BOISE STATE UNIVERSITY

# Structural Graph Representation Learning for Cybersecurity Applications

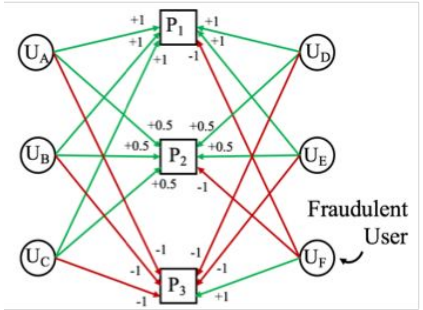
Edoardo Serra
edoardoserra@boisestate.edu

# Outline

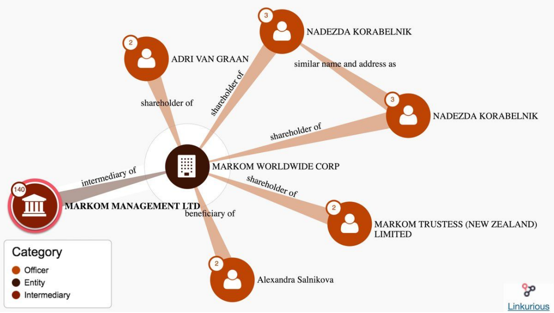
- Motivations
- Unsupervised Structural Representation Learning Procedures
- Applications for cybersecurity
- Conclusion

# Motivations

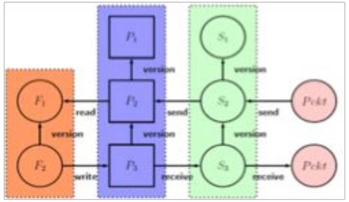
# Graphs are Ubiquitous and used for Information Integration



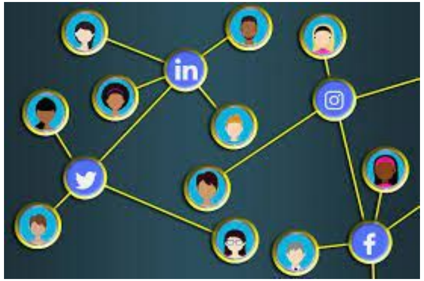
REV2: Kumar et al. 2018
Detecting Malicious user by the rating of other users



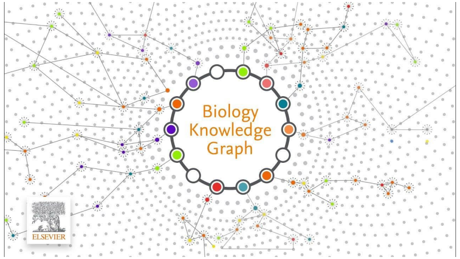
Offshore Leaks networks

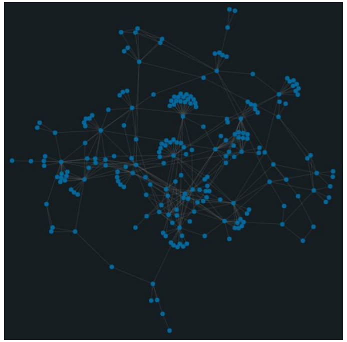


(a) CamFlow [8] Provenance subgraph. Nodes types are Processes, Files and Sockets with subscripts representing version. Edges represent actions between nodes.

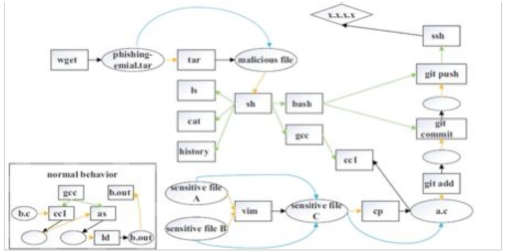


Social Networks



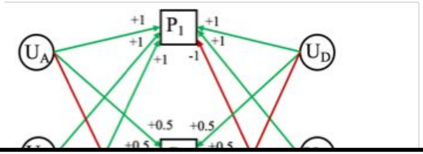


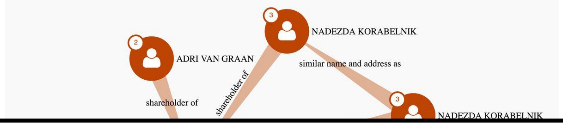
MalNet

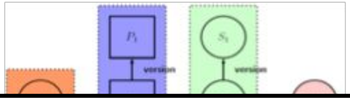


(b) Provenance graph for a theoretical phishing attack proposed by the authors of Paradise [9]. Normal behavior is shown in the dotted box. Edge types are represented with different colors.

# Graphs are Ubiquitous and used for Information Integration





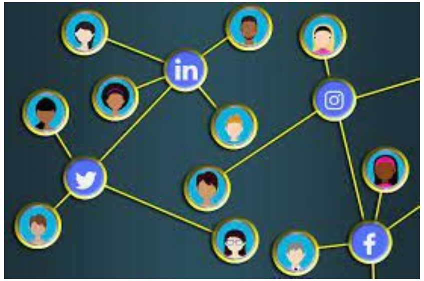


- Graphs  can be directed  or undirected
- Graph can have features on edges and vertices
- Graph can have temporal attributes
- Graph can be Spatio Temporal

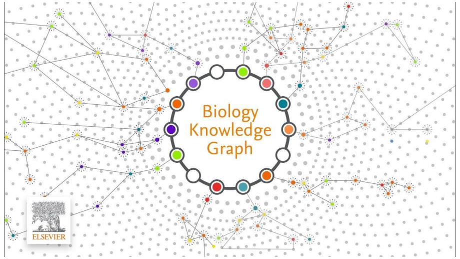
Detecting malicious user by the rating of other users

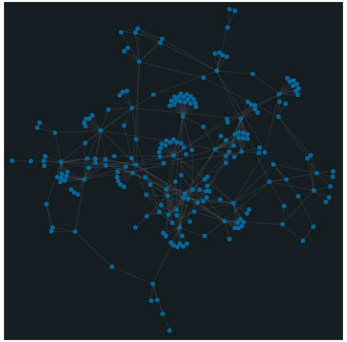
Offshore Leaks networks

s are ting

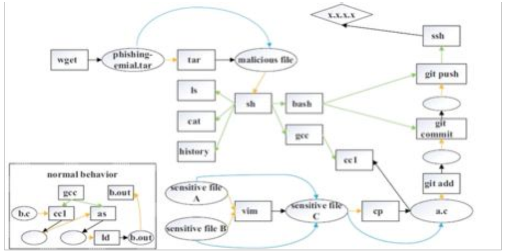


Social Networks





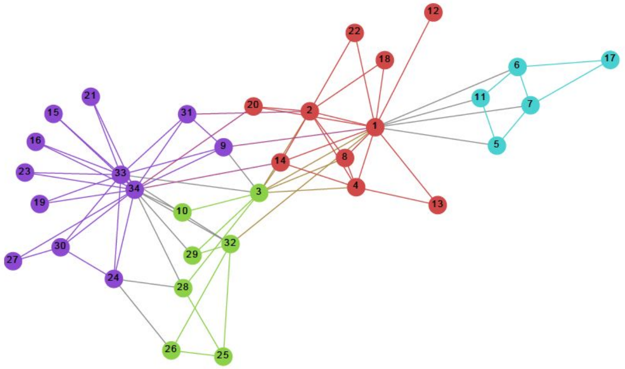
MalNet



(b) Provenance graph for a theoretical phishing attack proposed by the authors of Paradise [9]. Normal behavior is shown in the dotted box. Edge types are represented with different colors.

# Graph Representation Learning

- Convert graph nodes into numerical vectorial representations
  - Automatic feature-engineering
  - Maximize the amount of encoded information
- High volume of research works, high interest in applications



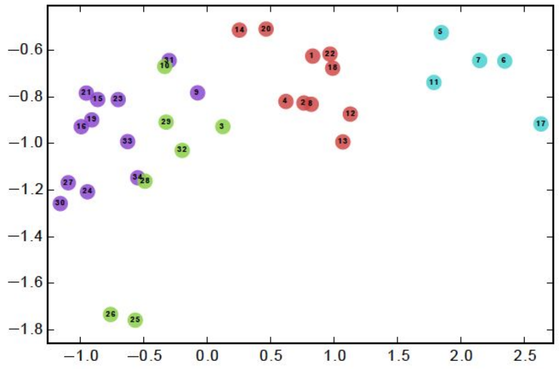
Karate network

$N_1$ = [0.8,-0.6]
$N_2$ = [0.7,-0.8]
$N_3$ = [0.1,-0.9]

⋮

$N_{32}$ = [-0.3,-1.0]
$N_{33}$ = [-0.6,-1.0]
$N_{34}$ = [-0.6,-1.2]

Vectorial representations

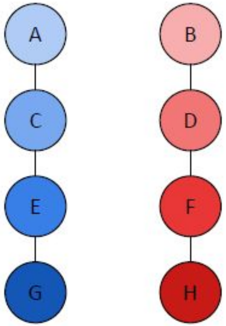


Latent space

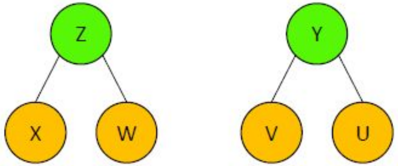
# Graph Representation Learning

- Two main representation types:
    - Connectivity-based methods:
        - Encode nodes connectivity information
        - The shorter the path between two nodes, more similar the generated representation
    - Structure-based methods:
        - Encode the node’s task inside the network, e.g:
            - Inside the network topology:
                - Star-center
                - Bridge node
            - Group administrator in a social network
            - Hubs and authorities ins the World Wide Web
        - The closer the role, more similar the representation

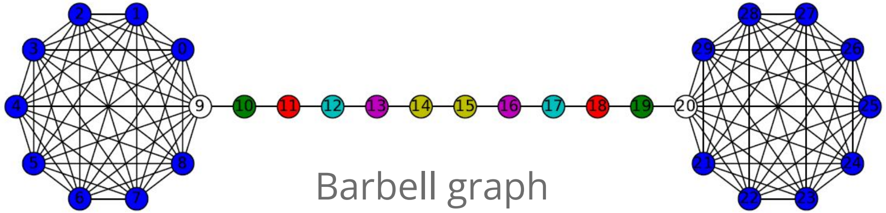
Connectivity-based



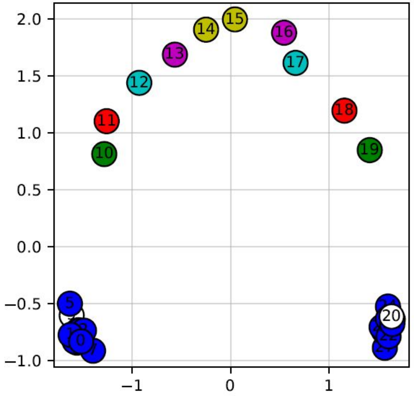
Structure-based



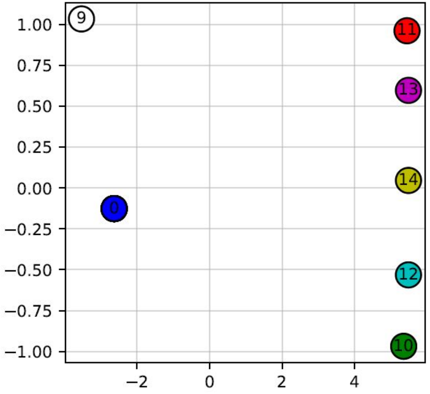
# Connectivity VS Structure



Barbell graph

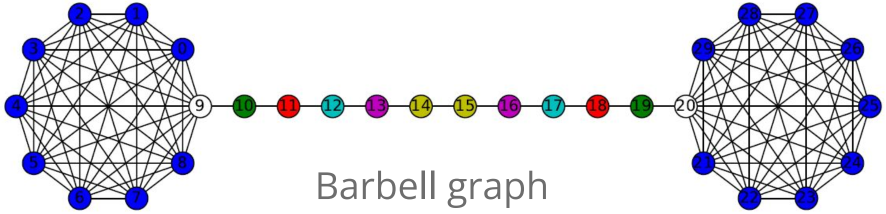


Connectivity-based

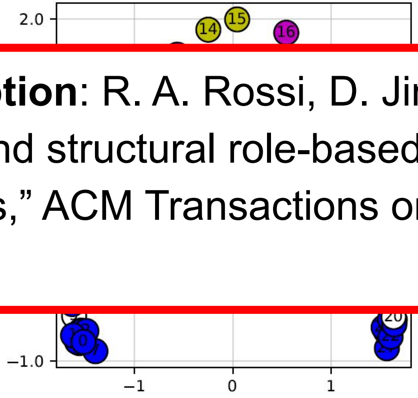


Structure-based

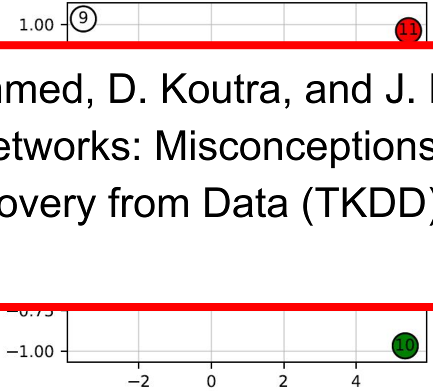
# Connectivity VS Structure



Barbell graph



Connectivity-based



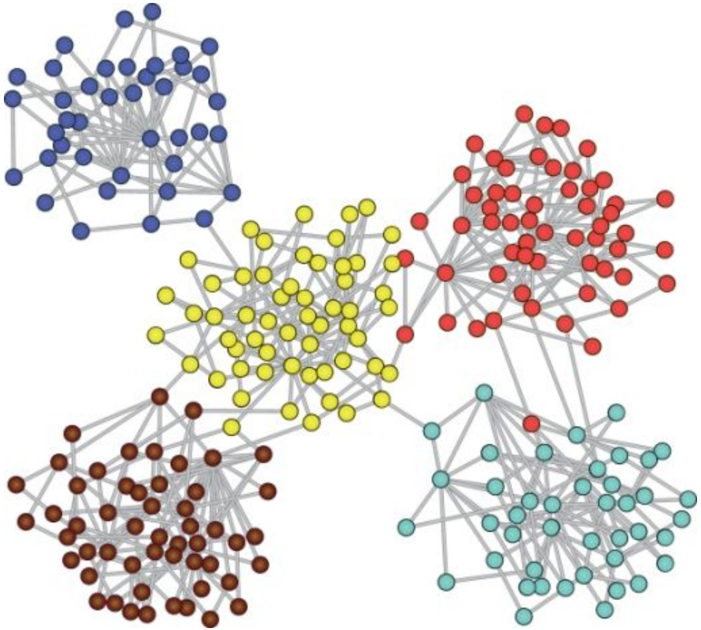
Structure-based

**Misconception**: R. A. Rossi, D. Jin, S. Kim, N. K. Ahmed, D. Koutra, and J. B. Lee, “On proximity and structural role-based embeddings in networks: Misconceptions, techniques, and applications,” ACM Transactions on Knowledge Discovery from Data (TKDD), vol. 14, no. 5, pp. 1–37,2020

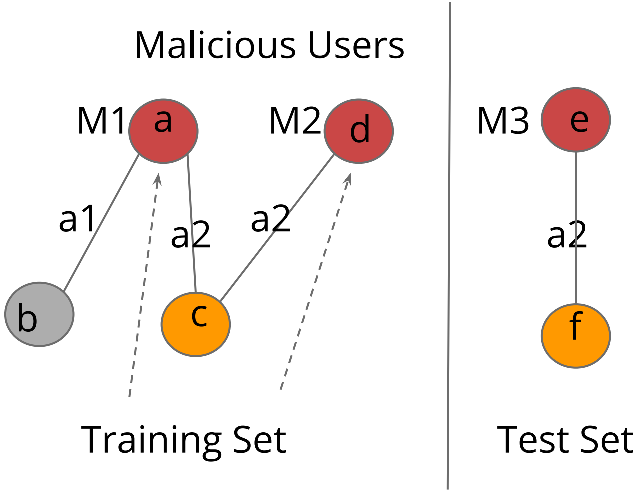
# Connectivity and Proximity for Social Networks

- Many data available

- Connectivity solve this task:
  - Community Detection
  - Link Prediction



# Connectivity and Proximity Problem



- **Classification concept**: all the red nodes are malicious because they perform a2 on the orange node
- What can be learned with connectivity in the training set ?
  - **Malicious user are connected (see M1 and M2)**
  - Result: **M3 is not a malicious because not connected to M1 and M2**
- Problem of cold start users
- Connectivity and Proximity does not generalize well.

