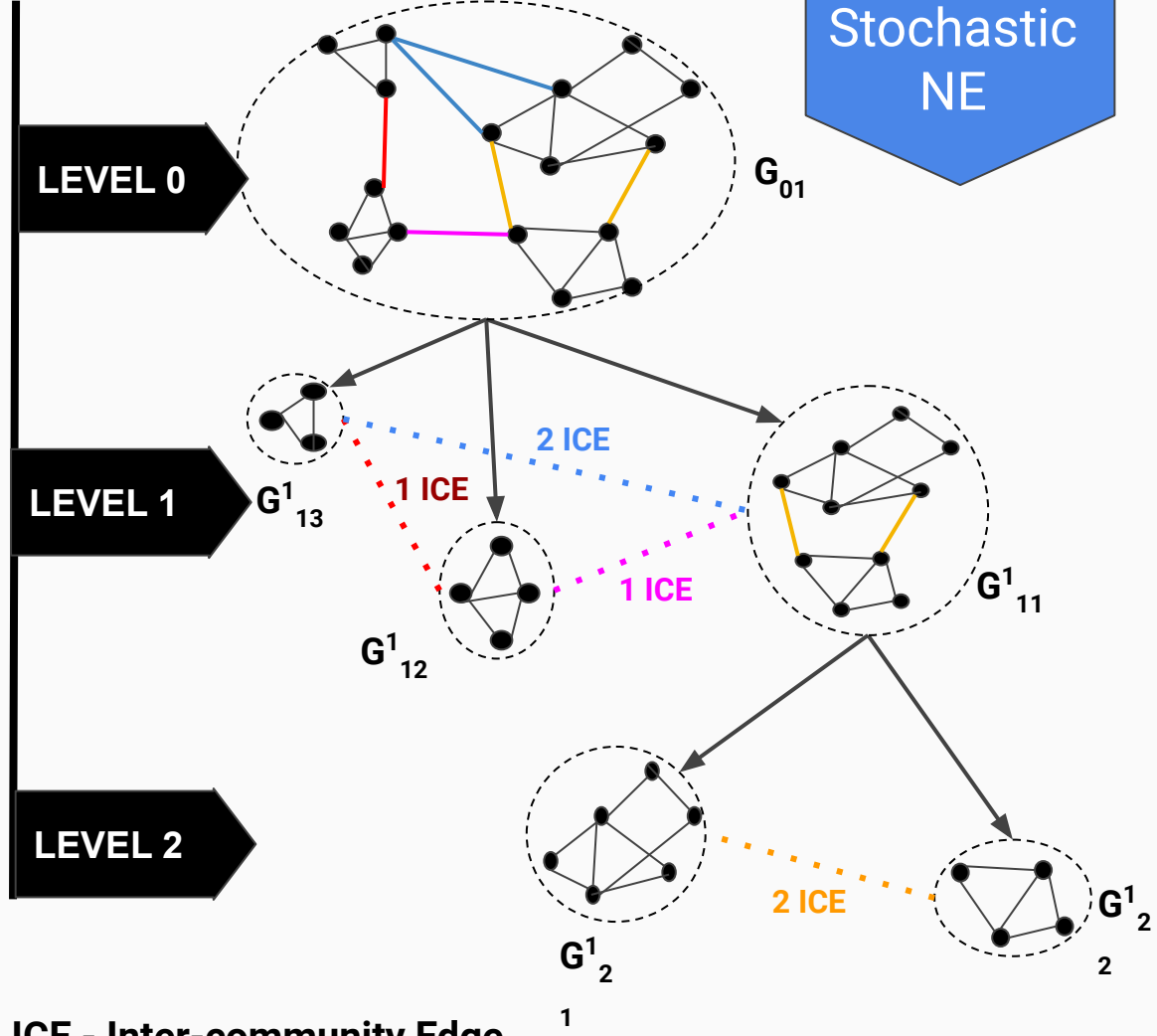
Step 1



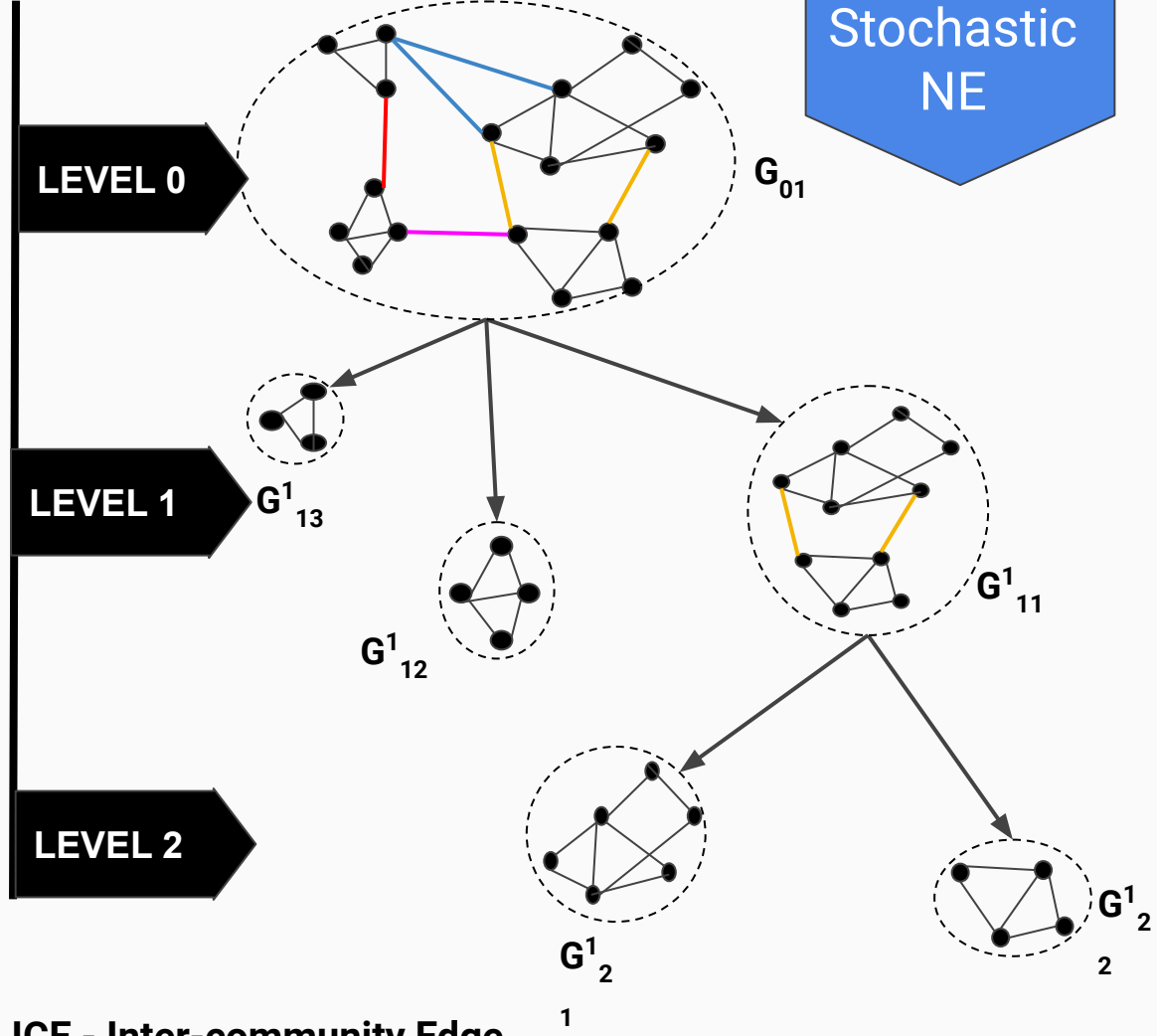
ICE - Inter-community Edge

Step 2
Generating Node Embeddings at Each Level (Stochastic NE)

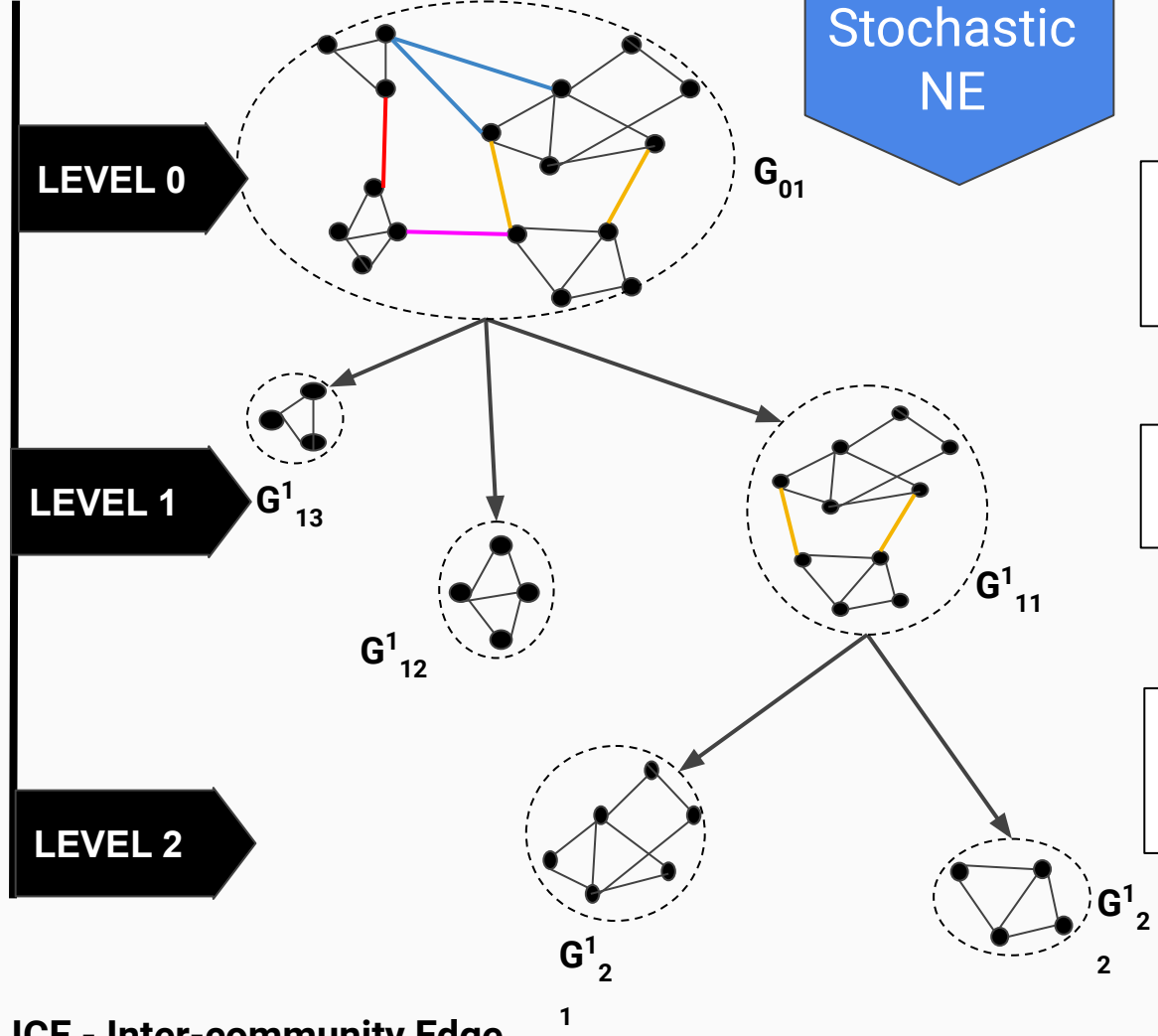
Ignoring the inter-community edges



ICE - Inter-community Edge



ICE - Inter-community Edge



Presence of intercommunity links have been totally ignored in this process

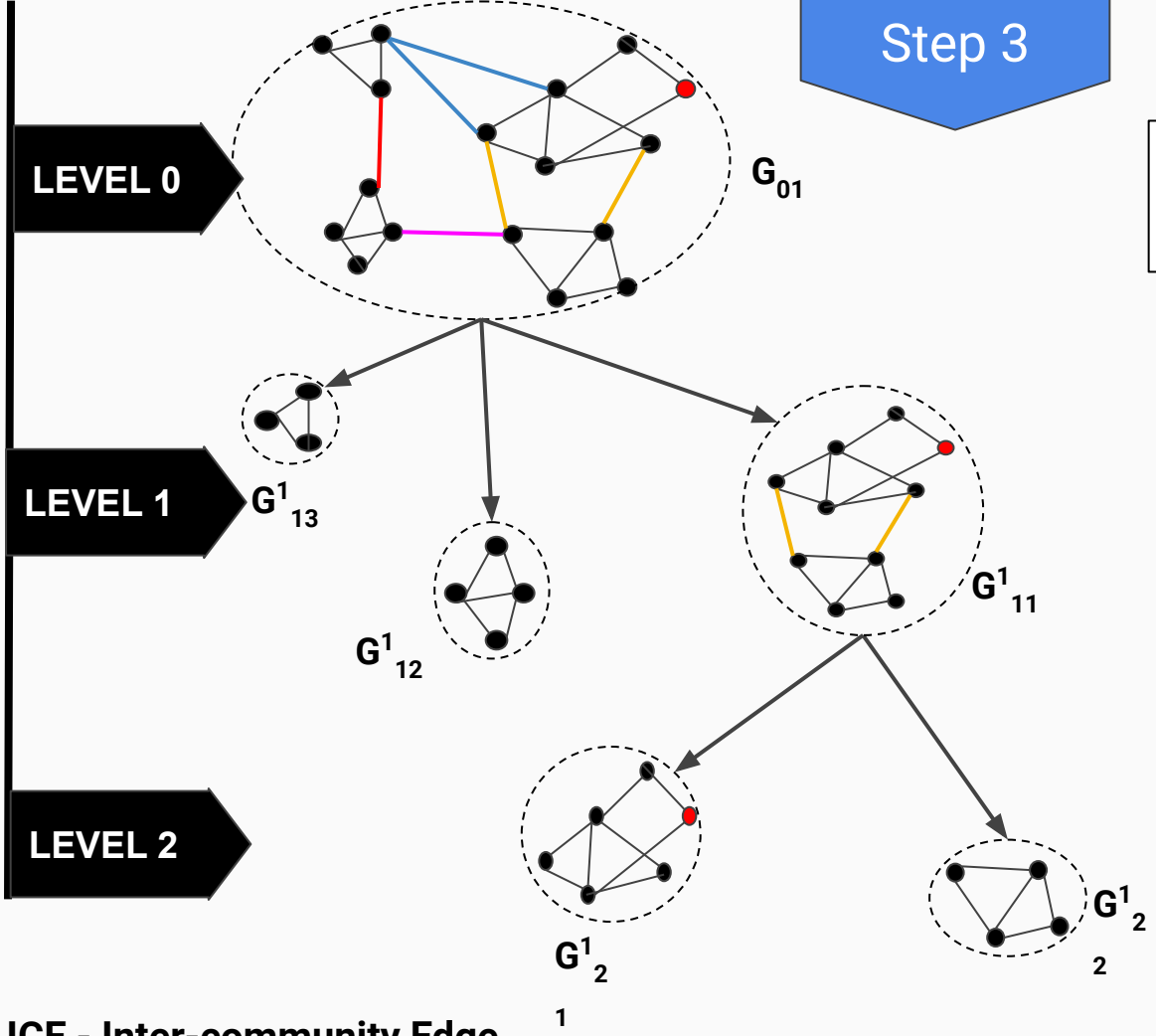
Each metanode is assigned a node embedding **randomly**

All nodes within the metanode at a specific level gets the same **Stochastic NE**

ICE - Inter-community Edge

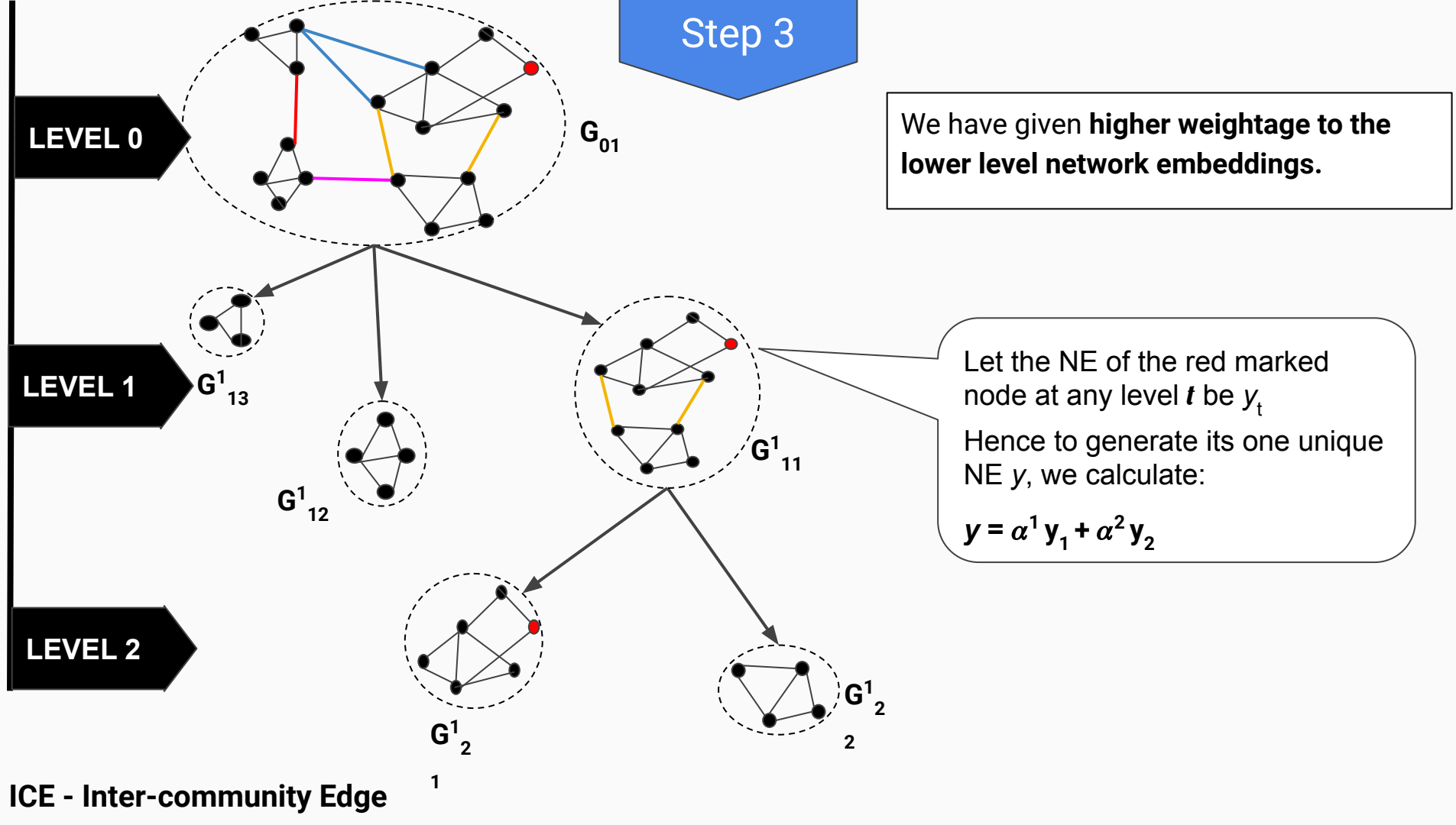
Step 3
Combining Node Embeddings Generated at Different Levels

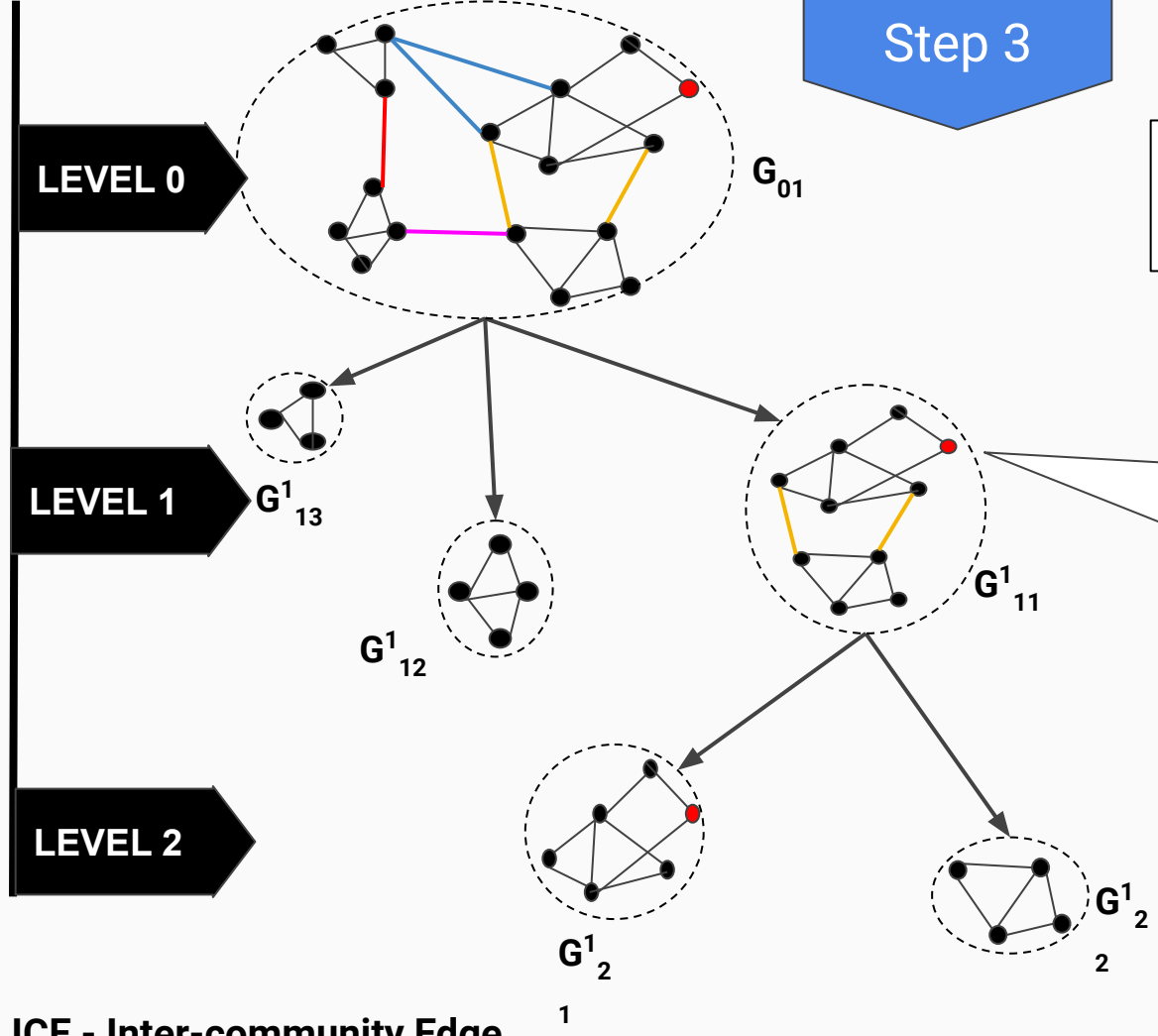
Step 3



We have given **higher weightage to the lower level network embeddings.**

ICE - Inter-community Edge


Step 3
LEVEL 0
LEVEL 1
LEVEL 2
G_{01}
G^1_{13}
G^1_{12}
G^1_{11}
$G^1_{2\ 1}$
$G^1_{2\ 2}$
We have given **higher weightage to the lower level network embeddings.**
Let the NE of the red marked node at any level ***t*** be y_t
Hence to generate its one unique NE *y*, we calculate:
$y = \alpha^1 y_1 + \alpha^2 y_2$
ICE - Inter-community Edge