# From Neural Networks to Graph Neural Networks

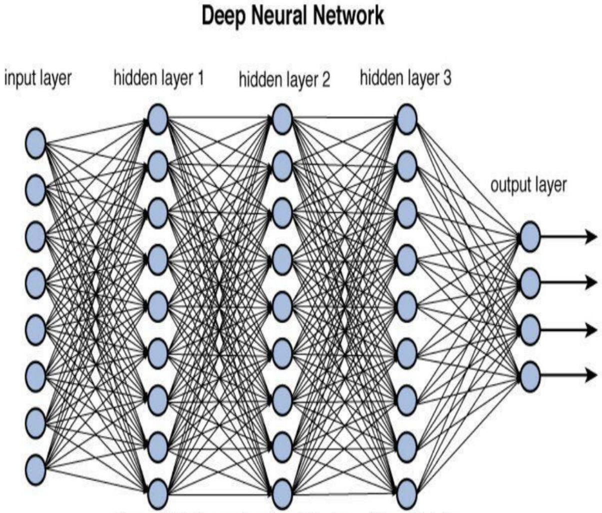
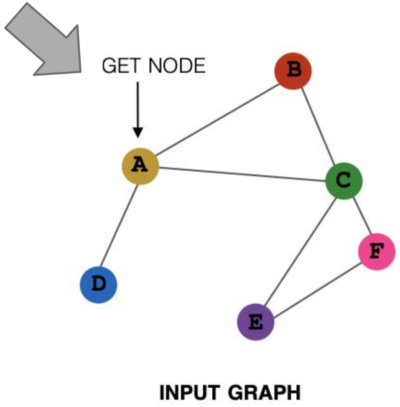
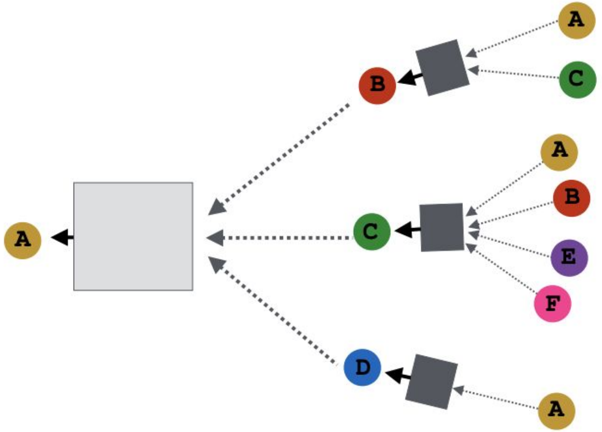


Figure 12.2 Deep network architecture with multiple layers.

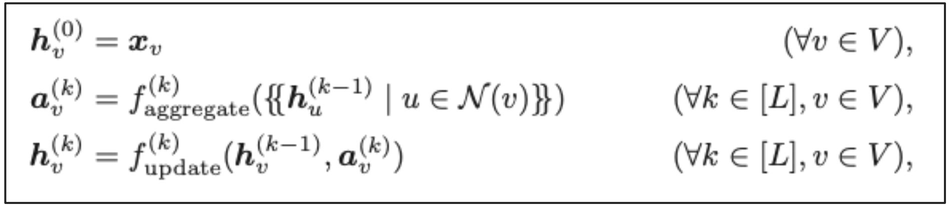


Graph Neural Network



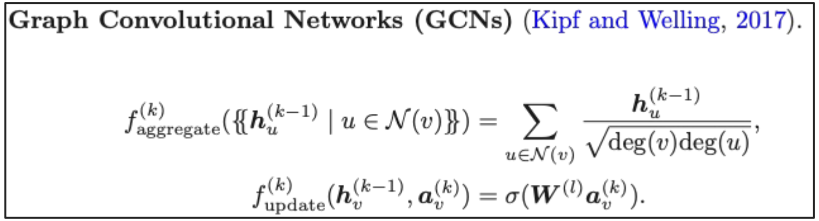
# GNNs and Message Passing (Sato, 2020)

- Message Passing



$$\begin{aligned}
\boldsymbol{h}_v^{(0)} &= \boldsymbol{x}_v && (\forall v \in V),\\
\boldsymbol{a}_v^{(k)} &= f_{\text{aggregate}}^{(k)}(\{\!\{\boldsymbol{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\!\}) && (\forall k \in [L], v \in V),\\
\boldsymbol{h}_v^{(k)} &= f_{\text{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) && (\forall k \in [L], v \in V),
\end{aligned}$$

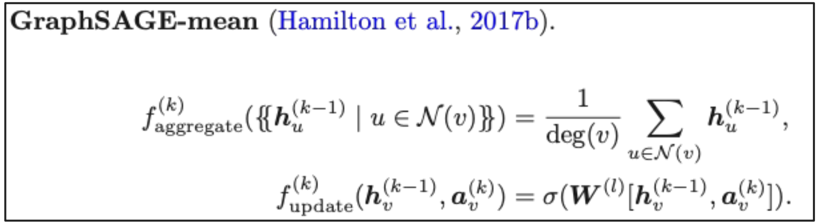
- GNNs



**Graph Convolutional Networks (GCNs)** (Kipf and Welling, 2017).

$$f_{\text{aggregate}}^{(k)}(\{\!\{\boldsymbol{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\!\}) = \sum_{u \in \mathcal{N}(v)} \frac{\boldsymbol{h}_u^{(k-1)}}{\sqrt{\deg(v)\deg(u)}},$$

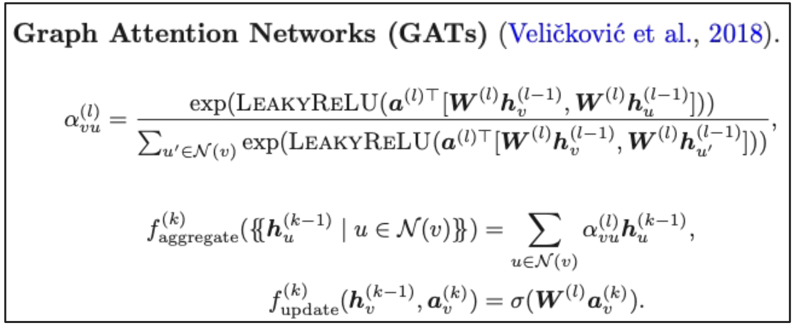
$$f_{\text{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) = \sigma(\boldsymbol{W}^{(l)}\boldsymbol{a}_v^{(k)}).$$



**GraphSAGE-mean** (Hamilton et al., 2017b).

$$f_{\text{aggregate}}^{(k)}(\{\!\{\boldsymbol{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\!\}) = \frac{1}{\deg(v)} \sum_{u \in \mathcal{N}(v)} \boldsymbol{h}_u^{(k-1)},$$

$$f_{\text{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) = \sigma(\boldsymbol{W}^{(l)}[\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}]).$$



**Graph Attention Networks (GATs)** (Veličković et al., 2018).

$$\alpha_{vu}^{(l)} = \frac{\exp(\textsc{LeakyReLU}(\boldsymbol{a}^{(l)\top}[\boldsymbol{W}^{(l)}\boldsymbol{h}_v^{(l-1)}, \boldsymbol{W}^{(l)}\boldsymbol{h}_u^{(l-1)}]))}{\sum_{u' \in \mathcal{N}(v)} \exp(\textsc{LeakyReLU}(\boldsymbol{a}^{(l)\top}[\boldsymbol{W}^{(l)}\boldsymbol{h}_v^{(l-1)}, \boldsymbol{W}^{(l)}\boldsymbol{h}_{u'}^{(l-1)}]))},$$

$$f_{\text{aggregate}}^{(k)}(\{\!\{\boldsymbol{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\!\}) = \sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(l)}\boldsymbol{h}_u^{(k-1)},$$

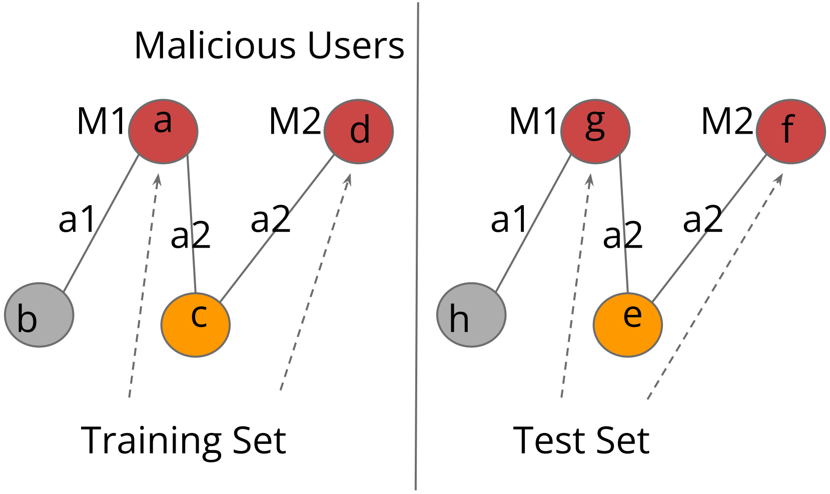
$$f_{\text{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) = \sigma(\boldsymbol{W}^{(l)}\boldsymbol{a}_v^{(k)}).$$

# Connectivity with Message Passing

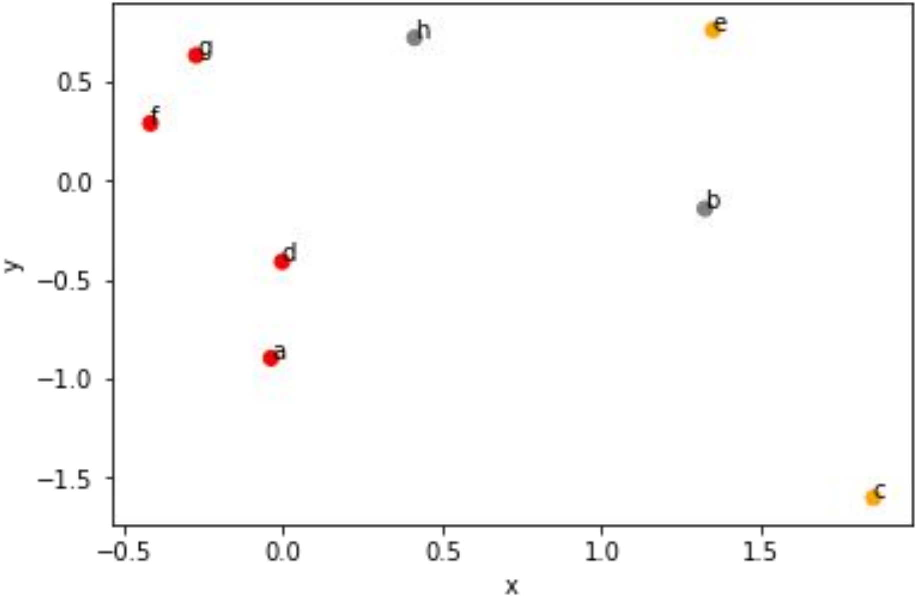
$$h_u^{(0)} \ initialized\ randomly$$

$$a_v^{(k)} = mean(\{\{h_u^{(k-1)} | (v,u) \in E\}\})$$

$$h_v^{(k)} = 0.5 * h_v^{(k-1)} + 0.5 * a_v^{(k)})$$



Random Initialization

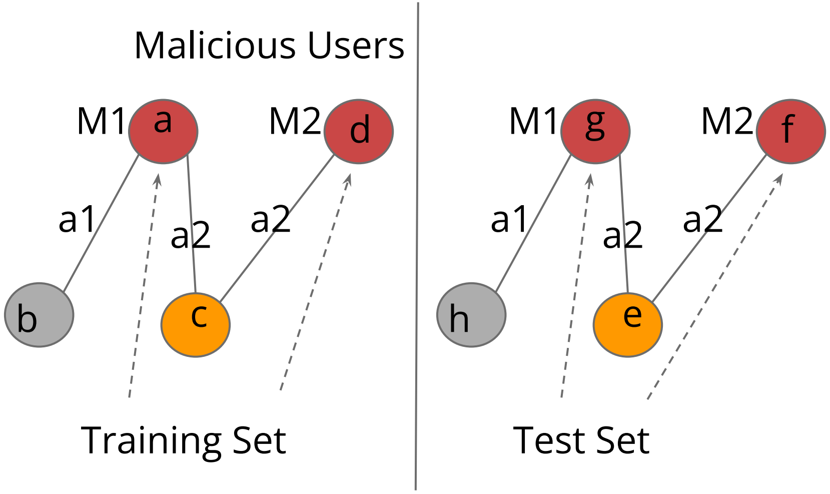


# Connectivity with Message Passing

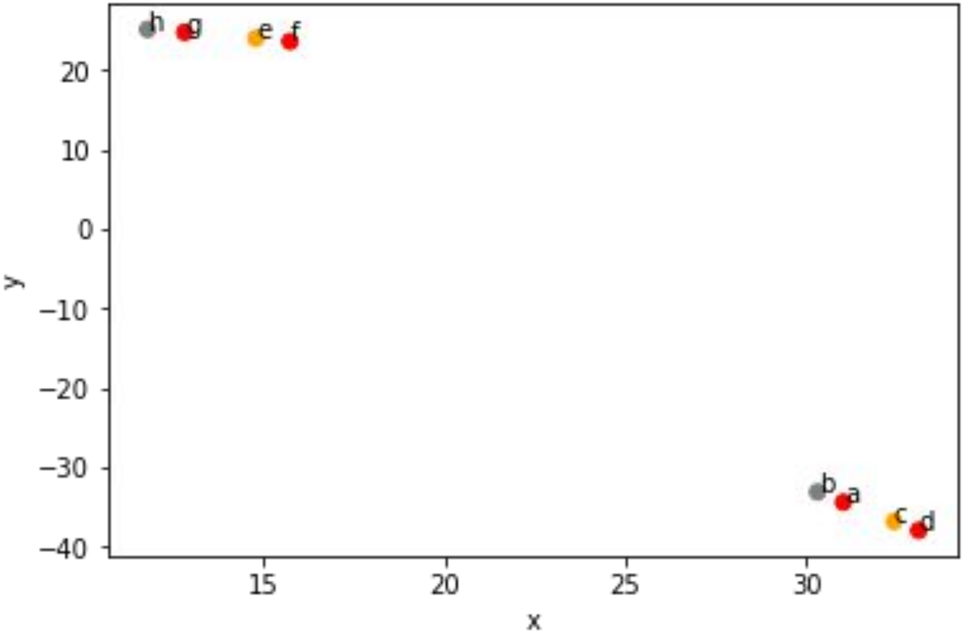
$$h_u^{(0)}\ initialized\ randomly$$

$$a_v^{(k)} = mean(\{\{h_u^{(k-1)} | (v,u) \in E\}\})$$

$$h_v^{(k)} = 0.5 * h_v^{(k-1)} + 0.5 * a_v^{(k)})$$



After 8 levels



# Why not just Connectivity?

- Connectivity -> Homophily principle
- Connectivity -> Models do not learn general structural patterns
- Connectivity ->Can be achieved with GNNs
- Connectivity ->It produces overfitting especially with supervised training

Structural Representation Learning  allow to learn general structural patterns.