We have given **higher weightage to the lower level network embeddings.**

Let the NE of the red marked node at any level ***t*** be y_t

Hence to generate its one unique NE *y*, we calculate:

$$y = \alpha^1 y_1 + \alpha^2 y_2$$

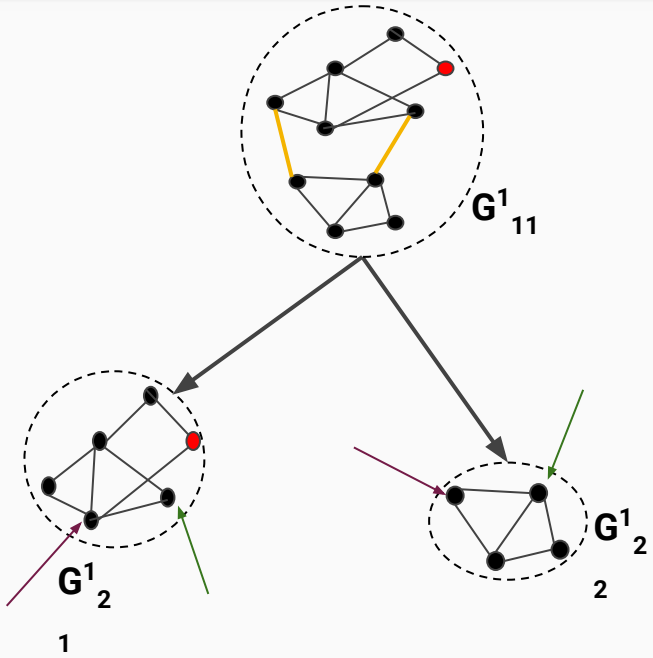
Or, in general, we use:

$$y = \alpha^1 y_1 + \alpha^2 y_2 + \alpha^3 y_3 + \ldots + \alpha^t y_t$$

ICE - Inter-community Edge

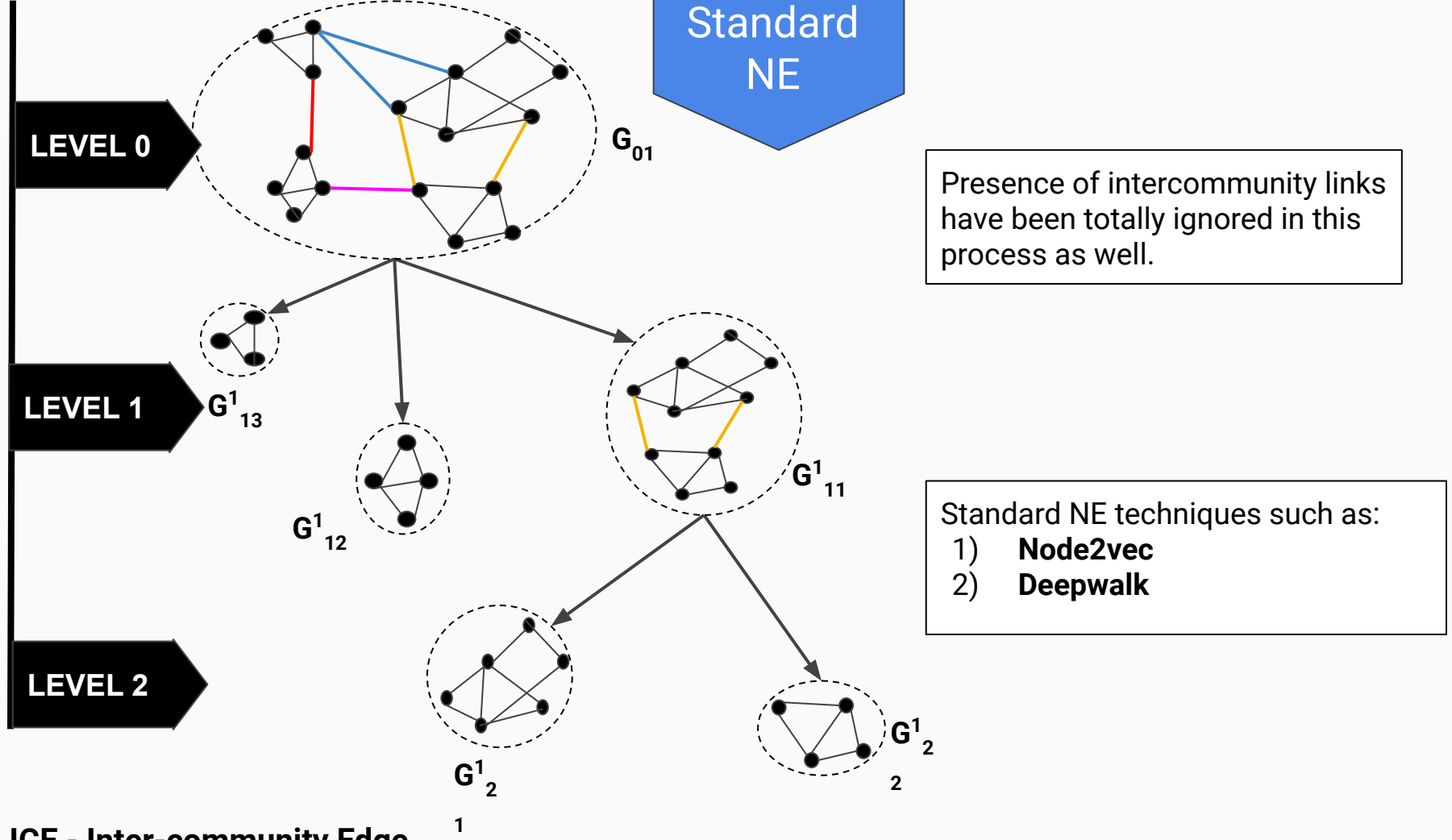
Need of Improving Step 2 of the Method

- Stochastic embedding affects quality
- Nodes which are actually **connected** in the graph gets **separated in Step 1** ends up getting different embeddings
- Need to **detect edges between nodes** belonging to different communities.



Step 2
Generating Node Embeddings at Each Level (Standard NE)

Ignoring the inter-community edges



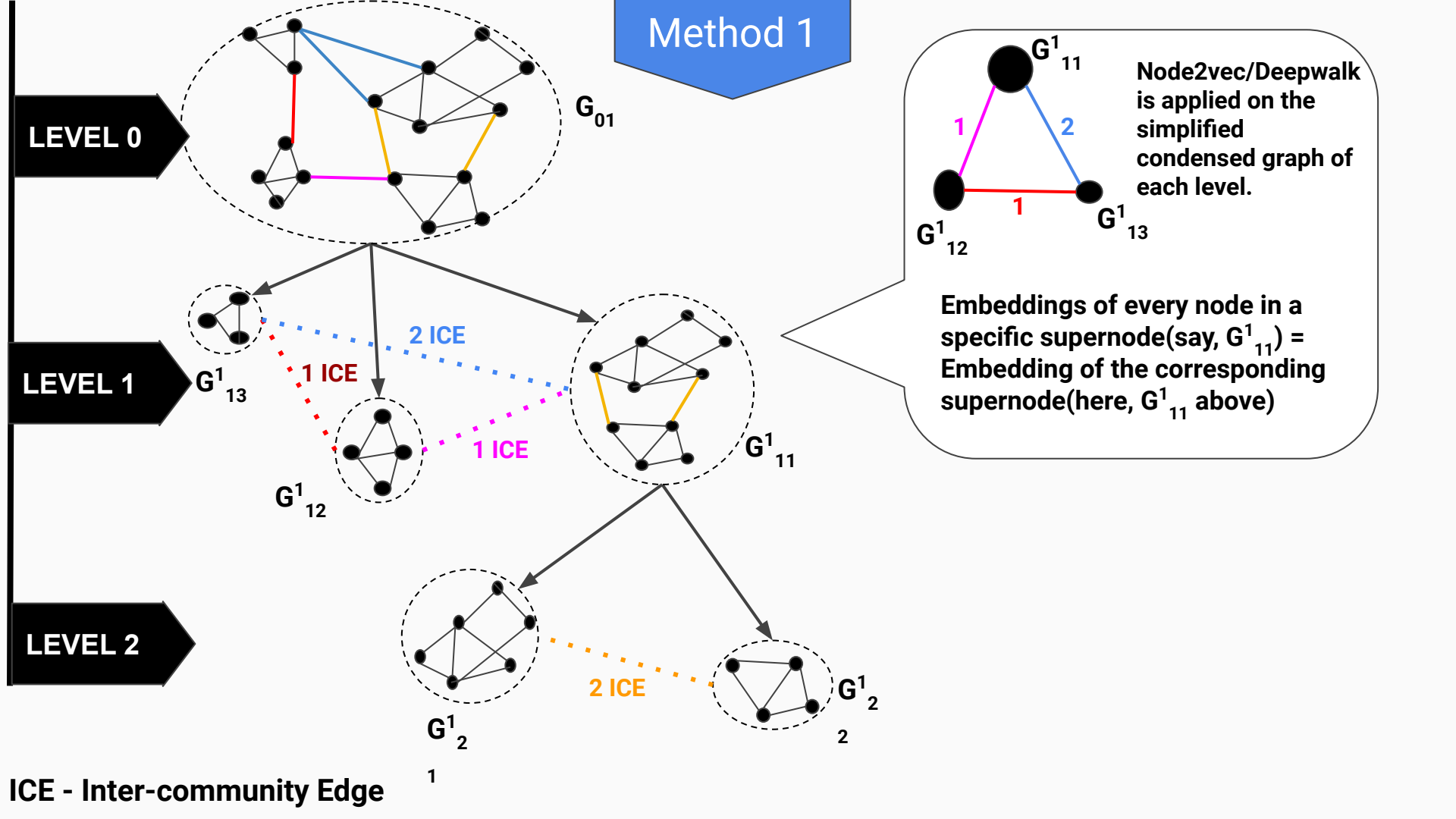
Presence of intercommunity links have been totally ignored in this process as well.