# State-of-the-art (Unsupervised)

| Model | Connectivity-based | Structure-based | Neural Network |
|---|---|---|---|
| DeepWalk | Yes | No | No |
| LINE | Yes | No | No |
| Node2vec | Yes | No | No |
| GraphSAGE | Yes | No | Yes |
| ARGA | Yes | No | Yes |
| VGAE | Yes | No | Yes |
| SDNE | Yes | No | Yes |
| Struc2vec | No | Yes | No |
| GraphWave | No | Yes | No |
| DRNE | No | Yes | Yes |

# State-of-the-art (Supervised)

| Model | Connectivity-based | Structure-based | Neural Network |
|---|---|---|---|
| GIN | Yes | Yes | Yes |
| GCN | Yes | -- | Yes |
| GAT | Yes | -- | Yes |

More …

# Motivation:

- Limited literature.
  - Many connectivity-based works (DeepWalk,LINE,Node2vec,GraphSAGE,ARGA,...)
  - Few structure-based (Struc2vec,GraphWave,DRNE)
- Limitations of the existing methods.
  - No theoretical guarantees (Struc2vec,DRNE)
  - Computational cost (Struc2vec,GraphWave)

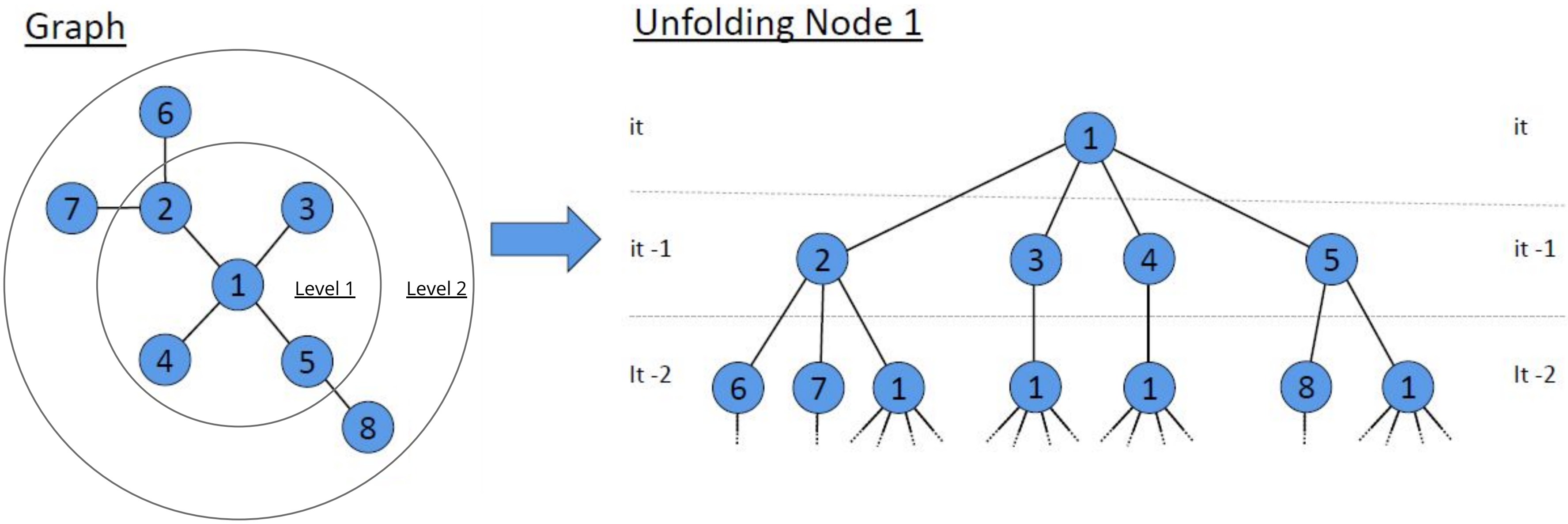
# Unsupervised Structural Representation Learning Procedures

# Proposed methods

- Three unsupervised structural graph representation learning methods
  - **SIR-GN**: Structural Iterative Representation learning approach for Graph Nodes
  - **SILA**: Structural Iterative Lexicographic Autoencoded Node Representation
  - **SparseStruct**: Sparse Structural Node Representation
- Iterative approaches
- Improve upon current state-of-the-art:
  - Comparable or better performance
  - Better computational cost with plenty of possibility for extension
  - Theoretical guarantees
  - Experimental framework
  - No gradient vanishing problem
  - More interpretable

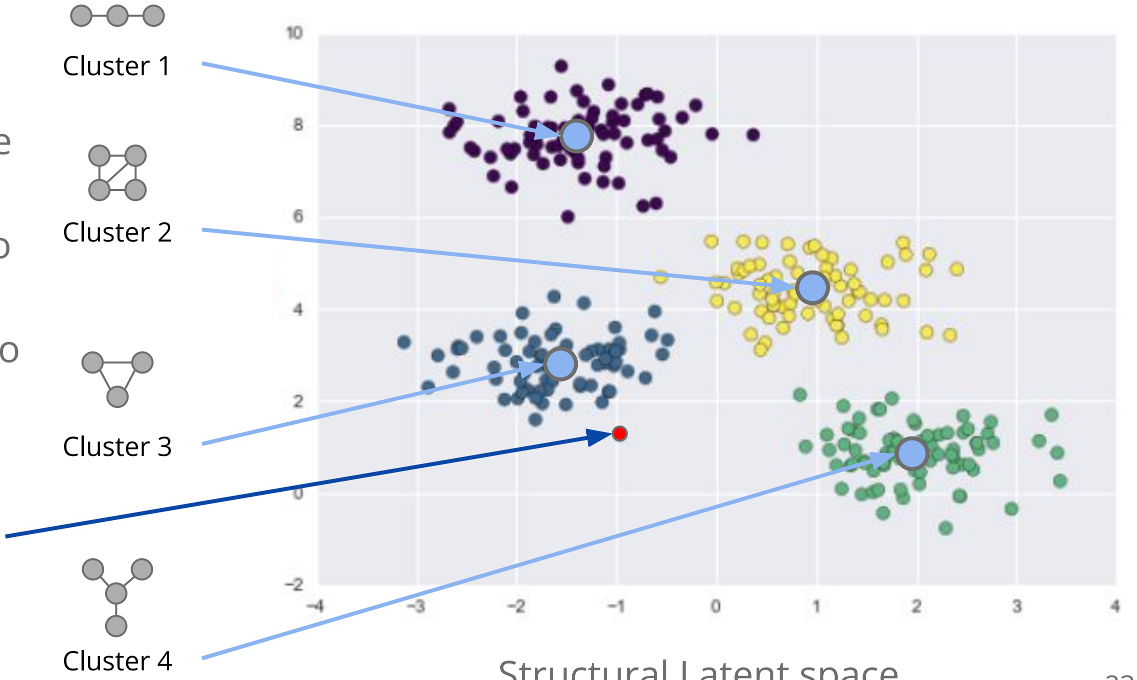
# Why iteration?

- Iteration + neighborhood aggregation:



# SIR-GN: Structural Iterative Representation learning approach for Graph Nodes

- Vertex description process:
  - Structural latent space
  - Clustering:
    - Main structures in the graph
  - Use the cluster centroids to describe each point
    - Distance from point to Cluster centroid

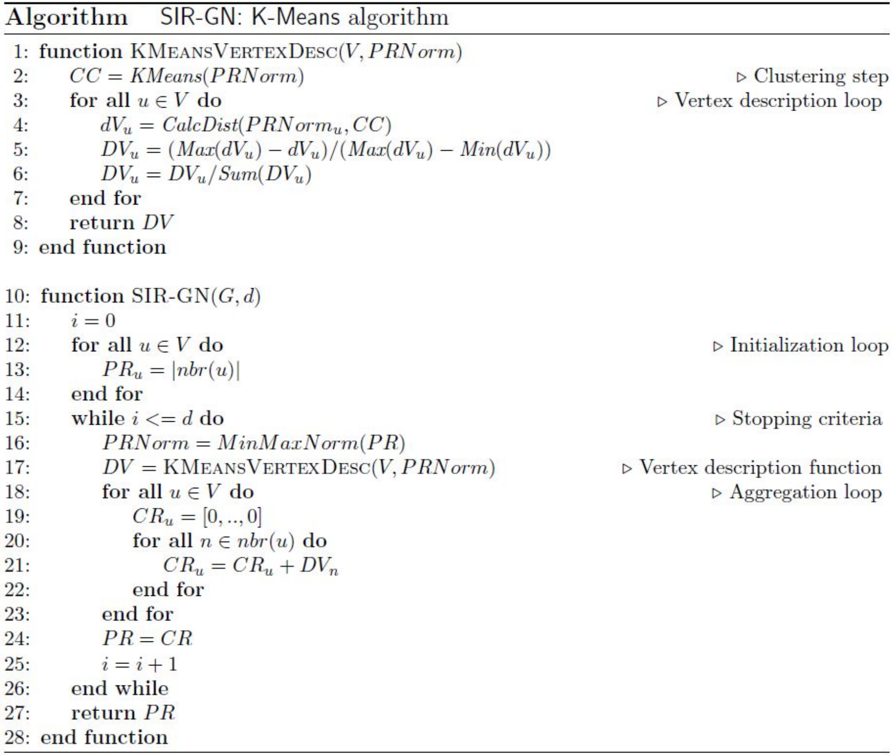


Euclidean distance = [7.0,3.7,1.5,3.0]

Structural Latent space

# SIR-GN: Structural Iterative Representation learning approach for Graph Nodes

- Initialization
- Iterative process: 2 steps
    - Vertex description:
        - Clustering:
            - GMM
            - K-Means
        - Description:
            - Normalized inverse distance from data point to each cluster
    - Neighborhood aggregation:
        - Aggregate structure non destructive way



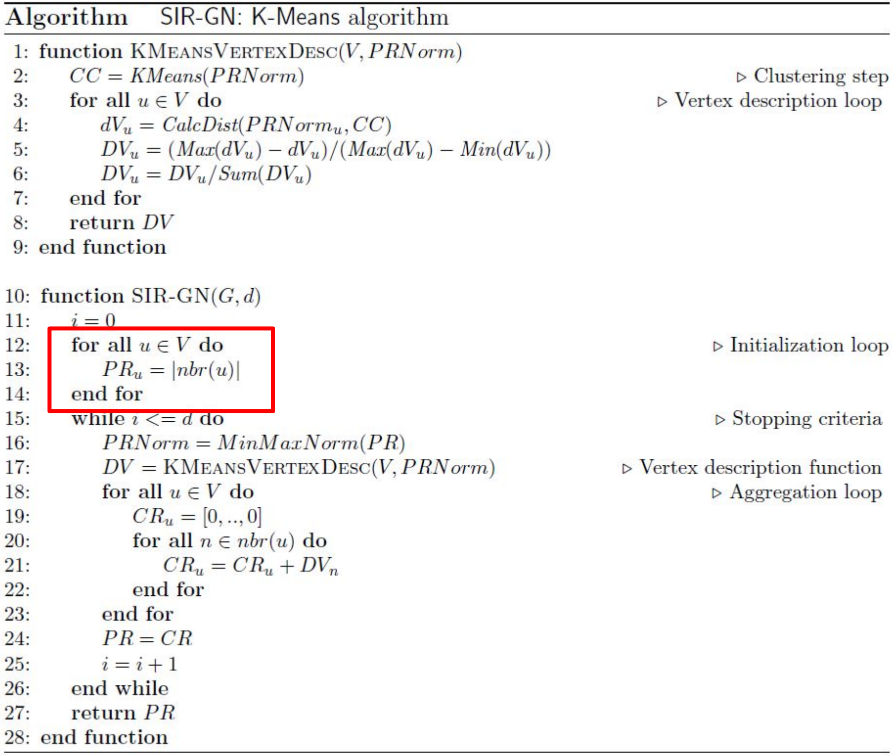
**Algorithm** SIR-GN: K-Means algorithm

```
function KMEANSVERTEXDESC($V, PRNorm$)
    $CC = KMeans(PRNorm)$                                  ▷ Clustering step
    for all $u \in V$ do                                   ▷ Vertex description loop
        $dV_u = CalcDist(PRNorm_u, CC)$
        $DV_u = (Max(dV_u) - dV_u)/(Max(dV_u) - Min(dV_u))$
        $DV_u = DV_u / Sum(DV_u)$
    end for
    return $DV$
end function

function SIR-GN($G, d$)
    $i = 0$
    for all $u \in V$ do                                   ▷ Initialization loop
        $PR_u = |nbr(u)|$
    end for
    while $i <= d$ do                                      ▷ Stopping criteria
        $PRNorm = MinMaxNorm(PR)$
        $DV$ = KMEANSVERTEXDESC($V, PRNorm$)               ▷ Vertex description function
        for all $u \in V$ do                               ▷ Aggregation loop
            $CR_u = [0, .., 0]$
            for all $n \in nbr(u)$ do
                $CR_u = CR_u + DV_n$
            end for
        end for
        $PR = CR$
        $i = i + 1$
    end while
    return $PR$
end function
```

# SIR-GN: Structural Iterative Representation learning approach for Graph Nodes

- Initialization
- Iterative process: 2 steps
  - Vertex description:
    - Clustering:
      - GMM
      - K-Means
    - Description:
      - Normalized inverse distance from data point to each cluster
  - Neighborhood aggregation:
    - Aggregate structure non destructive way



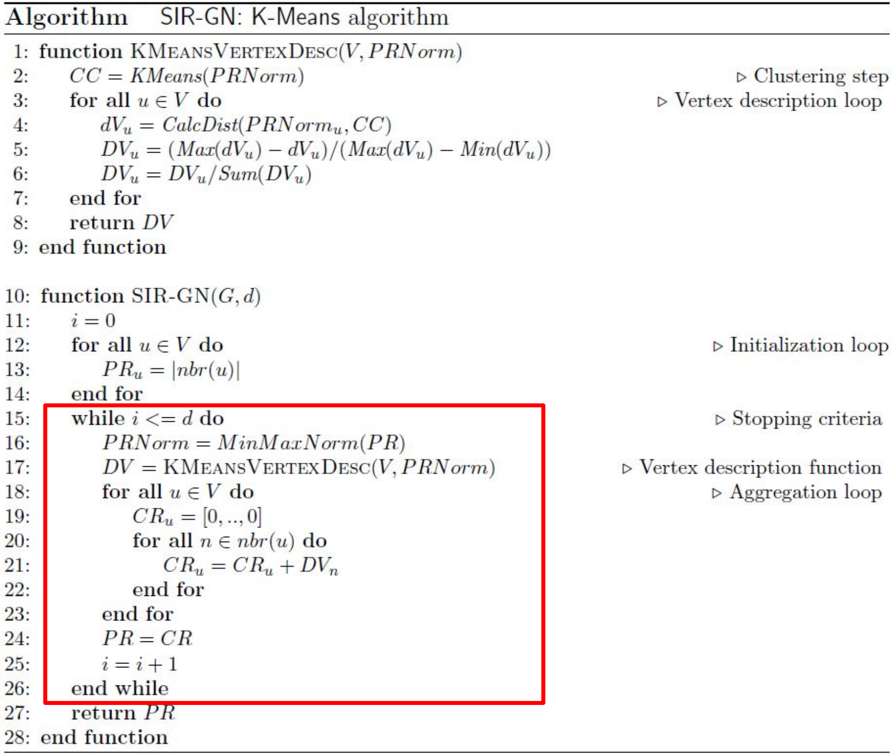
**Algorithm** SIR-GN: K-Means algorithm

```
function KMeansVertexDesc($V, PRNorm$)
    $CC = KMeans(PRNorm)$                                   ▷ Clustering step
    for all $u \in V$ do                                    ▷ Vertex description loop
        $dV_u = CalcDist(PRNorm_u, CC)$
        $DV_u = (Max(dV_u) - dV_u)/(Max(dV_u) - Min(dV_u))$
        $DV_u = DV_u/Sum(DV_u)$
    end for
    return $DV$
end function

function SIR-GN($G, d$)
    $i = 0$
    for all $u \in V$ do                                    ▷ Initialization loop
        $PR_u = |nbr(u)|$
    end for
    while $i <= d$ do                                       ▷ Stopping criteria
        $PRNorm = MinMaxNorm(PR)$
        $DV$ = KMeansVertexDesc($V, PRNorm$)                ▷ Vertex description function
        for all $u \in V$ do                                ▷ Aggregation loop
            $CR_u = [0, .., 0]$
            for all $n \in nbr(u)$ do
                $CR_u = CR_u + DV_n$
            end for
        end for
        $PR = CR$
        $i = i + 1$
    end while
    return $PR$
end function
```

# SIR-GN: Structural Iterative Representation learning approach for Graph Nodes

- Initialization
- Iterative process: 2 steps
  - Vertex description:
    - Clustering:
      - GMM
      - K-Means
    - Description:
      - Normalized inverse distance from data point to each cluster
  - Neighborhood aggregation:
    - Aggregate structure non destructive way



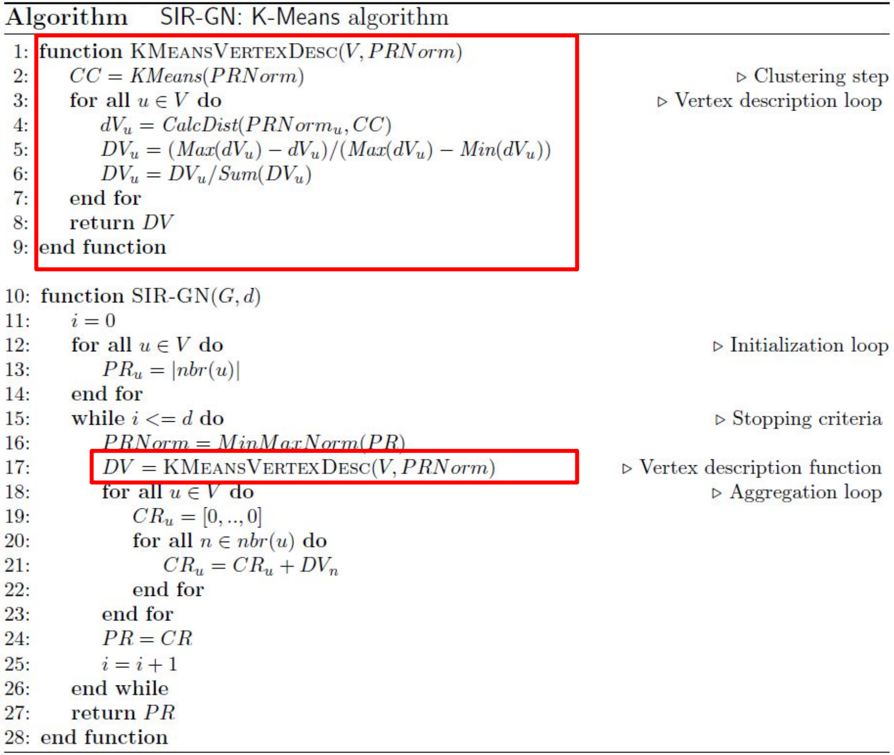
**Algorithm** SIR-GN: K-Means algorithm

```
function KMEANSVERTEXDESC($V, PRNorm$)
    $CC = KMeans(PRNorm)$                                   ▷ Clustering step
    for all $u \in V$ do                                    ▷ Vertex description loop
        $dV_u = CalcDist(PRNorm_u, CC)$
        $DV_u = (Max(dV_u) - dV_u)/(Max(dV_u) - Min(dV_u))$
        $DV_u = DV_u/Sum(DV_u)$
    end for
    return $DV$
end function

function SIR-GN($G, d$)
    $i = 0$
    for all $u \in V$ do                                    ▷ Initialization loop
        $PR_u = |nbr(u)|$
    end for
    while $i <= d$ do                                       ▷ Stopping criteria
        $PRNorm = MinMaxNorm(PR)$
        $DV$ = KMEANSVERTEXDESC($V, PRNorm$)                 ▷ Vertex description function
        for all $u \in V$ do                                ▷ Aggregation loop
            $CR_u = [0, .., 0]$
            for all $n \in nbr(u)$ do
                $CR_u = CR_u + DV_n$
            end for
        end for
        $PR = CR$
        $i = i + 1$
    end while
    return $PR$
end function
```

# SIR-GN: Structural Iterative Representation learning approach for Graph Nodes

- Initialization
- Iterative process: 2 steps
  - Vertex description:
    - Clustering:
      - GMM
      - K-Means
    - Description:
      - Normalized inverse distance from data point to each cluster
  - Neighborhood aggregation:
    - Aggregate structure non destructive way



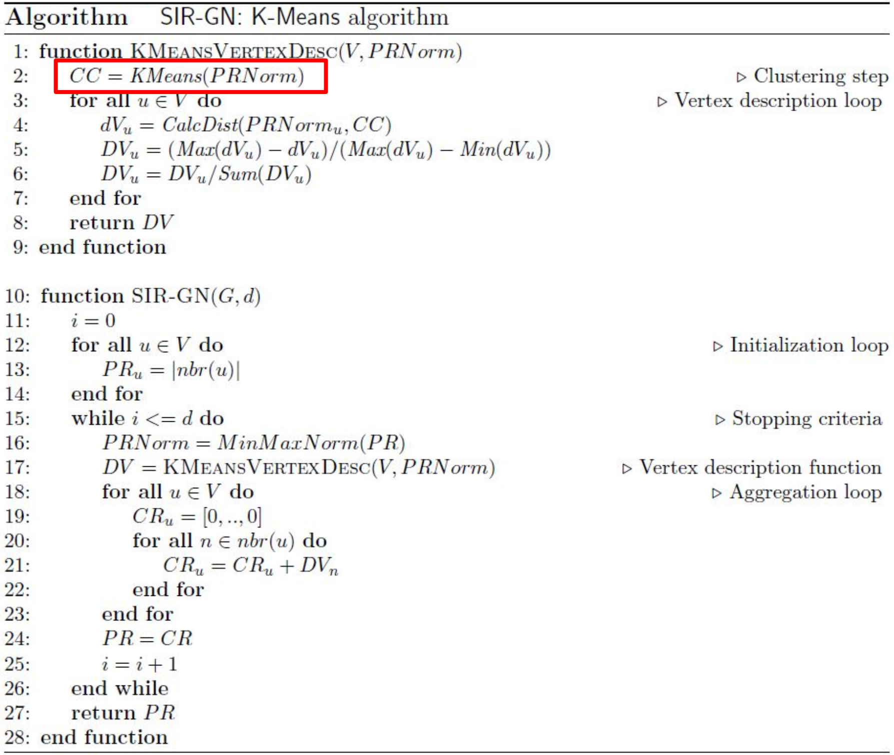
**Algorithm** SIR-GN: K-Means algorithm

```
function KMEANSVERTEXDESC($V, PRNorm$)
    $CC = KMeans(PRNorm)$                                          ▷ Clustering step
    for all $u \in V$ do                                           ▷ Vertex description loop
        $dV_u = CalcDist(PRNorm_u, CC)$
        $DV_u = (Max(dV_u) - dV_u)/(Max(dV_u) - Min(dV_u))$
        $DV_u = DV_u/Sum(DV_u)$
    end for
    return $DV$
end function

function SIR-GN($G, d$)
    $i = 0$
    for all $u \in V$ do                                           ▷ Initialization loop
        $PR_u = |nbr(u)|$
    end for
    while $i <= d$ do                                              ▷ Stopping criteria
        $PRNorm = MinMaxNorm(PR)$
        $DV$ = KMEANSVERTEXDESC($V, PRNorm$)                       ▷ Vertex description function
        for all $u \in V$ do                                       ▷ Aggregation loop
            $CR_u = [0, .., 0]$
            for all $n \in nbr(u)$ do
                $CR_u = CR_u + DV_n$
            end for
        end for
        $PR = CR$
        $i = i + 1$
    end while
    return $PR$
end function
```

# SIR-GN: Structural Iterative Representation learning approach for Graph Nodes

- Initialization
- Iterative process: 2 steps
  - Vertex description:
    - Clustering:
      - GMM
      - K-Means
    - Description:
      - Normalized inverse distance from data point to each cluster
  - Neighborhood aggregation:
    - Aggregate structure non destructive way



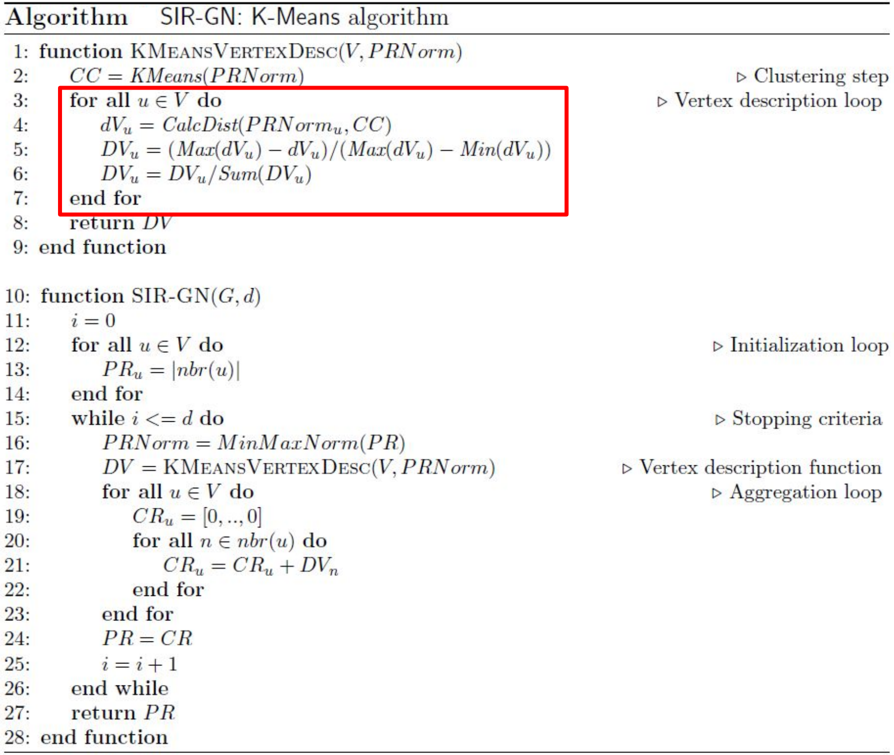
**Algorithm** SIR-GN: K-Means algorithm

```
function KMEANSVERTEXDESC($V, PRNorm$)
    $CC = KMeans(PRNorm)$                                        ▷ Clustering step
    for all $u \in V$ do                                          ▷ Vertex description loop
        $dV_u = CalcDist(PRNorm_u, CC)$
        $DV_u = (Max(dV_u) - dV_u)/(Max(dV_u) - Min(dV_u))$
        $DV_u = DV_u/Sum(DV_u)$
    end for
    return $DV$
end function

function SIR-GN($G, d$)
    $i = 0$
    for all $u \in V$ do                                          ▷ Initialization loop
        $PR_u = |nbr(u)|$
    end for
    while $i <= d$ do                                             ▷ Stopping criteria
        $PRNorm = MinMaxNorm(PR)$
        $DV$ = KMEANSVERTEXDESC($V, PRNorm$)                      ▷ Vertex description function
        for all $u \in V$ do                                      ▷ Aggregation loop
            $CR_u = [0, .., 0]$
            for all $n \in nbr(u)$ do
                $CR_u = CR_u + DV_n$
            end for
        end for
        $PR = CR$
        $i = i + 1$
    end while
    return $PR$
end function
```

# SIR-GN: Structural Iterative Representation learning approach for Graph Nodes

- Initialization
- Iterative process: 2 steps
  - Vertex description:
    - Clustering:
      - GMM
      - K-Means
    - Description:
      - Normalized inverse distance from data point to each cluster
  - Neighborhood aggregation:
    - Aggregate structure non destructive way



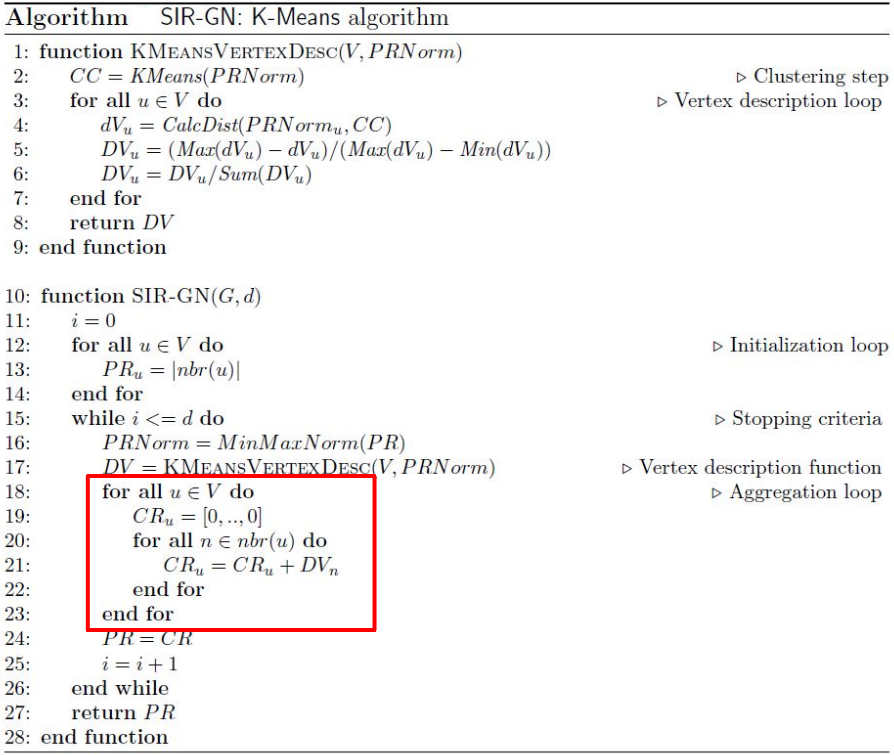
**Algorithm** SIR-GN: K-Means algorithm

```
function KMeansVertexDesc($V, PRNorm$)
    $CC = KMeans(PRNorm)$                                              ▷ Clustering step
    for all $u \in V$ do                                               ▷ Vertex description loop
        $dV_u = CalcDist(PRNorm_u, CC)$
        $DV_u = (Max(dV_u) - dV_u)/(Max(dV_u) - Min(dV_u))$
        $DV_u = DV_u/Sum(DV_u)$
    end for
    return $DV$
end function

function SIR-GN($G, d$)
    $i = 0$
    for all $u \in V$ do                                               ▷ Initialization loop
        $PR_u = |nbr(u)|$
    end for
    while $i <= d$ do                                                  ▷ Stopping criteria
        $PRNorm = MinMaxNorm(PR)$
        $DV$ = KMeansVertexDesc($V, PRNorm$)                           ▷ Vertex description function
        for all $u \in V$ do                                           ▷ Aggregation loop
            $CR_u = [0, .., 0]$
            for all $n \in nbr(u)$ do
                $CR_u = CR_u + DV_n$
            end for
        end for
        $PR = CR$
        $i = i + 1$
    end while
    return $PR$
end function
```