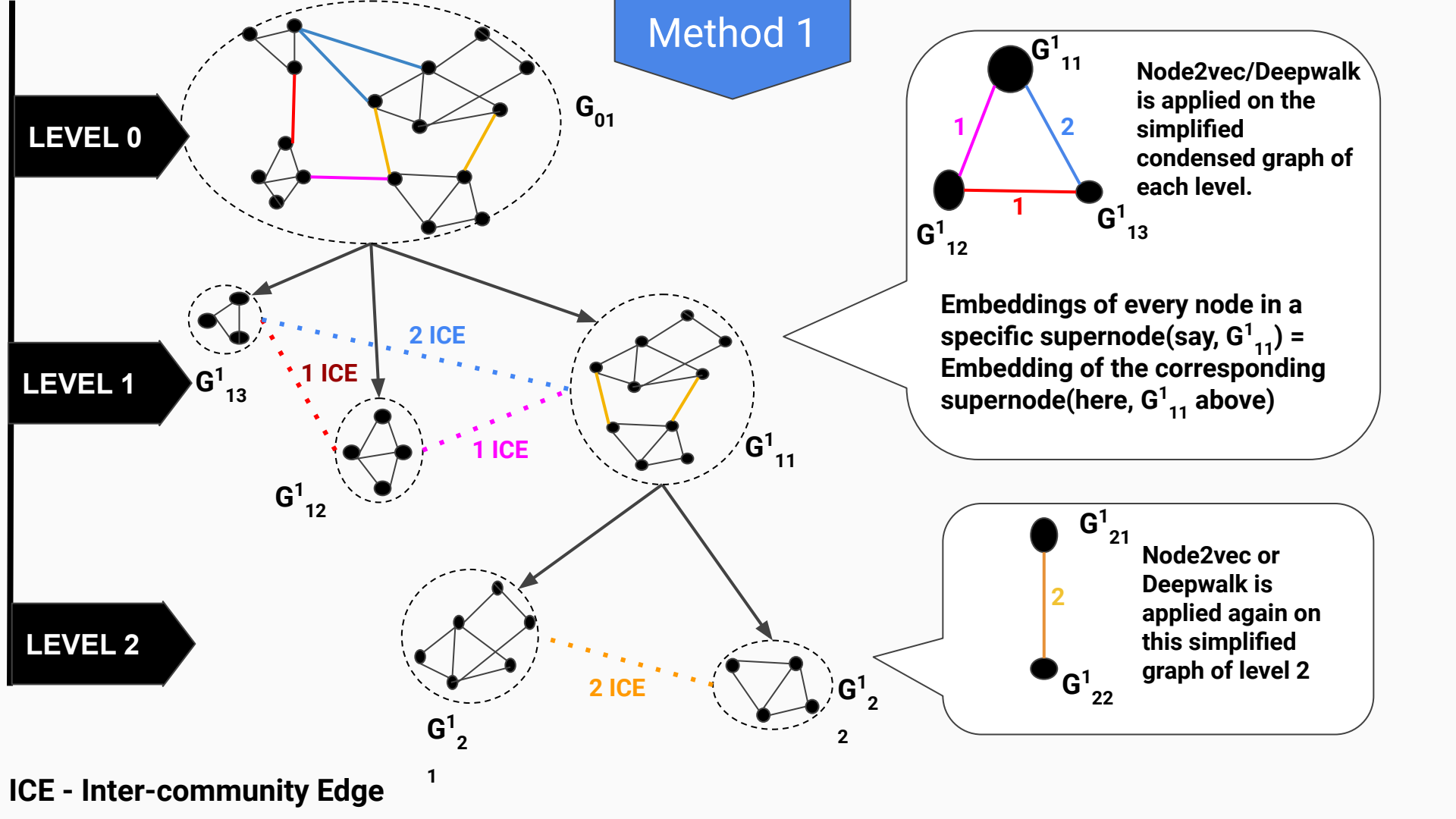
Standard NE techniques such as:

1) **Node2vec**
2) **Deepwalk**

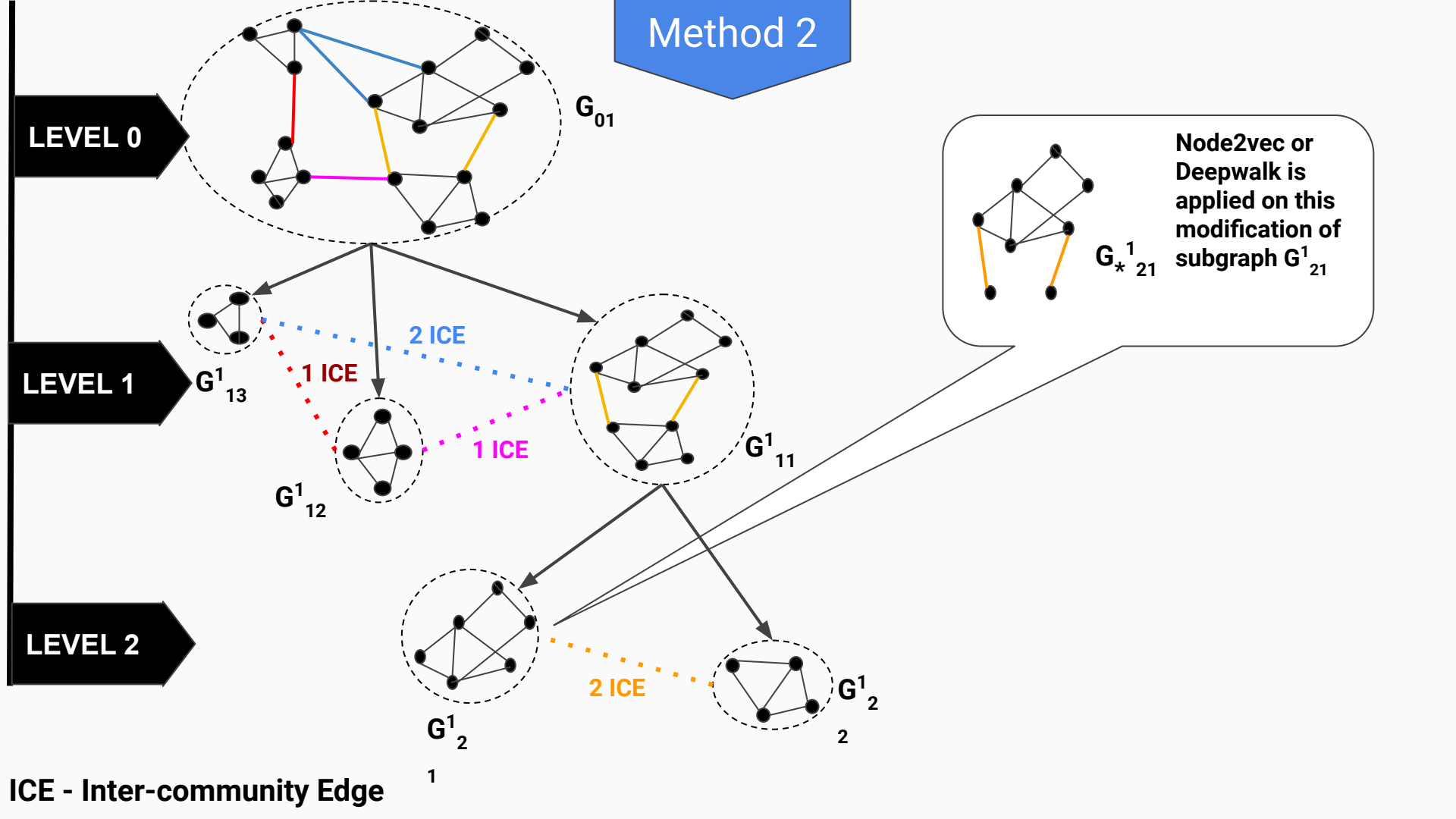
ICE - Inter-community Edge

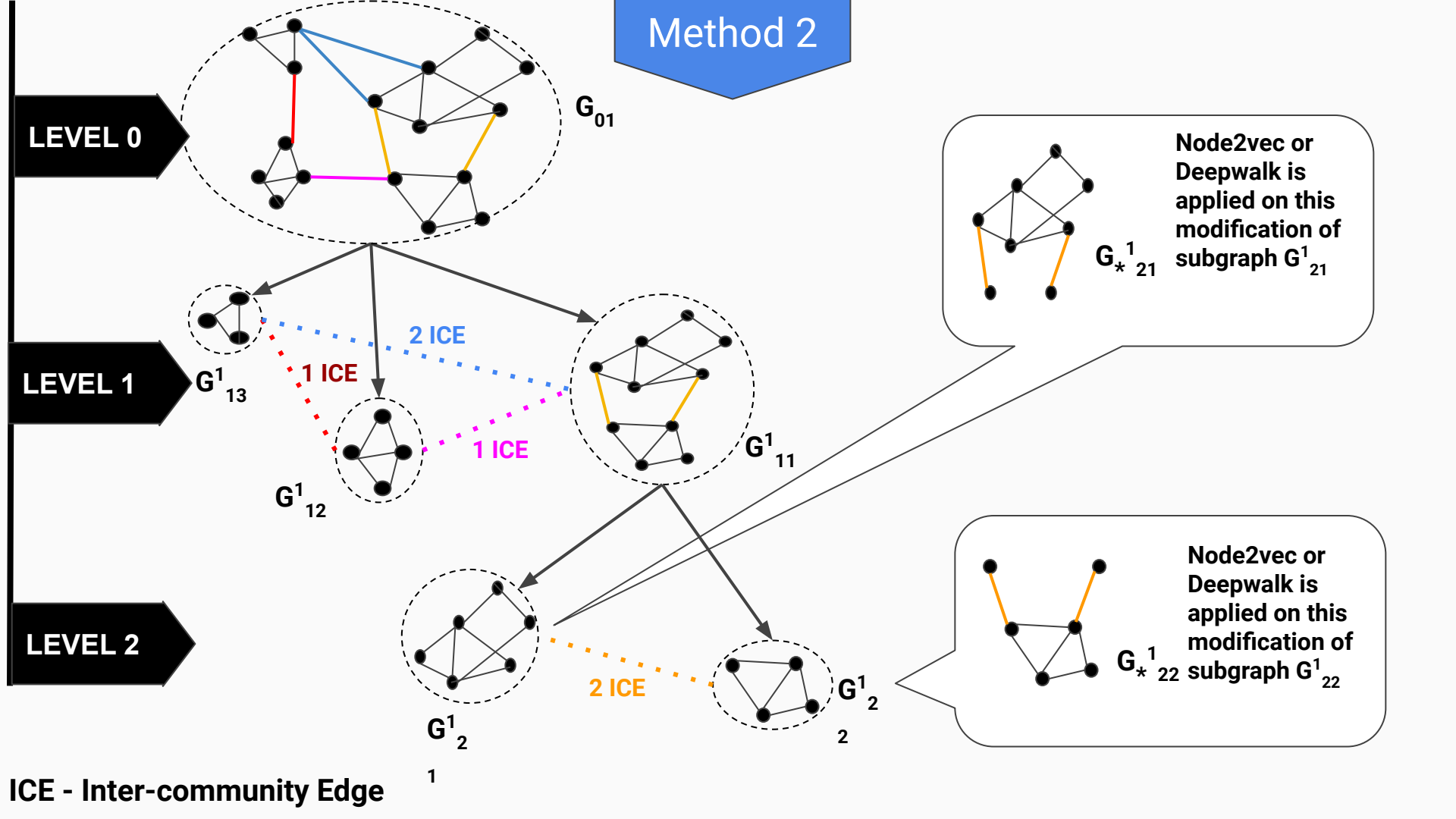
Step 2
Louvain Community Embedding (Method 1)


Method 1
LEVEL 0
LEVEL 1
LEVEL 2
G_{01}
G^1_{13}
G^1_{12}
G^1_{11}
$G^1_{2\,1}$
$G^1_{2\,2}$
2 ICE
1 ICE
1 ICE
2 ICE
G^1_{11}
1
2
1
G^1_{12}
G^1_{13}
Node2vec/Deepwalk is applied on the simplified condensed graph of each level.
Embeddings of every node in a specific supernode(say, G^1_{11}) = Embedding of the corresponding supernode(here, G^1_{11} above)
ICE - Inter-community Edge

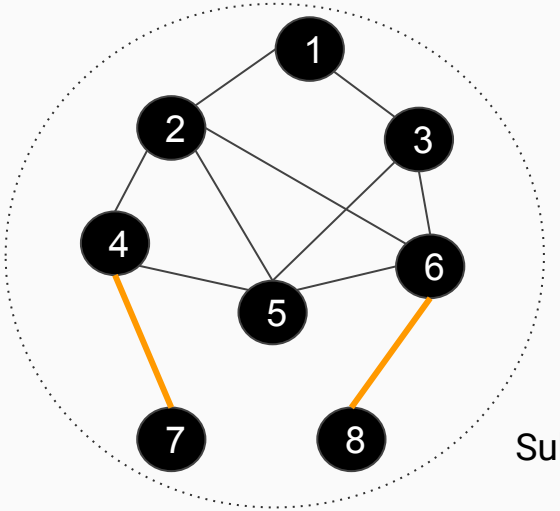

Method 1
LEVEL 0
LEVEL 1
LEVEL 2
G_{01}
G^1_{13}
G^1_{12}
G^1_{11}
$G^1_{2\ 1}$
$G^1_{2\ 2}$
2 ICE
1 ICE
1 ICE
2 ICE
G^1_{11}
1
2
1
G^1_{12}
G^1_{13}
Node2vec/Deepwalk is applied on the simplified condensed graph of each level.
Embeddings of every node in a specific supernode(say, G^1_{11}) = Embedding of the corresponding supernode(here, G^1_{11} above)
G^1_{21}
2
G^1_{22}
Node2vec or Deepwalk is applied again on this simplified graph of level 2
ICE - Inter-community Edge