# SIR-GN: Structural Iterative Representation learning approach for Graph Nodes

- Initialization
- Iterative process: 2 steps
  - Vertex description:
    - Clustering:
      - GMM
      - K-Means
    - Description:
      - Normalized inverse distance from data point to each cluster
  - Neighborhood aggregation:
    - Aggregate structure non destructive way



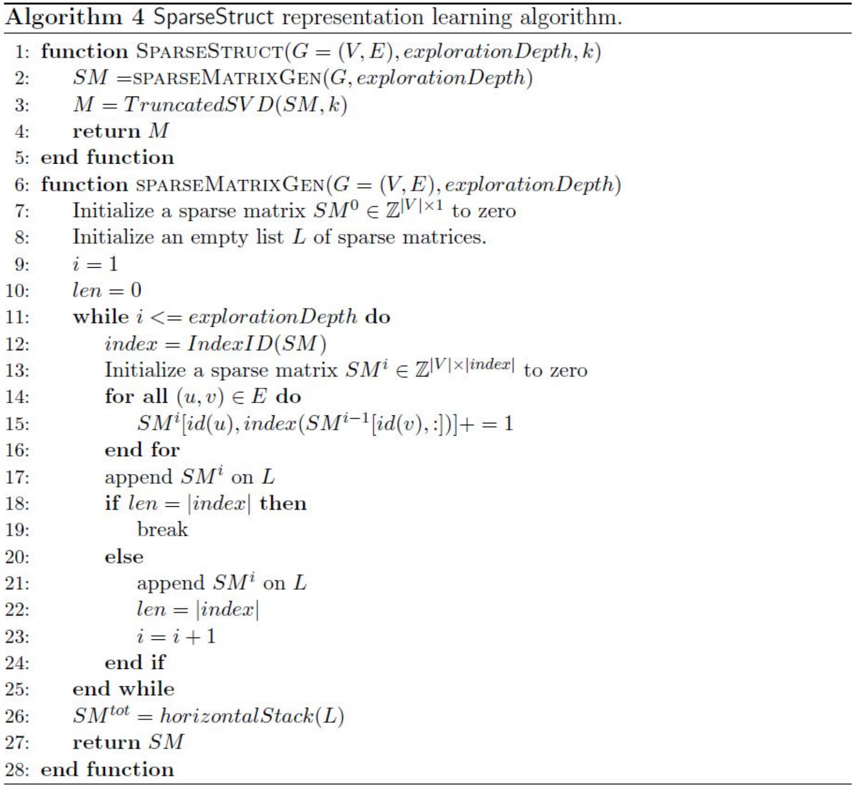
**Algorithm** SIR-GN: K-Means algorithm

```
function KMeansVertexDesc($V, PRNorm$)
    $CC = KMeans(PRNorm)$                                   ▷ Clustering step
    for all $u \in V$ do                                    ▷ Vertex description loop
        $dV_u = CalcDist(PRNorm_u, CC)$
        $DV_u = (Max(dV_u) - dV_u)/(Max(dV_u) - Min(dV_u))$
        $DV_u = DV_u/Sum(DV_u)$
    end for
    return $DV$
end function

function SIR-GN($G, d$)
    $i = 0$
    for all $u \in V$ do                                    ▷ Initialization loop
        $PR_u = |nbr(u)|$
    end for
    while $i <= d$ do                                       ▷ Stopping criteria
        $PRNorm = MinMaxNorm(PR)$
        $DV$ = KMeansVertexDesc($V, PRNorm$)                ▷ Vertex description function
        for all $u \in V$ do                                ▷ Aggregation loop
            $CR_u = [0, .., 0]$
            for all $n \in nbr(u)$ do
                $CR_u = CR_u + DV_n$
            end for
        end for
        $PR = CR$
        $i = i + 1$
    end while
    return $PR$
end function
```

# SparseStruct: Sparse Structural Node Representation

- Inspired on Weisfeiler-Lehman Isomorphism Test
- Iterative steps:
  - Generate Index
  - Initialization
  - Sparse vertex description
  - Concatenation
- Truncated SVD
- Theoretically proof:
  - Same as SIR-GN
  - Proof of convergence

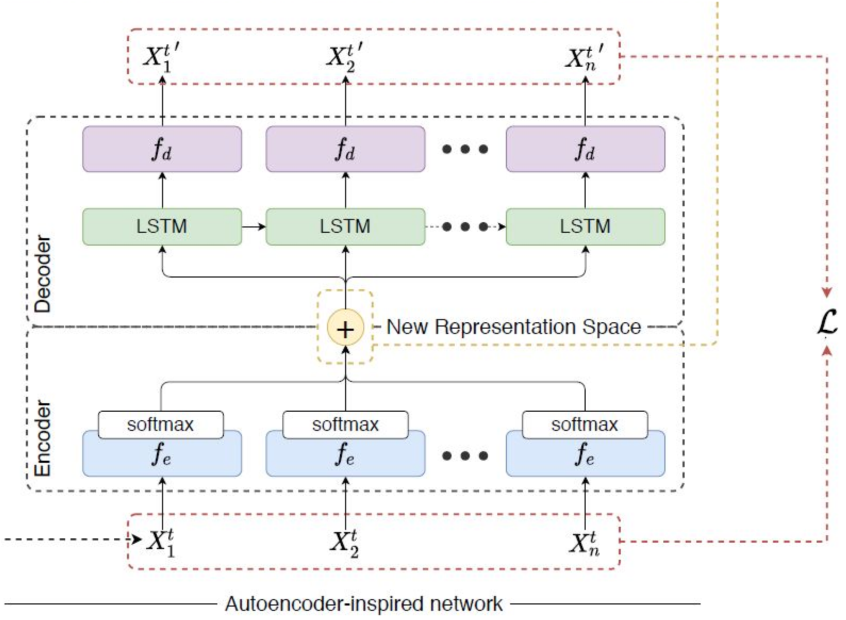


**Algorithm 4** SparseStruct representation learning algorithm.

```
function SPARSESTRUCT($G = (V, E), explorationDepth, k$)
    $SM$ =SPARSEMATRIXGEN($G, explorationDepth$)
    $M = TruncatedSVD(SM, k)$
    return $M$
end function
function SPARSEMATRIXGEN($G = (V, E), explorationDepth$)
    Initialize a sparse matrix $SM^0 \in \mathbb{Z}^{|V| \times 1}$ to zero
    Initialize an empty list $L$ of sparse matrices.
    $i = 1$
    $len = 0$
    while $i <= explorationDepth$ do
        $index = IndexID(SM)$
        Initialize a sparse matrix $SM^i \in \mathbb{Z}^{|V| \times |index|}$ to zero
        for all $(u, v) \in E$ do
            $SM^i[id(u), index(SM^{i-1}[id(v), :])] += 1$
        end for
        append $SM^i$ on $L$
        if $len = |index|$ then
            break
        else
            append $SM^i$ on $L$
            $len = |index|$
            $i = i + 1$
        end if
    end while
    $SM^{tot} = horizontalStack(L)$
    return $SM$
end function
```

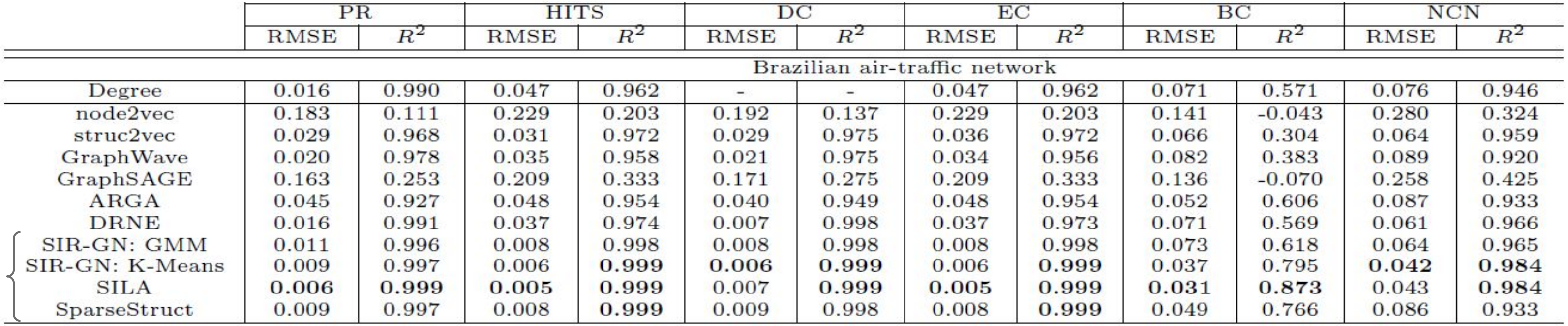
# SILA: Structural Iterative Lexicographic Autoencoded Node Representation

- Automatic vertex description:
  - Autoencoder neural network architecture
- Autoencoder architecture:
  - Input:
    - Representations of node neighborhoods
  - Transform to common space
  - Aggregation
  - Long Short-Term Memory
  - Minimize reconstruction error



# Results

- Latent information experiment:
    - Tested on 5 different graphs achieving the similar results



| | PR | | HITS | | DC | | EC | | BC | | NCN | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | RMSE | $R^2$ | RMSE | $R^2$ | RMSE | $R^2$ | RMSE | $R^2$ | RMSE | $R^2$ | RMSE | $R^2$ |
| | | | | | Brazilian air-traffic network | | | | | | | |
| Degree | 0.016 | 0.990 | 0.047 | 0.962 | - | - | 0.047 | 0.962 | 0.071 | 0.571 | 0.076 | 0.946 |
| node2vec | 0.183 | 0.111 | 0.229 | 0.203 | 0.192 | 0.137 | 0.229 | 0.203 | 0.141 | -0.043 | 0.280 | 0.324 |
| struc2vec | 0.029 | 0.968 | 0.031 | 0.972 | 0.029 | 0.975 | 0.036 | 0.972 | 0.066 | 0.304 | 0.064 | 0.959 |
| GraphWave | 0.020 | 0.978 | 0.035 | 0.958 | 0.021 | 0.975 | 0.034 | 0.956 | 0.082 | 0.383 | 0.089 | 0.920 |
| GraphSAGE | 0.163 | 0.253 | 0.209 | 0.333 | 0.171 | 0.275 | 0.209 | 0.333 | 0.136 | -0.070 | 0.258 | 0.425 |
| ARGA | 0.045 | 0.927 | 0.048 | 0.954 | 0.040 | 0.949 | 0.048 | 0.954 | 0.052 | 0.606 | 0.087 | 0.933 |
| DRNE | 0.016 | 0.991 | 0.037 | 0.974 | 0.007 | 0.998 | 0.037 | 0.973 | 0.071 | 0.569 | 0.061 | 0.966 |
| SIR-GN: GMM | 0.011 | 0.996 | 0.008 | 0.998 | 0.008 | 0.998 | 0.008 | 0.998 | 0.073 | 0.618 | 0.064 | 0.965 |
| SIR-GN: K-Means | 0.009 | 0.997 | 0.006 | **0.999** | **0.006** | **0.999** | 0.006 | **0.999** | 0.037 | 0.795 | **0.042** | **0.984** |
| SILA | **0.006** | **0.999** | **0.005** | **0.999** | 0.007 | **0.999** | **0.005** | **0.999** | **0.031** | **0.873** | 0.043 | **0.984** |
| SparseStruct | 0.009 | 0.997 | 0.008 | **0.999** | 0.009 | 0.998 | 0.008 | **0.999** | 0.049 | 0.766 | 0.086 | 0.933 |

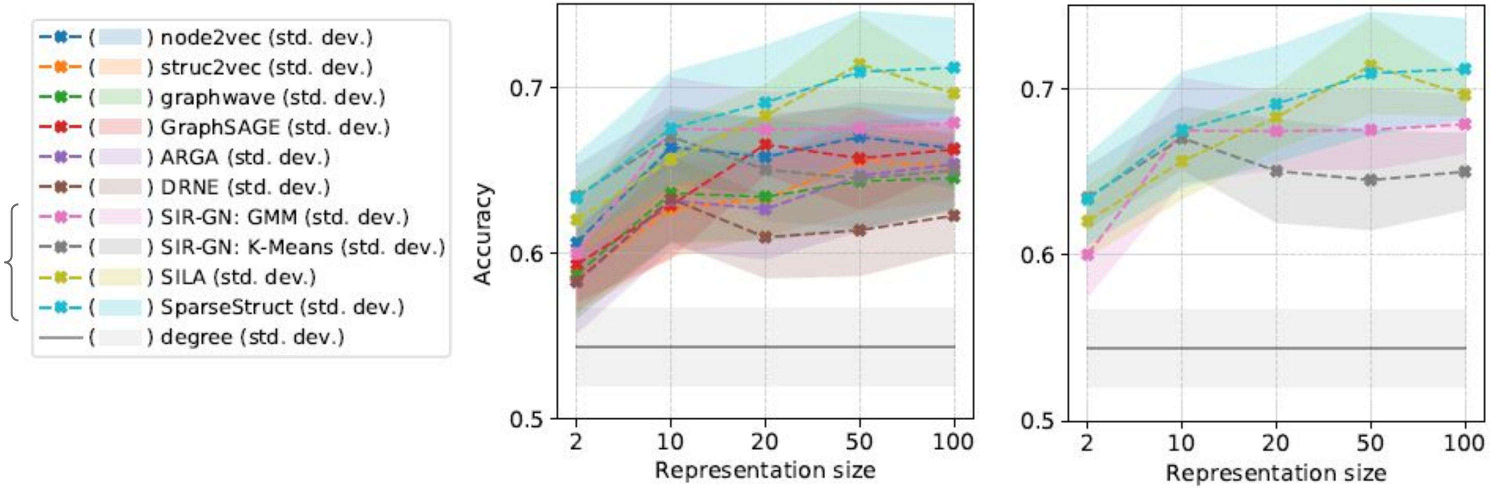
Proposed methods

Our propose methods obtain comparable or better results than the competitors on predicting centrality measures, this implies better encoding of structural properties.

# Results

- Node classification experiment:
  - Tested on 3 different graphs Brasil, Europe and American
  - American ->

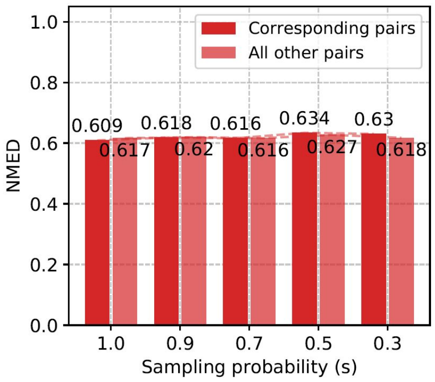
Proposed methods



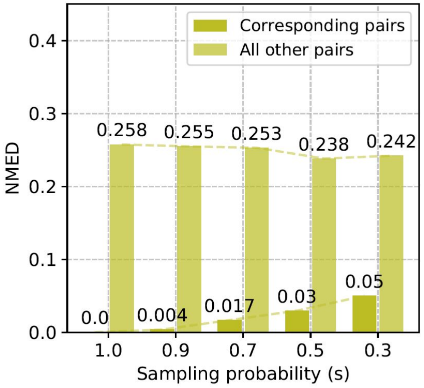
Our propose methods obtain comparable or higher accuracy on structural classification tasks. Results imply better encoding of structural properties.

# Results

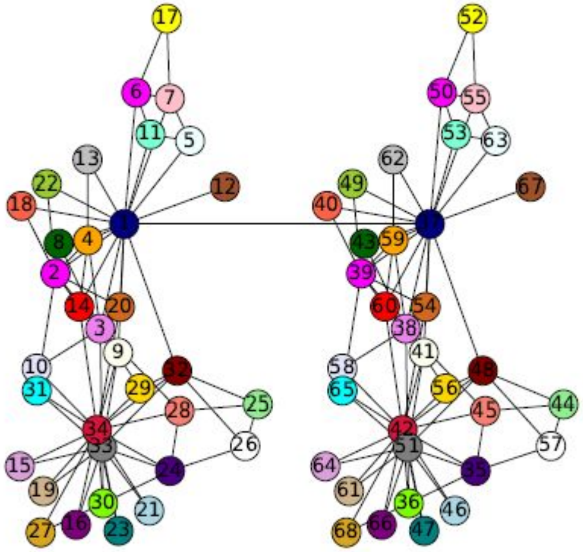
- Robustness to structural noise experiment:
  - Mirrored graph
    - Known structurally equivalent pairs
  - Noise = random edges
  - Measure mean distance between:
    - **All corresponding pairs** and **All other pairs**



GraphSAGE



SILA

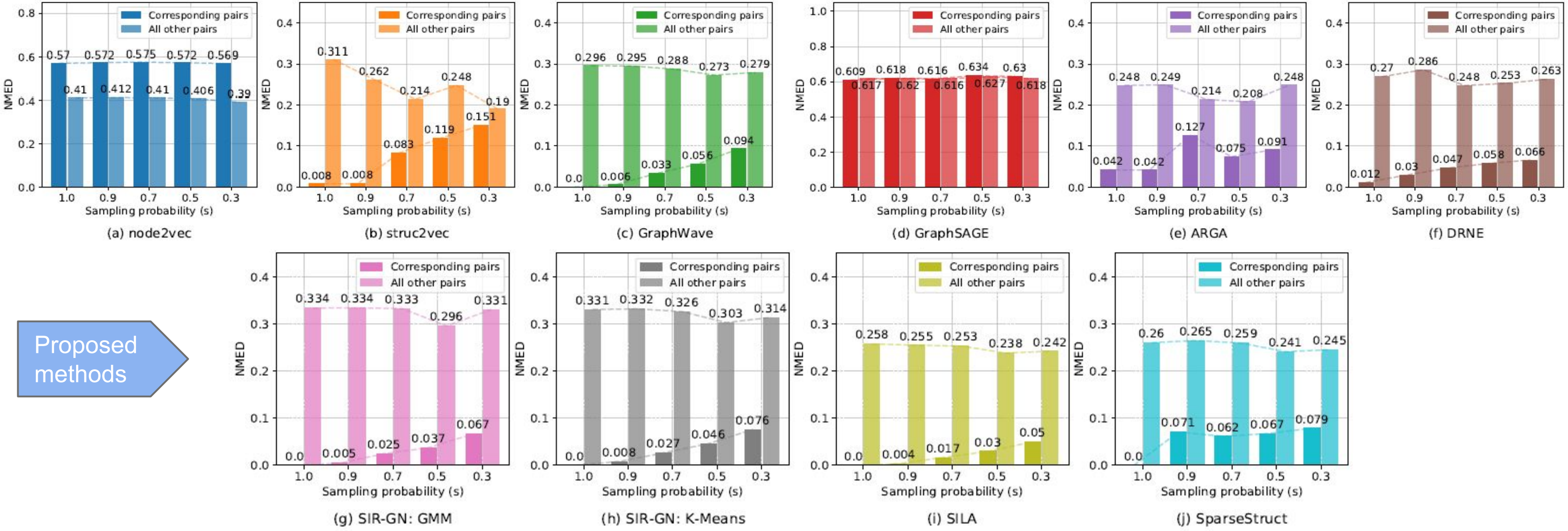


Mirrored network

NMED: Normalized Mean Euclidean Distance

# Results

Proposed methods



(a) node2vec (b) struc2vec (c) GraphWave (d) GraphSAGE (e) ARGA (f) DRNE

(g) SIR-GN: GMM (h) SIR-GN: K-Means (i) SILA (j) SparseStruct

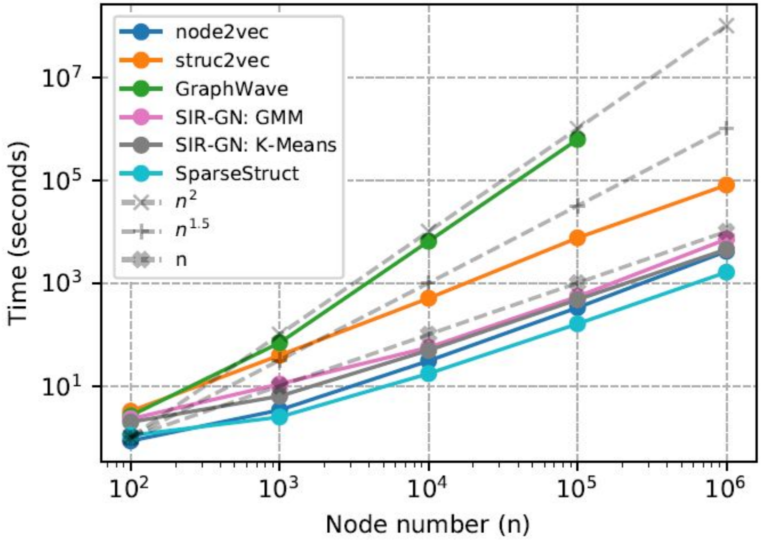
Bigger distance between corresponding pairs and all other pairs means lower sensibility to noise, our propose methods obtain comparable or better results than the competitors.

# Results

- Scalability experiment:
  - Generate synthetic graphs:
    - Erdos-Renyi random graphs
    - Sizes 100 to 1.000.000 nodes
  - Result comparison  (cpu only):
    - GraphWave: 3 years
    - Struc2vec: 28 hours
    - SIR-GN: K-Means: 1.25 hours
    - SparseStruct: 27 minutes

Proposed methods

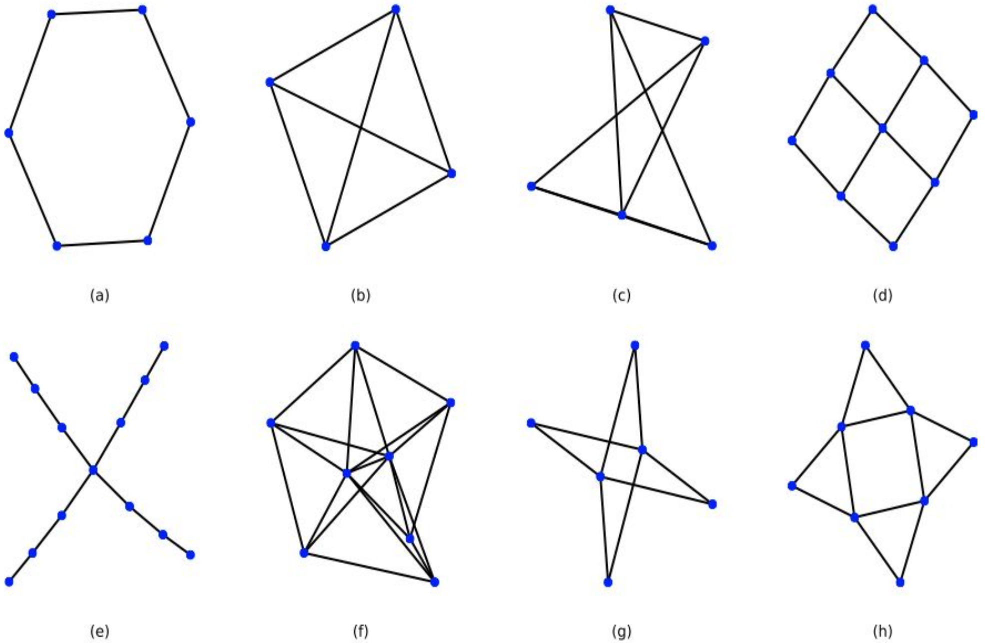
Our propose methods demonstrate to be considerably faster than the competitors



Node number (rep. size 20, log-log scale)

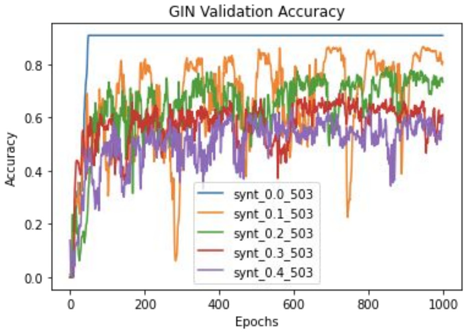
# Comparison With Supervised Procedures

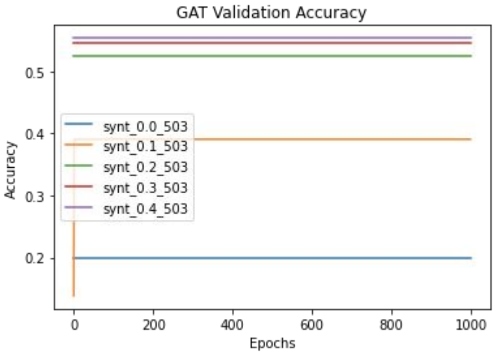
- 8 different prototype structures
- Several duplications of this structures
- Interconnected with a certain percentage of random edges w.r.t. the number of nodes
- Classification task each node should be classified with the prototype id that

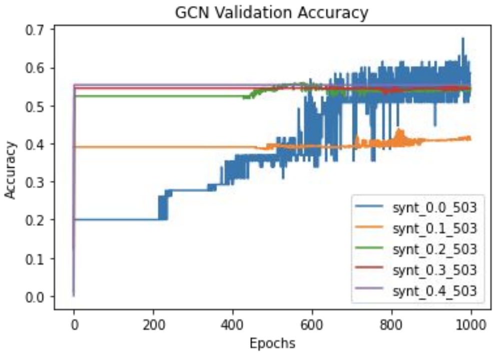

(a) (b) (c) (d)

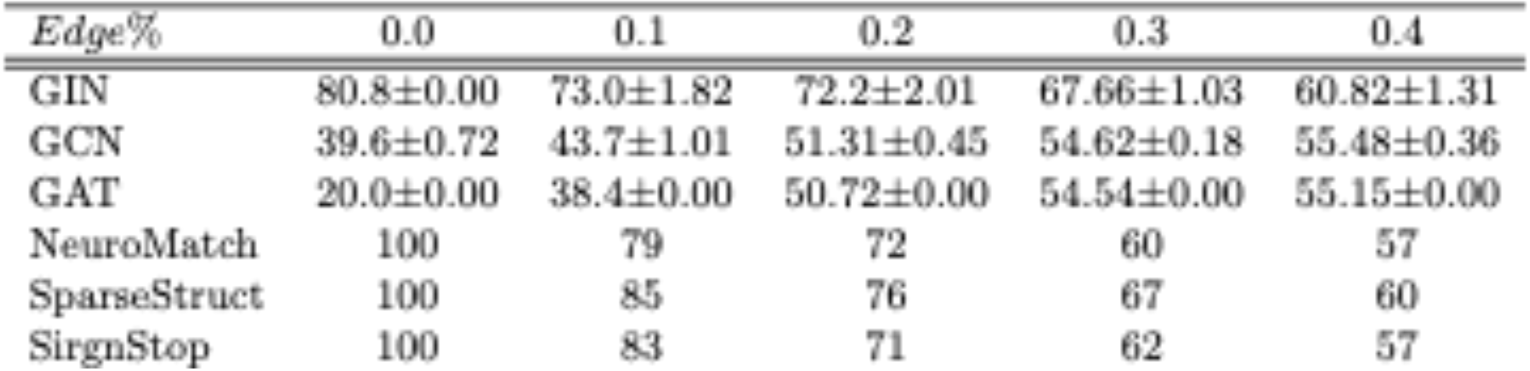
(e) (f) (g) (h)

# Comparison With Supervised Procedures


GIN Validation Accuracy


GAT Validation Accuracy


GCN Validation Accuracy

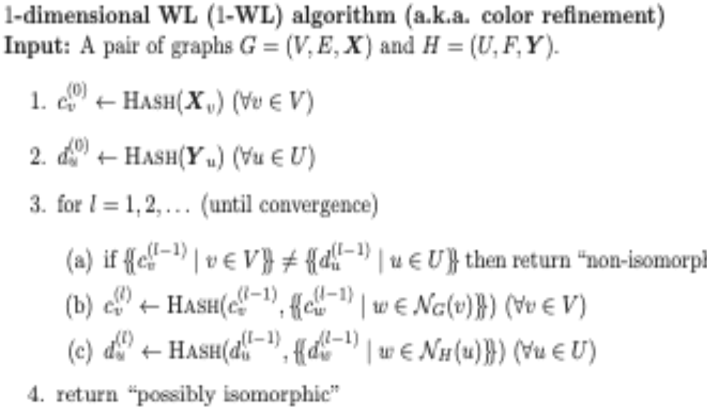


| *Edge%* | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 |
|---|---|---|---|---|---|
| GIN | 80.8±0.00 | 73.0±1.82 | 72.2±2.01 | 67.66±1.03 | 60.82±1.31 |
| GCN | 39.6±0.72 | 43.7±1.01 | 51.31±0.45 | 54.62±0.18 | 55.48±0.36 |
| GAT | 20.0±0.00 | 38.4±0.00 | 50.72±0.00 | 54.54±0.00 | 55.15±0.00 |
| NeuroMatch | 100 | 79 | 72 | 60 | 57 |
| SparseStruct | 100 | 85 | 76 | 67 | 60 |
| SirgnStop | 100 | 83 | 71 | 62 | 57 |

# Current Works

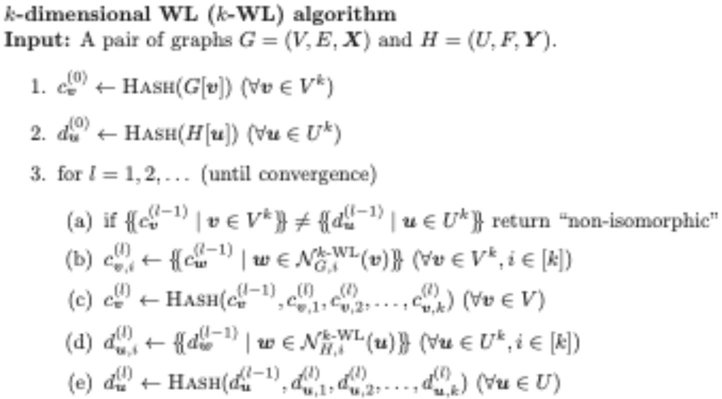
- Spatio-Temporal Models
- K-WL version
- Explainability
- HPC and extension of SIR-GN, SILA, and SparseStruct (3-dim WL) can works on large scale graphs and produce powerful structure representations.

# Weisfeiler-Lehman Isomorphism Test



**1-dimensional WL (1-WL) algorithm (a.k.a. color refinement)**
**Input:** A pair of graphs $G = (V, E, \boldsymbol{X})$ and $H = (U, F, \boldsymbol{Y})$.