Step 2
Louvain Community Embedding (Method 2)


Method 2
LEVEL 0
LEVEL 1
LEVEL 2
G_{01}
G^1_{13}
G^1_{12}
G^1_{11}
$G^1_{2\,1}$
$G^1_{2\,2}$
2 ICE
1 ICE
1 ICE
2 ICE
$G^{*\,1}_{21}$
Node2vec or Deepwalk is applied on this modification of subgraph G^1_{21}
ICE - Inter-community Edge

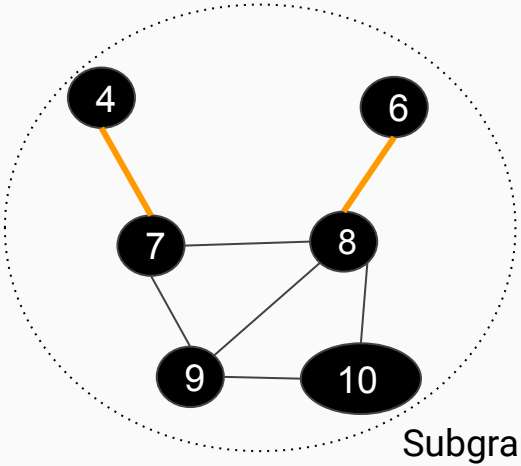

Method 2
LEVEL 0
LEVEL 1
LEVEL 2
G_{01}
G^1_{13}
G^1_{12}
G^1_{11}
$G^1_{2\,1}$
$G^1_{2\,2}$
2 ICE
1 ICE
1 ICE
2 ICE
$G_*{}^1_{21}$
Node2vec or Deepwalk is applied on this modification of subgraph G^1_{21}
$G_*{}^1_{22}$
Node2vec or Deepwalk is applied on this modification of subgraph G^1_{22}
ICE - Inter-community Edge

Method 2



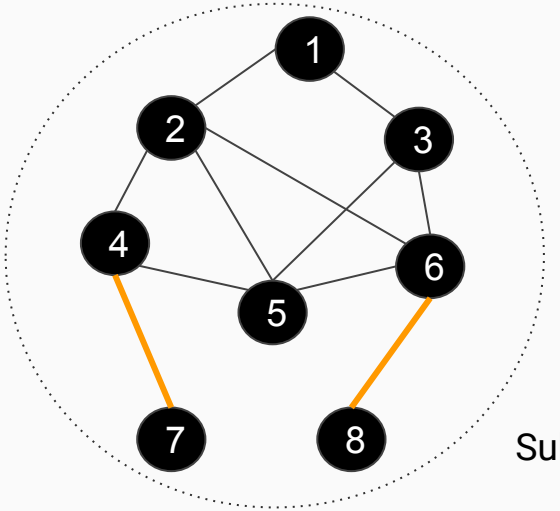
Subgraph $G_*^1{}_{21}$

Notice that nodes 4,6,7 and 8 are actually the intercommunity nodes.



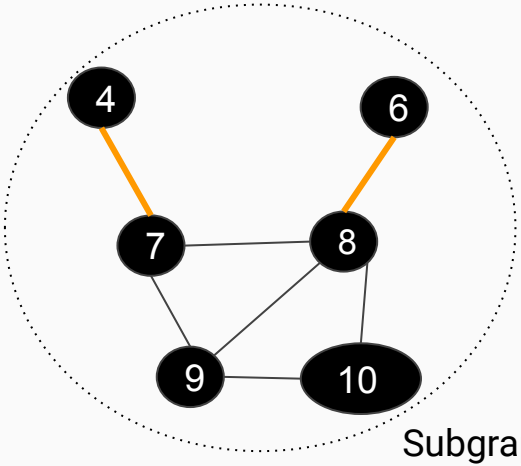
Subgraph $G_*^1{}_{22}$

Method 2



Subgraph $G_*^1{}_{21}$

Notice that nodes 4,6,7 and 8 are actually the intercommunity nodes.



Subgraph $G_*^1{}_{22}$

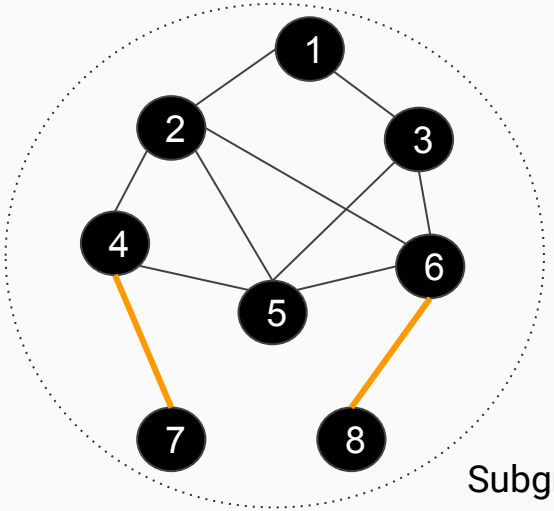
Nodes 4 and 6 originally belong to subgraph $G^1{}_{21}$ while nodes 7 and 8 are just auxiliary intercommunity nodes of the above mentioned modified subgraph $G_*^1{}_{21}$

So, higher weightage will be given to NE of nodes 4 and 6 obtained from this subgraph.

Nodes 7 and 8 originally belong to subgraph $G^1{}_{22}$ while nodes 4 and 6 are just auxiliary intercommunity nodes of the above mentioned modified subgraph $G_*^1{}_{22}$

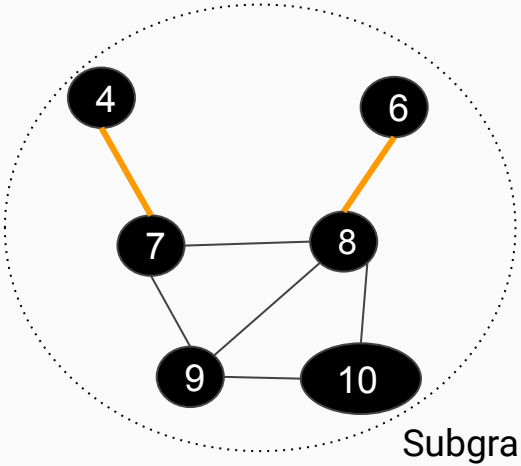
So, higher weightage will be given to NE of nodes 7 and 8 obtained from this subgraph.

Method 2



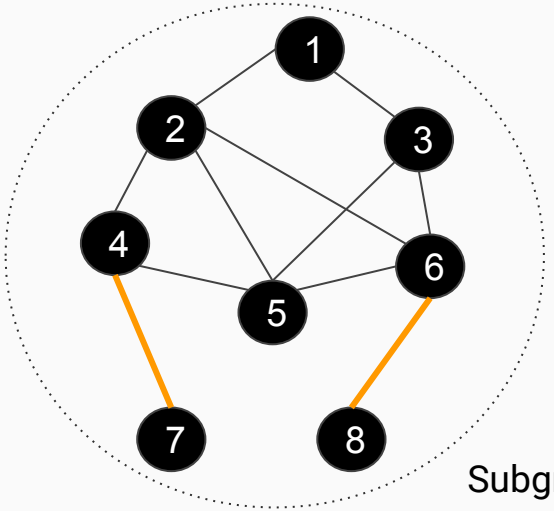
Subgraph $G_*^1{}_{21}$

How to get the combined embedding vector for intercommunity nodes?

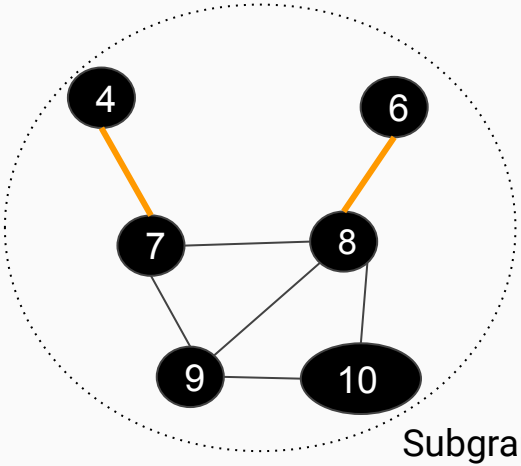


Subgraph $G_*^1{}_{22}$

Method 2



Subgraph $G_*^1{}_{21}$



Subgraph $G_*^1{}_{22}$

How to get the combined embedding vector for intercommunity nodes?

Embedding of node 4 = y_{41}

Embedding of node 7 = y_{71}

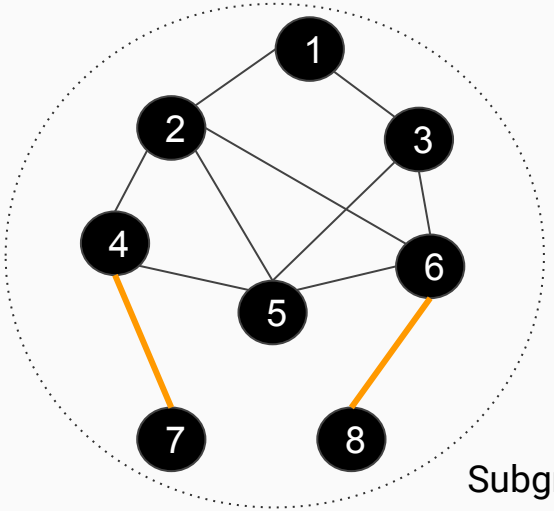
Embedding of node 4 = y_{42}

Embedding of node 7 = y_{72}

$y_4 = y_{41} + \alpha y_{42}, \alpha \in [0,1]$

$y_7 = \alpha y_{71} + y_{72}, \alpha \in [0,1]$

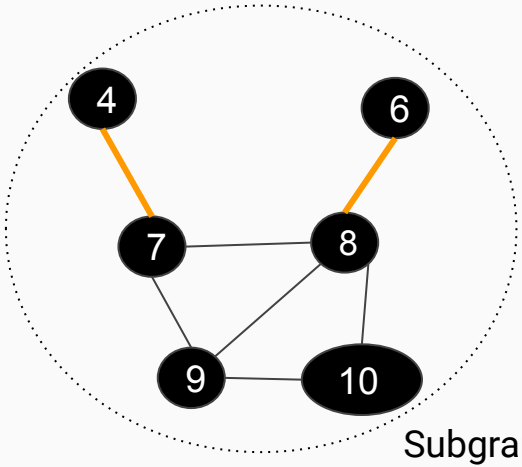
Method 2



Subgraph $G_*^1{}_{21}$

In general, at **level t** and for **node** belonging to **subgraph i**:

$$y_{node} = y_{node,i} + \alpha \Sigma y_{node,j}$$



Subgraph $G_*^1{}_{22}$

Embedding of node 4 = y_{41}

Embedding of node 7 = y_{71}

Embedding of node 4 = y_{42}

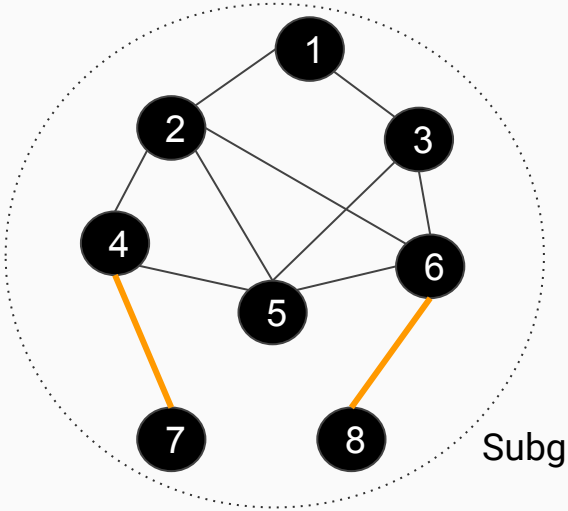
Embedding of node 7 = y_{72}

$y_4 = y_{41} + \alpha y_{42}, \alpha \in [0,1]$

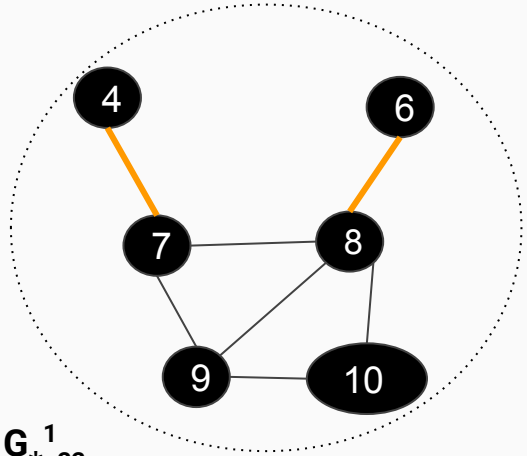
$y_7 = \alpha y_{71} + y_{72}, \alpha \in [0,1]$