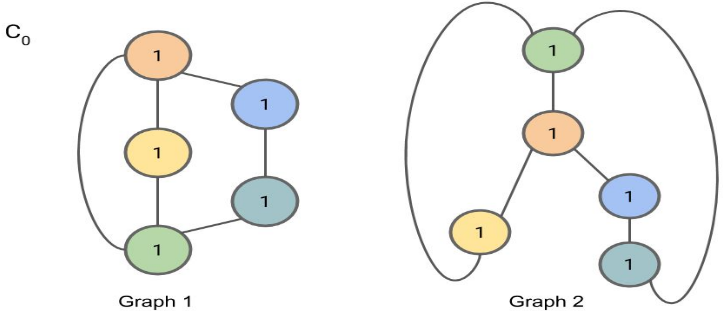
```
1. $c_v^{(0)} \leftarrow \textsc{Hash}(\boldsymbol{X}_v)\ (\forall v \in V)$
2. $d_u^{(0)} \leftarrow \textsc{Hash}(\boldsymbol{Y}_u)\ (\forall u \in U)$
3. for $l = 1, 2, \ldots$ (until convergence)
    (a) if $\{\!\{c_v^{(l-1)} \mid v \in V\}\!\} \neq \{\!\{d_u^{(l-1)} \mid u \in U\}\!\}$ then return "non-isomorpl
    (b) $c_v^{(l)} \leftarrow \textsc{Hash}(c_v^{(l-1)}, \{\!\{c_w^{(l-1)} \mid w \in \mathcal{N}_G(v)\}\!\})\ (\forall v \in V)$
    (c) $d_u^{(l)} \leftarrow \textsc{Hash}(d_u^{(l-1)}, \{\!\{d_w^{(l-1)} \mid w \in \mathcal{N}_H(u)\}\!\})\ (\forall u \in U)$
4. return "possibly isomorphic"
```



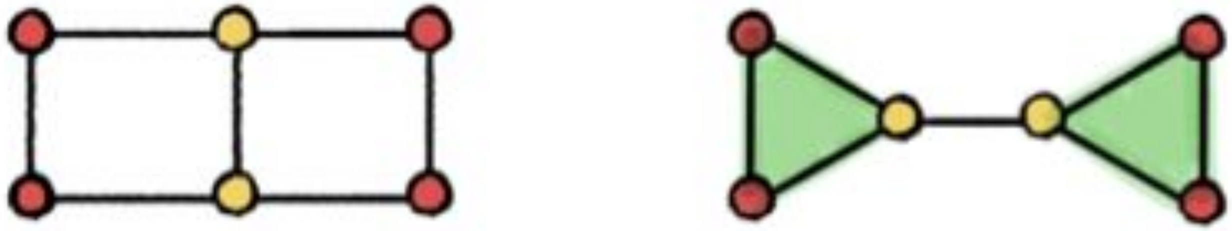
**$k$-dimensional WL ($k$-WL) algorithm**
**Input:** A pair of graphs $G = (V, E, \boldsymbol{X})$ and $H = (U, F, \boldsymbol{Y})$.

```
1. $c_{\boldsymbol{v}}^{(0)} \leftarrow \textsc{Hash}(G[\boldsymbol{v}])\ (\forall \boldsymbol{v} \in V^k)$
2. $d_{\boldsymbol{u}}^{(0)} \leftarrow \textsc{Hash}(H[\boldsymbol{u}])\ (\forall \boldsymbol{u} \in U^k)$
3. for $l = 1, 2, \ldots$ (until convergence)
    (a) if $\{\!\{c_{\boldsymbol{v}}^{(l-1)} \mid \boldsymbol{v} \in V^k\}\!\} \neq \{\!\{d_{\boldsymbol{u}}^{(l-1)} \mid \boldsymbol{u} \in U^k\}\!\}$ return "non-isomorphic"
    (b) $c_{\boldsymbol{v},i}^{(l)} \leftarrow \{\!\{c_{\boldsymbol{w}}^{(l-1)} \mid \boldsymbol{w} \in \mathcal{N}_{G,i}^{k\text{-WL}}(\boldsymbol{v})\}\!\}\ (\forall \boldsymbol{v} \in V^k, i \in [k])$
    (c) $c_{\boldsymbol{v}}^{(l)} \leftarrow \textsc{Hash}(c_{\boldsymbol{v}}^{(l-1)}, c_{\boldsymbol{v},1}^{(l)}, c_{\boldsymbol{v},2}^{(l)}, \ldots, c_{\boldsymbol{v},k}^{(l)})\ (\forall \boldsymbol{v} \in V)$
    (d) $d_{\boldsymbol{u},i}^{(l)} \leftarrow \{\!\{d_{\boldsymbol{w}}^{(l-1)} \mid \boldsymbol{w} \in \mathcal{N}_{H,i}^{k\text{-WL}}(\boldsymbol{u})\}\!\}\ (\forall \boldsymbol{u} \in U^k, i \in [k])$
    (e) $d_{\boldsymbol{u}}^{(l)} \leftarrow \textsc{Hash}(d_{\boldsymbol{u}}^{(l-1)}, d_{\boldsymbol{u},1}^{(l)}, d_{\boldsymbol{u},2}^{(l)}, \ldots, d_{\boldsymbol{u},k}^{(l)})\ (\forall \boldsymbol{u} \in U)$
```

Sato, R. (2020). A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*.



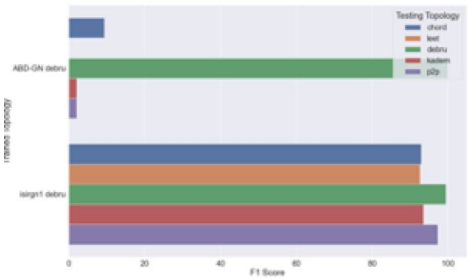
# Weisfeiler-Lehman Limitation

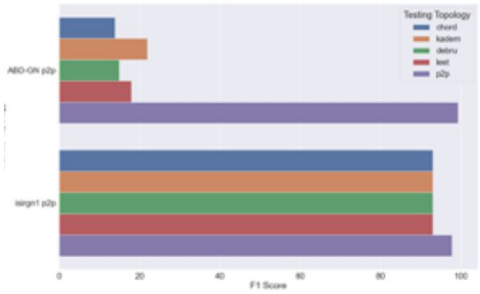


Examples of non-isomorphic graphs that cannot be distinguished by 1-WL but can be distinguished by 3-WL due to its capability of counting triangles.

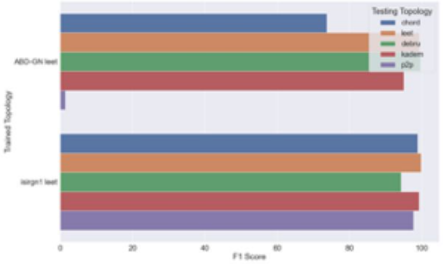
# Applications in National Security

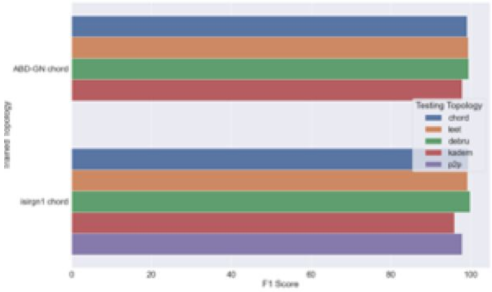
# Detecting Botnet Nodes via Structural Node Representation Learning



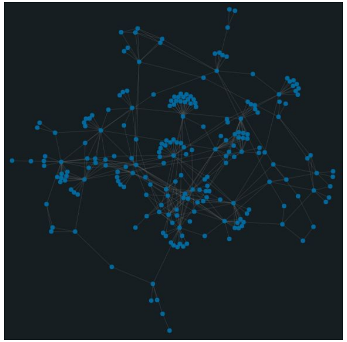


- Traffic IP networks (IP 1 communicate with IP2)
- Peer to peer DDoS attack collected in 2011
- Several Artificial Context
- Structural Task -> proximity problem

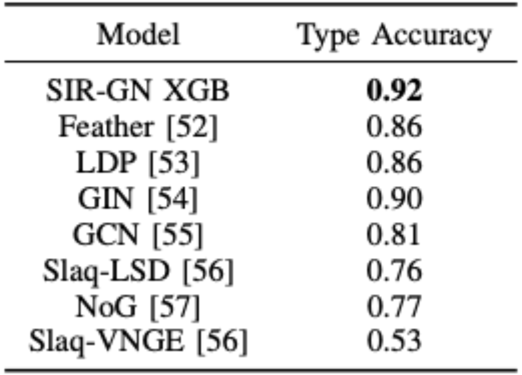




# Malware Classification, Evolution, and Explanation

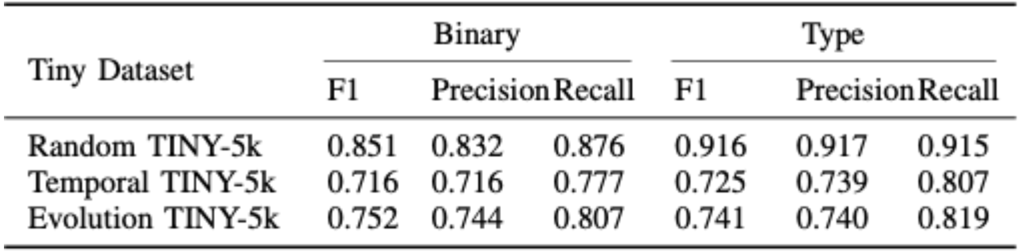


MalNet



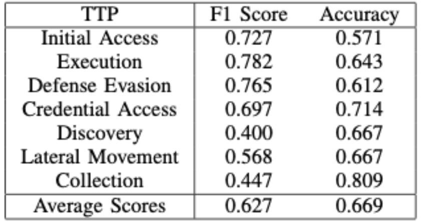
| Model | Type Accuracy |
|---|---|
| SIR-GN XGB | **0.92** |
| Feather [52] | 0.86 |
| LDP [53] | 0.86 |
| GIN [54] | 0.90 |
| GCN [55] | 0.81 |
| Slaq-LSD [56] | 0.76 |
| NoG [57] | 0.77 |
| Slaq-VNGE [56] | 0.53 |

Evolution



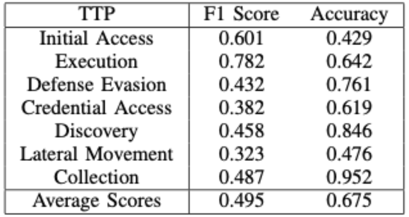
| Tiny Dataset | Binary | | | Type | | |
|---|---|---|---|---|---|---|
| | F1 | Precision | Recall | F1 | Precision | Recall |
| Random TINY-5k | 0.851 | 0.832 | 0.876 | 0.916 | 0.917 | 0.915 |
| Temporal TINY-5k | 0.716 | 0.716 | 0.777 | 0.725 | 0.739 | 0.807 |
| Evolution TINY-5k | 0.752 | 0.744 | 0.807 | 0.741 | 0.740 | 0.819 |

Graph Isomorphic Network (GIN) classification results for each TTP.



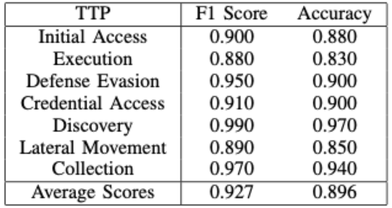
| TTP | F1 Score | Accuracy |
|---|---|---|
| Initial Access | 0.727 | 0.571 |
| Execution | 0.782 | 0.643 |
| Defense Evasion | 0.765 | 0.612 |
| Credential Access | 0.697 | 0.714 |
| Discovery | 0.400 | 0.667 |
| Lateral Movement | 0.568 | 0.667 |
| Collection | 0.447 | 0.809 |
| Average Scores | 0.627 | 0.669 |

GAT classification results for each TTP.



| TTP | F1 Score | Accuracy |
|---|---|---|
| Initial Access | 0.601 | 0.429 |
| Execution | 0.782 | 0.642 |
| Defense Evasion | 0.432 | 0.761 |
| Credential Access | 0.382 | 0.619 |
| Discovery | 0.458 | 0.846 |
| Lateral Movement | 0.323 | 0.476 |
| Collection | 0.487 | 0.952 |
| Average Scores | 0.495 | 0.675 |

Inferential SIR-GN + Random Forest TTPs classification results.

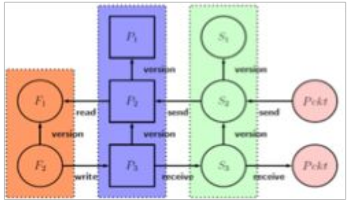


| TTP | F1 Score | Accuracy |
|---|---|---|
| Initial Access | 0.900 | 0.880 |
| Execution | 0.880 | 0.830 |
| Defense Evasion | 0.950 | 0.900 |
| Credential Access | 0.910 | 0.900 |
| Discovery | 0.990 | 0.970 |
| Lateral Movement | 0.890 | 0.850 |
| Collection | 0.970 | 0.940 |
| Average Scores | 0.927 | 0.896 |

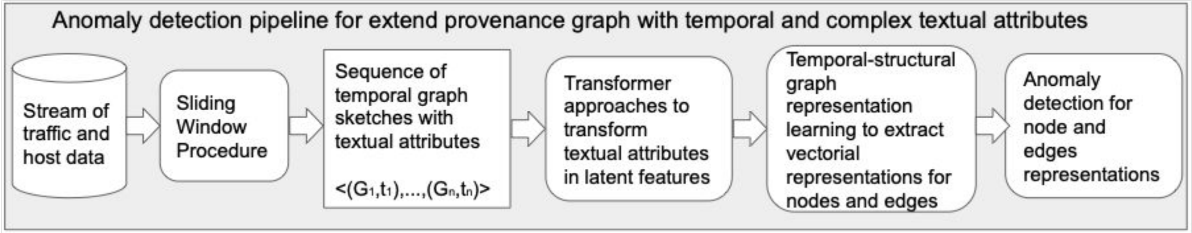
ATT&CK®

Explanation

# Meta Data Traffic Anomaly Detection via Interpretable Temporal Structural Provenance Graph Representation Learning



(a) CamFlow [8] Provenance subgraph. Nodes types are Processes, Files and Sockets with subscripts representing version. Edges represent actions between nodes.



Anomaly detection pipeline for extend provenance graph with temporal and complex textual attributes

Stream of traffic and host data

Sliding Window Procedure

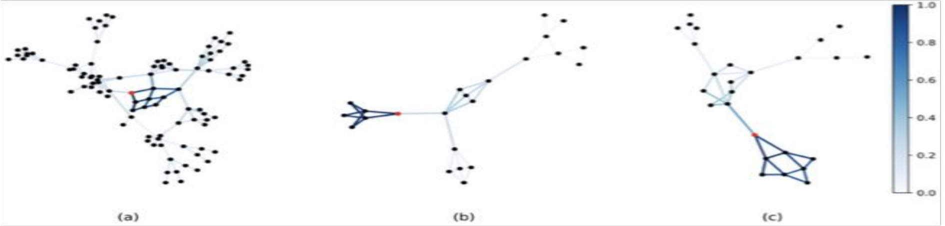
Sequence of temporal graph sketches with textual attributes

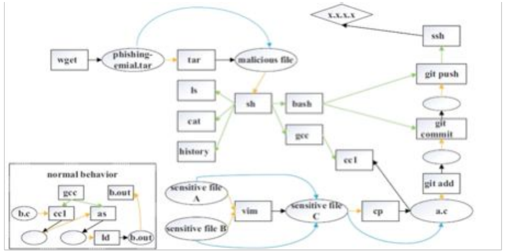
$<(G_1,t_1),\ldots,(G_n,t_n)>$

Transformer approaches to transform textual attributes in latent features

Temporal-structural graph representation learning to extract vectorial representations for nodes and edges

Anomaly detection for node and edges representations





(b) Provenance graph for a theoretical phishing attack proposed by the authors of Paradise [9]. Normal behavior is shown in the dotted box. Edge types are represented with different colors.