Step 2
Louvain Community Embedding (Method 3)

Method 3



Subgraph $G_*{}^1{}_{21}$

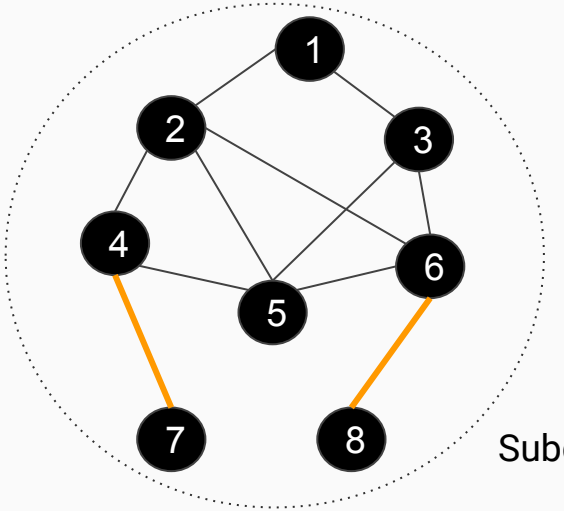


Subgraph $G_*{}^1{}_{22}$

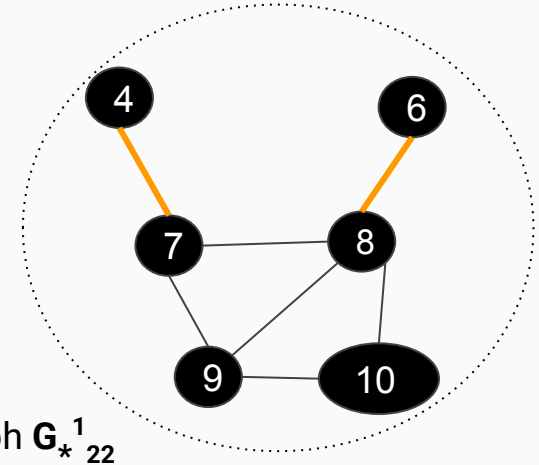
NE generated as in Method 2

Deals with the aggregation of NE at a specific level only for the intercommunity nodes.

Method 3



Subgraph $G_*^{1}{}_{21}$



Subgraph $G_*^{1}{}_{22}$

Embedding of node 4 = y_{41}

Embedding of node 7 = y_{71}

Embedding of node 4 = y_{42}

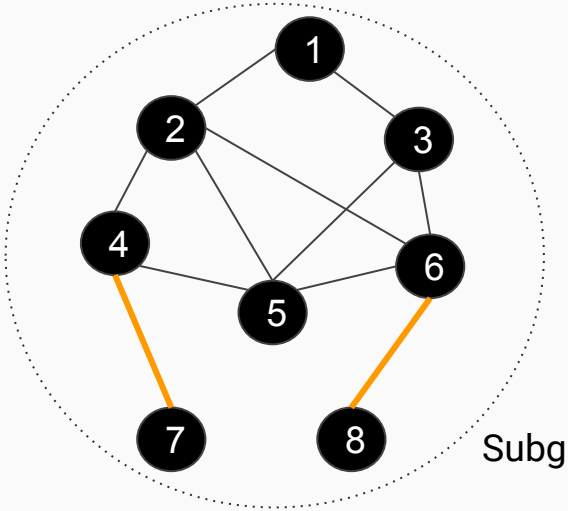
Embedding of node 7 = y_{72}

Now we want to write:

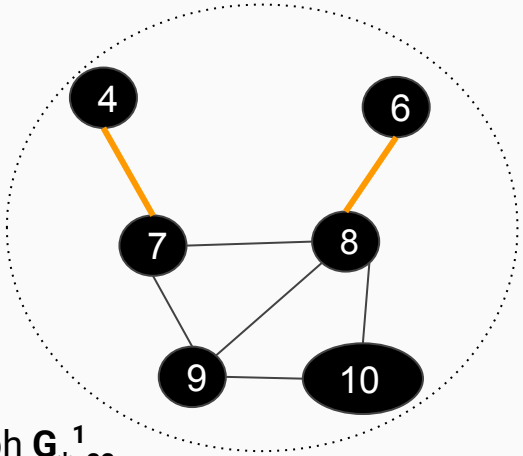
$y_4 = w_{41}y_{41} + w_{42}y_{42}$, such that: $\mathbf{w_{41} + w_{42} = 1}$ and $\mathbf{w_{41} > w_{42}}$

$y_7 = w_{71}y_{71} + w_{72}y_{72}$, such that: $\mathbf{w_{71} + w_{72} = 1}$ and $\mathbf{w_{71} < w_{72}}$

Method 3



How to select such $\mathbf{w_{41}}$, $\mathbf{w_{42}}$, $\mathbf{w_{71}}$ and $\mathbf{w_{72}}$?



Subgraph $\mathbf{G_*}^1_{21}$

Subgraph $\mathbf{G_*}^1_{22}$

Embedding of node 4 = y_{41}

Embedding of node 7 = y_{71}

Embedding of node 4 = y_{42}

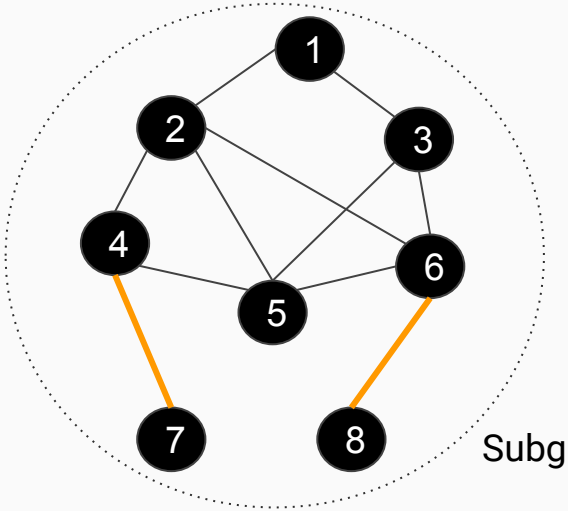
Embedding of node 7 = y_{72}

Now we want to write:

$y_4 = w_{41}y_{41} + w_{42}y_{42}$, such that: $\mathbf{w_{41} + w_{42} = 1}$ and $\mathbf{w_{41} > w_{42}}$

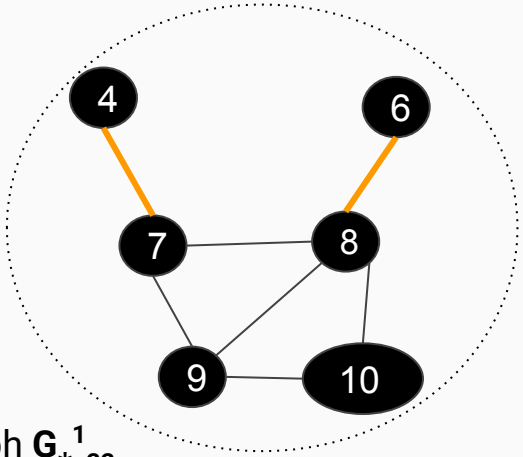
$y_7 = w_{71}y_{71} + w_{72}y_{72}$, such that: $\mathbf{w_{71} + w_{72} = 1}$ and $\mathbf{w_{71} < w_{72}}$

Method 3



Subgraph $G_*^1{}_{21}$

How to select such w_{41}, w_{42}, w_{71} and w_{72} ?



Subgraph $G_*^1{}_{22}$

In this subgraph $G_*^1{}_{21}$:

Total no of links = l_1=10

Total no of links within community = l_2= 8

Total no of links connecting subgraph $G^1{}_{21}$ to $G^1{}_{22}$= l_3 = 2

Then, w_{41}= l_2/l_1 = 0.8

w_{42}= l_3/l_1= 0.2

In this subgraph $G_*^1{}_{22}$:

Total no of links = l_1=7

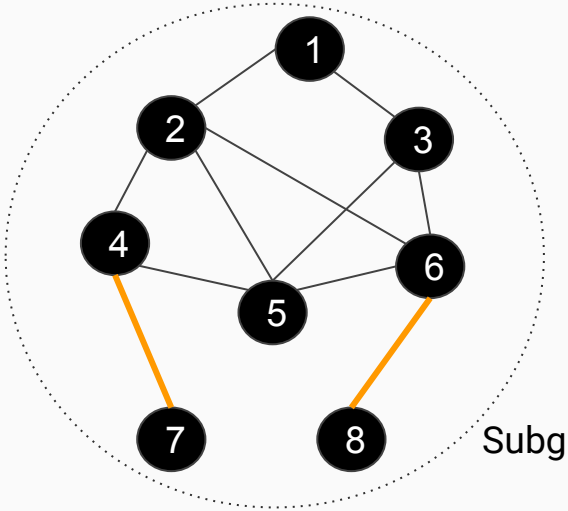
Total no of links within community = l_2= 5

Total no of links connecting subgraph $G^1{}_{22}$ to $G^1{}_{21}$= l_3 = 2

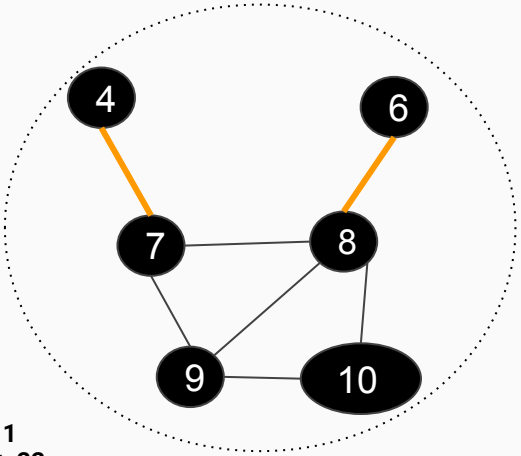
Then, w_{72}= l_2/l_1 = 0.71

w_{71}= l_3/l_1= 0.29

Method 3



Subgraph $G_*^1{}_{21}$



Subgraph $G_*^1{}_{22}$

Embedding of node 4 = y_{41}

Embedding of node 7 = y_{71}

Embedding of node 4 = y_{42}

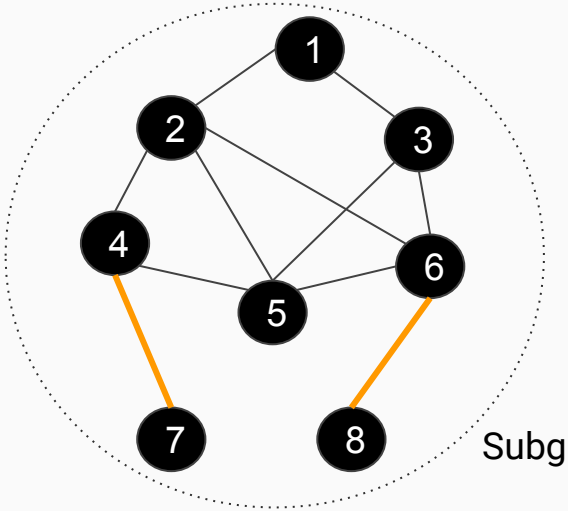
Embedding of node 7 = y_{72}

$y_4 = w_{41}y_{41} + w_{42}y_{42} = 0.8y_{41} + 0.2y_{42}$

$y_7 = w_{71}y_{71} + w_{72}y_{72} = 0.29y_{71} + 0.71y_{72}$

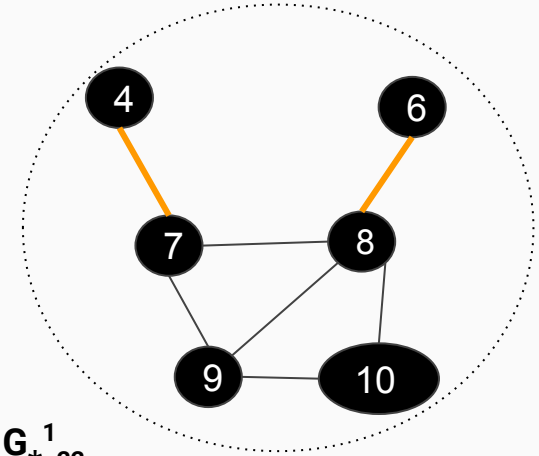
Step 2
Louvain Community Embedding (Method 4)

Method 4



Subgraph $G_*^1{}_{21}$

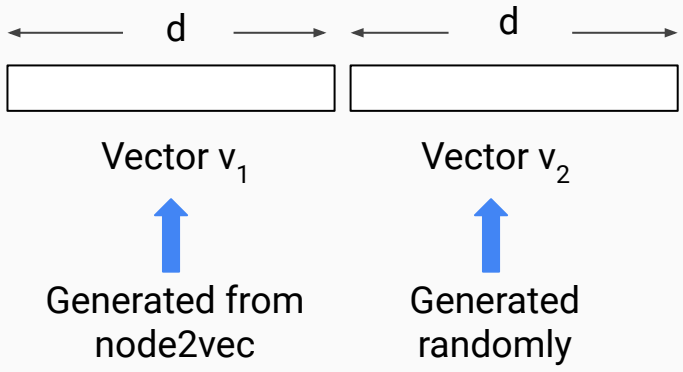
Use 2 components of NE for a node



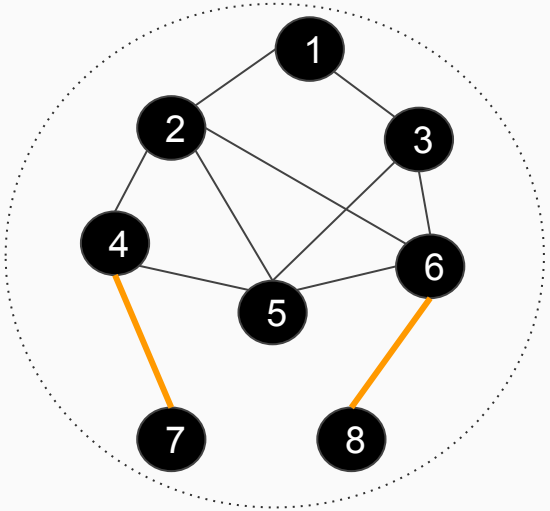
Subgraph $G_*^1{}_{22}$

Vector v_1 generated for all nodes using Method 2.

Vector v_2 generated only for inter-community nodes.



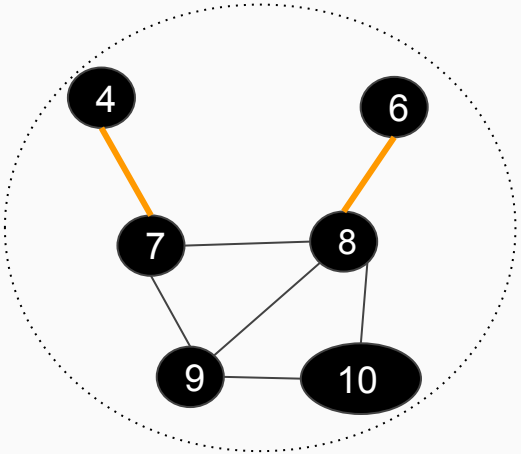
Method 4



Subgraph $G_*^{1}{}_{21}$

v_1 for node i = y_{i1}

v_2 for node i = $y_{r,i}$



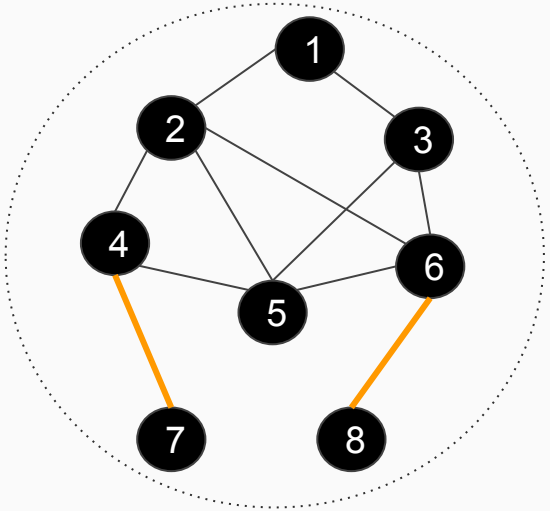
Subgraph $G_*^{1}{}_{22}$

v_2 is generated randomly for nodes 4, 6, 7 and 8 (intercommunity nodes) satisfying the constraint

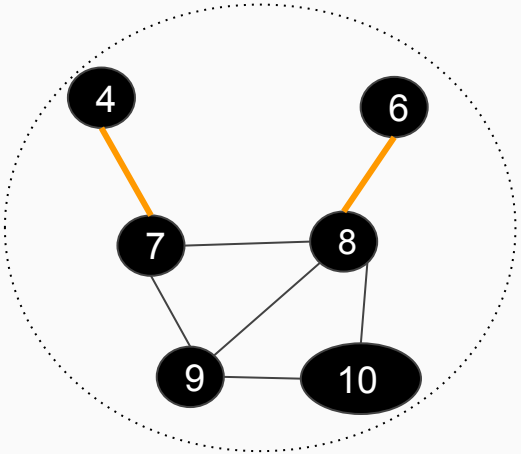
$$\mathbf{y_{r,4} = y_{r,7}} \quad \text{and} \quad \mathbf{y_{r,6} = y_{r,8}}$$

Reason: (4,7) and (6,8) forms as edge

Method 4



Subgraph $G_{*}^{1}{}_{21}$



Subgraph $G_{*}^{1}{}_{22}$

Non-Intercommunity Nodes

100% weightage to d1

$y_i = y_{i1}$

Intercommunity Nodes

80% weightage to d1
20% weightage to d2

$y_i = 0.8y_{i1} + 0.2y_{r,i}$

Results

Datasets Used

karate

No of Nodes = **34**
No of Links = **78**

Facebook

No of Nodes = **4039**
No of Links = **88239**

BlogCatalog3

No of Nodes = **10312**
No of Links = **333983**

fb-CMU

No of Nodes = **6600**
No of Links = **250000**

Enron Email

No of Nodes = **36692**
No of Links = **183831**

Evaluation Tasks

Network Reconstruction

Metric: **AUC**

Node Classification

Metric: **Macro-F1**, **Micro-F1**

Link Prediction

Metric: **AUC**

Network Reconstruction: AUC Scores

	Louvain (Node2Vec)	Louvain (Stochastic)	Louvain (Community Embedding) [Method 1] Node2Vec	Louvain (Community Embedding) [Method 2] Node2Vec	Louvain (Community Embedding) [Method 3] Node2Vec	Louvain (Community Embedding) [Method 4] Node2Vec	HARP Node2Vec
karate	0.675	0.619	0.699	0.754	0.436	**0.833**	0.770
BlogCatalog3	0.657	0.602	0.623	0.669	0.368	**0.923**	0.544
Facebook	0.884	0.485	0.819	0.779	0.602	**0.962**	0.964
Enron Email	0.839	0.427	0.363	0.925	0.317	**0.942**	0.973
fb-CMU	0.697	0.701	0.621	0.712	0.493	**0.833**	0.829

Network Reconstruction: AUC Scores

	Louvain (DeepWalk)	Louvain (Stochastic)	Louvain (Community Embedding) [Method 1] Deepwalk	Louvain (Community Embedding) [Method 2] Deepwalk	Louvain (Community Embedding) [Method 3] Deepwalk	Louvain (Community Embedding) [Method 4] Deepwalk	HARP Deepwalk
karate	0.657	0.619	0.664	0.735	0.419	**0.801**	0.758
BlogCatalog3	0.627	0.602	0.566	0.651	0.507	**0.921**	0.527
Facebook	0.813	0.485	0.722	0.829	0.733	**0.918**	0.945
Enron Email	0.790	0.427	0.402	0.888	0.521	**0.945**	0.962
fb-CMU	0.683	0.701	0.566	0.715	0.594	**0.843**	0.821

Node Classification: fb-CMU

	Louvain (Node2Vec)	Louvain (Deepwalk)	Louvain (Stochastic)	Louvain (Community Embedding) [Method 2] Node2Vec	Louvain (Community Embedding) [Method 4] Node2Vec	Louvain (Community Embedding) [Method 2] Deepwalk	Louvain (Community Embedding) [Method 4] Deepwalk	HARP
Micro-F1	0.535	0.533	**0.560**	0.547	0.558	0.539	0.533	0.555
Macro-F1	0.275	0.248	0.257	**0.278**	0.249	0.258	0.252	0.244

Node Classification: BlogCatalog

	Louvain (Node2Vec)	Louvain (Deepwalk)	Louvain (Stochastic)	Louvain (Community Embedding) [Method 2] Node2Vec	Louvain (Community Embedding) [Method 4] Node2Vec	Louvain (Community Embedding) [Method 2] Deepwalk	Louvain (Community Embedding) [Method 4] Deepwalk	HARP
Micro-F1	0.427	0.344	0.350	0.434	0.411	0.391	**0.453**	0.304
Macro-F1	0.209	0.172	0.151	0.219	0.189	0.179	**0.244**	0.134

Link Prediction: AUC Score (40% pruning)

	Louvain (Node2Vec)	Louvain (Deepwalk)	Louvain (Stochastic)	Louvain (Community Embedding) [Method 2] Node2Vec	Louvain (Community Embedding) [Method 4] Node2Vec	Louvain (Community Embedding) [Method 2] Deepwalk	Louvain (Community Embedding) [Method 4] Deepwalk	HARP
karate	**0.7236**	0.6521	0.4868	0.6053	0.6447	0.5791	0.6348	0.7634
BlogCatalog3	0.6426	0.6345	0.5225	0.6994	**0.8668**	0.6783	0.8561	0.8363
Facebook	**0.7578**	0.6702	0.6529	0.7169	0.7376	0.6912	0.7283	0.7748
Enron Email	0.7261	0.7016	0.6637	0.6785	**0.8374**	0.6551	0.8256	0.7859
fb-CMU	0.6711	0.6351	0.5764	0.6544	0.7252	0.5764	**0.7265**	0.6901

Conclusions

- **Louvain (Community Embedding) Method 4** outperforms other variations of the algorithm in most of the tasks
- Louvain NE using standard embedding performed better in link prediction for smaller graphs (karate, facebook), however **Method 4 outperformed it for larger graphs**
- Enhanced Performance of Method 4 can be attributed to the addition of the **second d-dimensional embedding** that captures the connectivity of two connected nodes from different communities

Future Work

- Hard assignment of nodes to communities: Louvain segregates the nodes so that each node belongs to one community. But if an algorithm (for example, CESNA) allows overlapping communities, then the embedding algorithm needs to be changed. That is one possible future research work.

- Currently, we have evaluated the methods on size up to ~40,000 due to time and memory constraints. We need to test the method on larger graphs of size in order of million nodes.

THANK YOU!

Network Reconstruction: AUC

- Consider all connected node pairs and randomly choose equal number of disconnected node pairs in the actual graph
- The similarity between the selected node pairs is computed by taking the inner product between embedding vectors
- AUC is the fraction of times the connected node pairs have greater similarity than the disconnected ones

Conclusions from AUC

- Louvain (Community Embedding) Method 4 was seen to perform better than Method 1, Method 2 and Method 3 for both Node2Vec and Deepwalk. Method 2 also showed promising results
- Louvain (Community Embedding) Method 4 was better than the naive method for all the graphs.
- For some graphs for Method 2, Node2Vec was better and for others, Deepwalk was better. Hence, we need to consider both of them for the following experiments.

Node Classification

- We apply the generated embedding vectors as features in a supervised learning framework to classify a node into its corresponding class
- We use Chain of Binary Classifiers(*Logistic Regression Model*) for multi-label classification
- 80% of the vertices are used as training data and the rest 20% is the test data
- Macro-F1 and Micro-F1 scores of each algorithm is reported

Conclusions from Node Classification

- For a relatively smaller graphs(fb-CMU), node classification results of Louvain(Community Embedding) Method 2 were comparable with the naive methods.

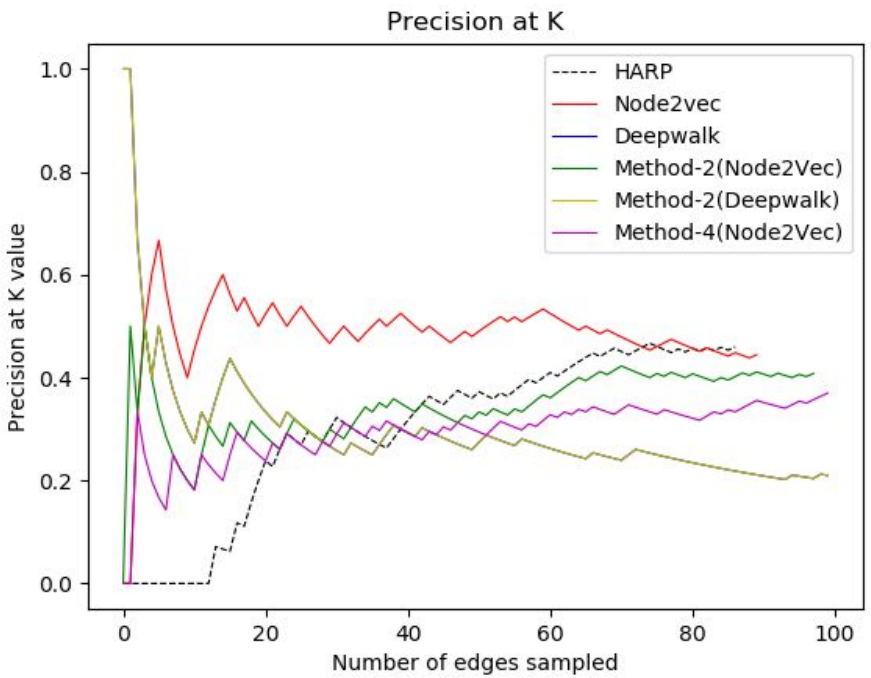
- For a larger graph(BlogCatalog), Louvain(Community Embedding) Method 2 out-performed all the other methods closely followed by Louvain(Node2Vec) and Louvain(Community Embedding) Method 4

Network Reconstruction: Precision@K

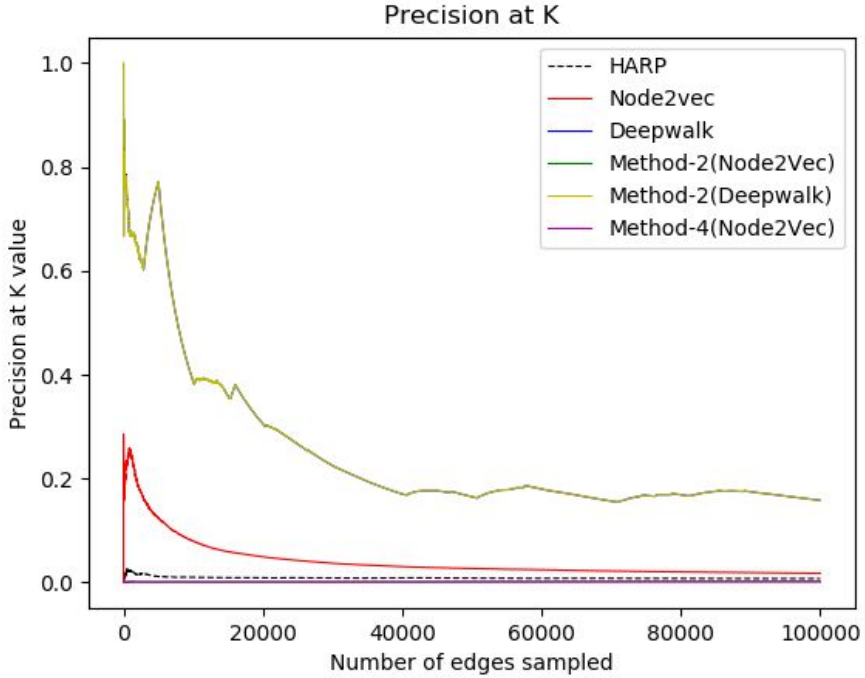
- We take all possible pairs of nodes and rank them based on the Euclidean distance between their embedding vectors
- The top K node pairs are considered and their accuracy is checked based on the actual links in the graph.

Precision @ K Graph

karate

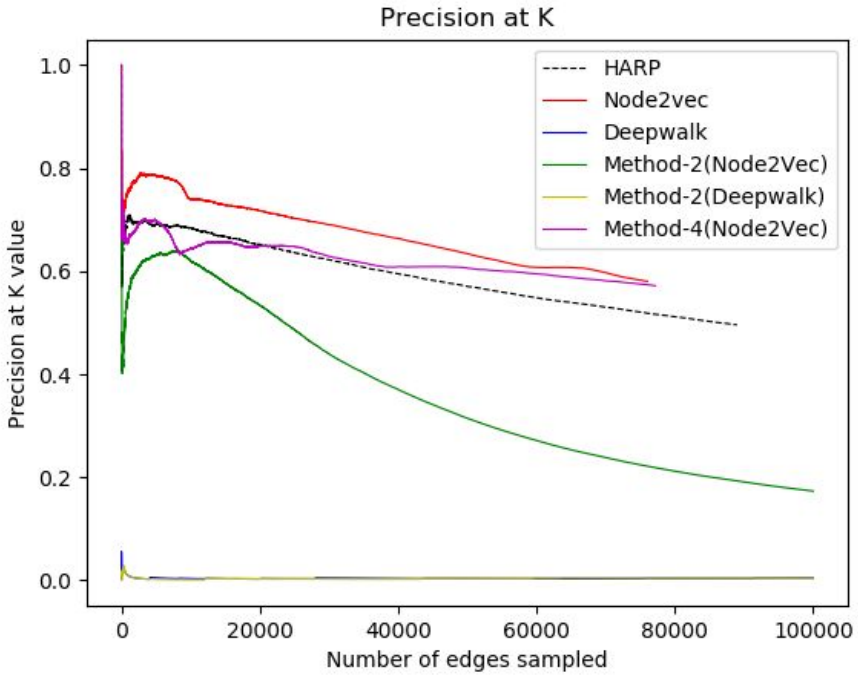


BlogCatalog3

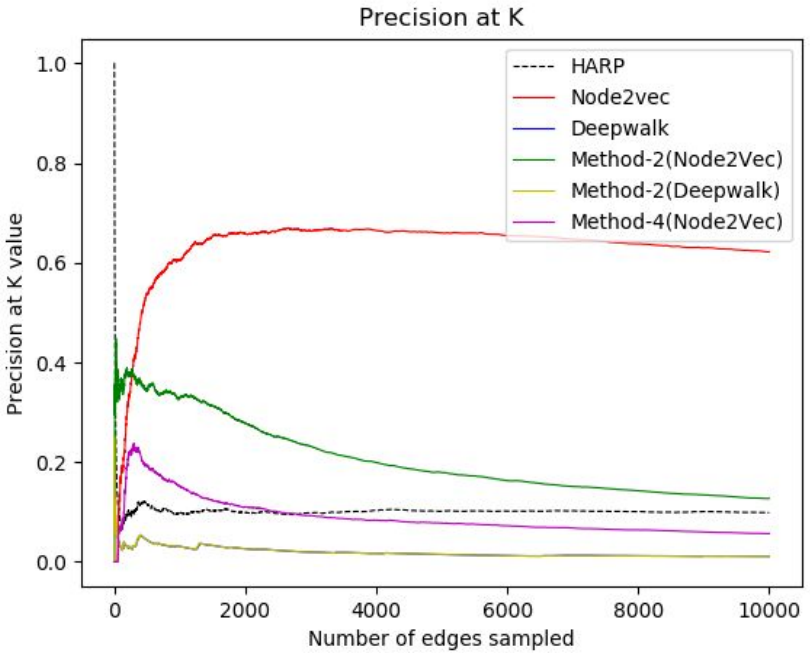


Precision @ K Graph

Facebook

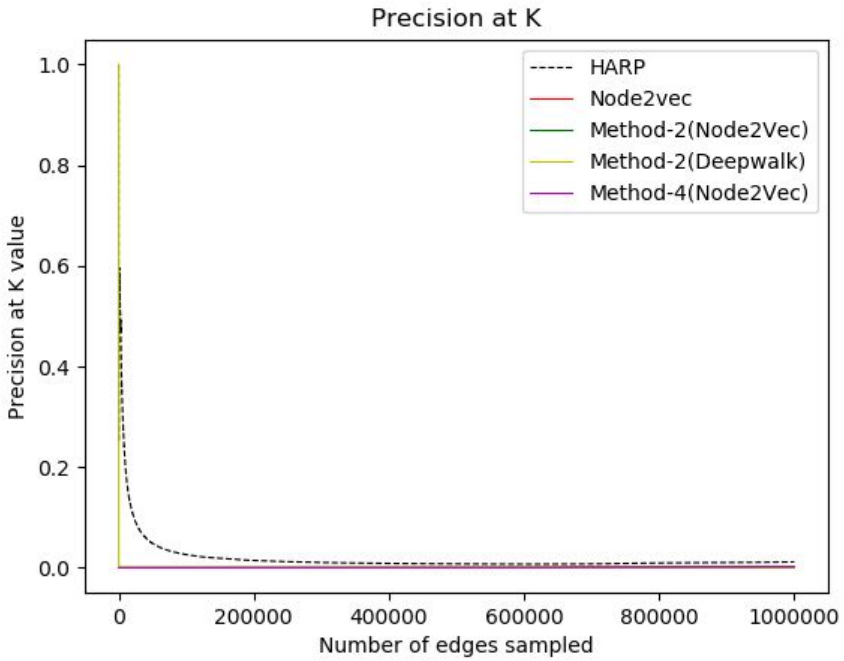


fb-CMU



Precision @ K Graph

Enron Email



Link Prediction

- Firstly, we prune a fixed number of edges from the graph. Here, we prune 40% of the edges
- From the embedding generated for the pruned graph, we predict the new edges that can be formed based on the pruned graph and then compare with the original graph
- AUC score of link prediction has been reported

Conclusions from Link Prediction

- Louvain (Community Embedding) Method 4 out-performed all the other algorithms for larger graphs. This can be attributed to the addition of the second d-dimensional embedding that captures the similarity of two connected nodes in different communities

- For smaller graphs, the results were close with Louvain(Node2Vec) defeating the other variations of the algorithm