This research is funded by a National Centers of Academic Excellence in Cybersecurity grant (H98230-22-1-0300), which is part of the US National Security Agency

# Conclusion

# Conclusion

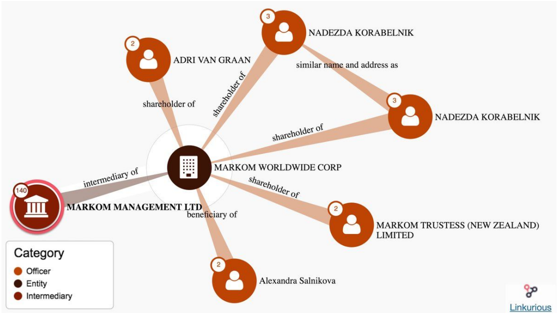
- Structural VS Connectivity
- Problems of Connectivity
- Approach for Unsupervised Structural Representation Learning
  - Sir-GN
  - Sparsestruct
  - SILA
- Applications in National Security
  - Offshore Leaks Networks
  - Malware
  - Botnets
  - Meta Data Traffic Anomaly Detection (NCAE-grant)
- Future Directions
  - Improve expressive power of Graph Representation Learning Procedure (K-WL)
  - Improve performance and creation of parallel/distributed version
  - Spatio Temporal Extensions
  - Graph generation
  - Explanation

# References

1. Joaristi, M., & Serra, E. (2022). Structural iterative lexicographic autoencoded node representation. Data Mining and Knowledge Discovery, 1-29.
2. Joaristi, M., & Serra, E. (2021). Sir-gn: A fast structural iterative representation learning approach for graph nodes. ACM Transactions on Knowledge Discovery from Data (TKDD), 15(6), 1-39.
3. Serra, E., Joaristi, M., & Cuzzocrea, A. (2020, December). Large-scale sparse structural node representation. In 2020 IEEE International Conference on Big Data (Big Data) (pp. 5247-5253). IEEE.
4. Layne, J., & Serra, E. (2021). Inferential sir-gn: Scalable graph representation learning. arXiv preprint arXiv:2111.04826.
5. Campedelli, G. M., Layne, J., Herzoff, J., & Serra, E. (2022). The geometrical shapes of violence: predicting and explaining terrorist operations through graph embeddings. Journal of Complex Networks, 10(2), cnac008.
6. Carpenter, J., Layne, J., Serra, E., & Cuzzocrea, A. (2021, December). Detecting Botnet Nodes via Structural Node Representation Learning. In 2021 IEEE International Conference on Big Data (Big Data) (pp. 5357-5364). IEEE.
7. Quebrado, M., Serra, E., & Cuzzocrea, A. (2021, December). Android Malware Identification and Polymorphic Evolution Via Graph Representation Learning. In 2021 IEEE International Conference on Big Data (Big Data) (pp. 5441-5449). IEEE.
8. Daley, B., Serra, E., & Cuzzocrea, A. (2021, December). Identifying Malicious Users in the Offshore Leaks Networks via Structural Node Representation Learning. In 2021 IEEE International Conference on Big Data (Big Data) (pp. 5095-5101). IEEE.
9. Fairbanks, J., Orbe, A., Patterson, C., Layne, J., Serra, E., & Scheepers, M. (2021, December). Identifying ATT&CK Tactics in Android Malware Control Flow Graph Through Graph Representation Learning and Interpretability. In 2021 IEEE International Conference on Big Data (Big Data) (pp. 5602-5608). IEEE.

# Thanks
# Any questions?

# Identifying Malicious Users in the Offshore Leaks Networks via Structural Node Representation Learning



- Blacklists SDN entities
- Novelty detection task to identify r malicious user
- Structural Task

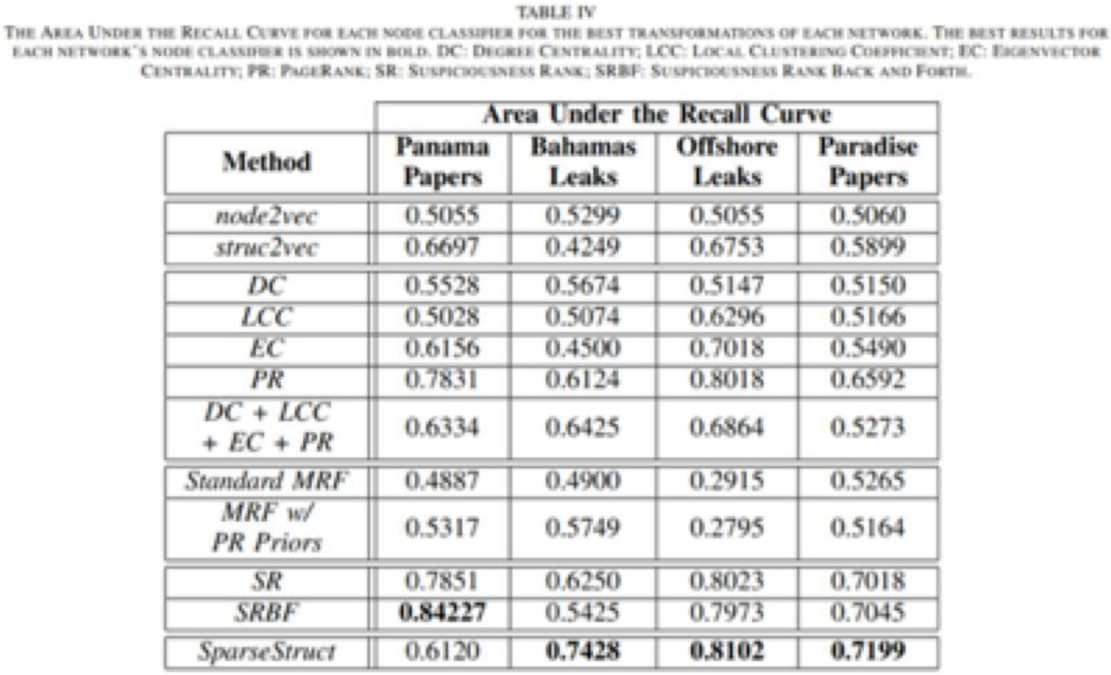
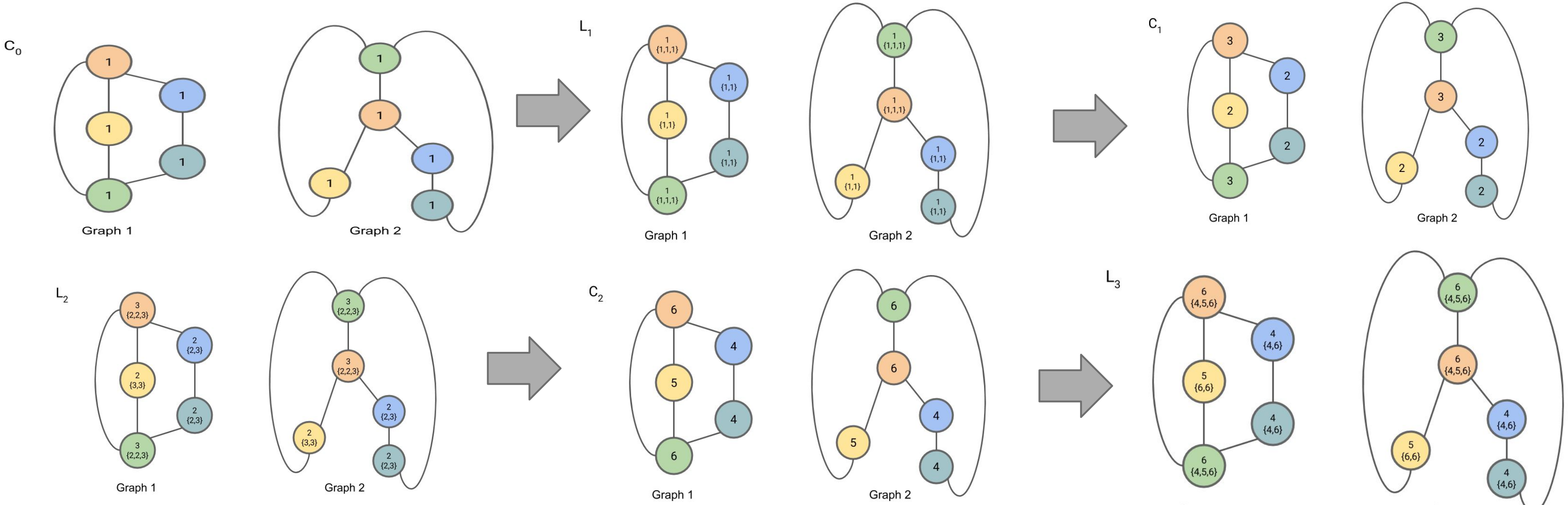


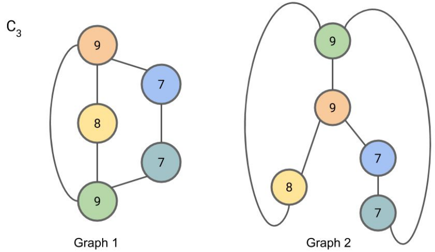
TABLE IV

The Area Under the Recall Curve for each node classifier for the best transformations of each network. The best results for each network's node classifier is shown in bold. DC: Degree Centrality; LCC: Local Clustering Coefficient; EC: Eigenvector Centrality; PR: PageRank; SR: Suspiciousness Rank; SRBF: Suspiciousness Rank Back and Forth.

| | Area Under the Recall Curve | | | |
|---|---|---|---|---|
| Method | Panama Papers | Bahamas Leaks | Offshore Leaks | Paradise Papers |
| *node2vec* | 0.5055 | 0.5299 | 0.5055 | 0.5060 |
| *struc2vec* | 0.6697 | 0.4249 | 0.6753 | 0.5899 |
| *DC* | 0.5528 | 0.5674 | 0.5147 | 0.5150 |
| *LCC* | 0.5028 | 0.5074 | 0.6296 | 0.5166 |
| *EC* | 0.6156 | 0.4500 | 0.7018 | 0.5490 |
| *PR* | 0.7831 | 0.6124 | 0.8018 | 0.6592 |
| *DC + LCC + EC + PR* | 0.6334 | 0.6425 | 0.6864 | 0.5273 |
| *Standard MRF* | 0.4887 | 0.4900 | 0.2915 | 0.5265 |
| *MRF w/ PR Priors* | 0.5317 | 0.5749 | 0.2795 | 0.5164 |
| *SR* | 0.7851 | 0.6250 | 0.8023 | 0.7018 |
| *SRBF* | **0.84227** | 0.5425 | 0.7973 | 0.7045 |
| *SparseStruct* | 0.6120 | **0.7428** | **0.8102** | **0.7199** |

# Weisfeiler-Lehman Isomorphism Test



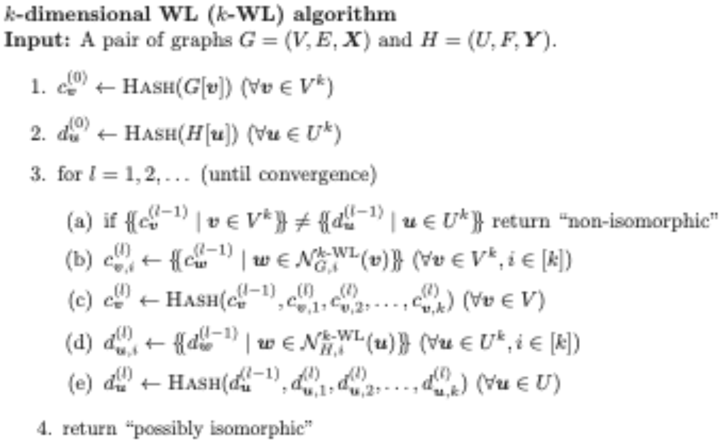


**1-dimensional WL (1-WL) algorithm (a.k.a. color refinement)**

**Input:** A pair of graphs $G = (V, E, \boldsymbol{X})$ and $H = (U, F, \boldsymbol{Y})$.

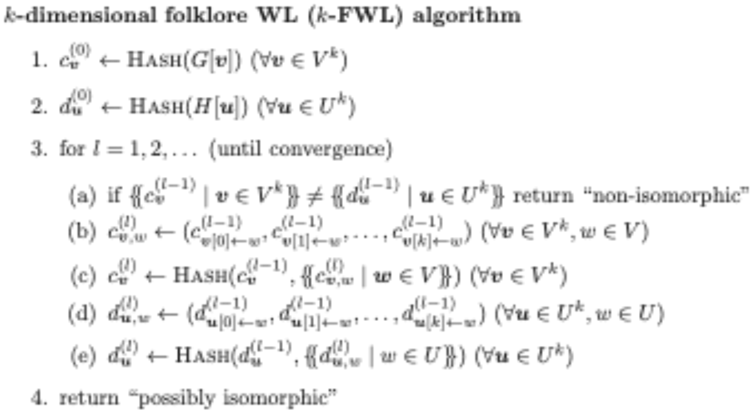
```
1. $c_v^{(0)} \leftarrow \textsc{Hash}(\boldsymbol{X}_v)$ $(\forall v \in V)$
2. $d_u^{(0)} \leftarrow \textsc{Hash}(\boldsymbol{Y}_u)$ $(\forall u \in U)$
3. for $l = 1, 2, \ldots$ (until convergence)
    (a) if $\{\!\{c_v^{(l-1)} \mid v \in V\}\!\} \neq \{\!\{d_u^{(l-1)} \mid u \in U\}\!\}$ then return "non-isomorphic"
    (b) $c_v^{(l)} \leftarrow \textsc{Hash}(c_v^{(l-1)}, \{\!\{c_w^{(l-1)} \mid w \in \mathcal{N}_G(v)\}\!\})$ $(\forall v \in V)$
    (c) $d_u^{(l)} \leftarrow \textsc{Hash}(d_u^{(l-1)}, \{\!\{d_w^{(l-1)} \mid w \in \mathcal{N}_H(u)\}\!\})$ $(\forall u \in U)$
4. return "possibly isomorphic"
```

# k-dim Weisfeiler-Lehman



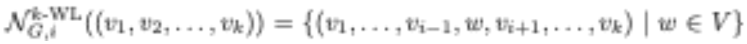
**$k$-dimensional WL ($k$-WL) algorithm**
**Input:** A pair of graphs $G = (V, E, \boldsymbol{X})$ and $H = (U, F, \boldsymbol{Y})$.

```
1. $c_{\boldsymbol{v}}^{(0)} \leftarrow \textsc{Hash}(G[\boldsymbol{v}])$ $(\forall \boldsymbol{v} \in V^k)$
2. $d_{\boldsymbol{u}}^{(0)} \leftarrow \textsc{Hash}(H[\boldsymbol{u}])$ $(\forall \boldsymbol{u} \in U^k)$
3. for $l = 1, 2, \ldots$ (until convergence)
    (a) if $\{\!\{c_{\boldsymbol{v}}^{(l-1)} \mid \boldsymbol{v} \in V^k\}\!\} \neq \{\!\{d_{\boldsymbol{u}}^{(l-1)} \mid \boldsymbol{u} \in U^k\}\!\}$ return "non-isomorphic"
    (b) $c_{\boldsymbol{v},i}^{(l)} \leftarrow \{\!\{c_{\boldsymbol{w}}^{(l-1)} \mid \boldsymbol{w} \in \mathcal{N}_{G,i}^{k\text{-WL}}(\boldsymbol{v})\}\!\}$ $(\forall \boldsymbol{v} \in V^k, i \in [k])$
    (c) $c_{\boldsymbol{v}}^{(l)} \leftarrow \textsc{Hash}(c_{\boldsymbol{v}}^{(l-1)}, c_{\boldsymbol{v},1}^{(l)}, c_{\boldsymbol{v},2}^{(l)}, \ldots, c_{\boldsymbol{v},k}^{(l)})$ $(\forall \boldsymbol{v} \in V)$
    (d) $d_{\boldsymbol{u},i}^{(l)} \leftarrow \{\!\{d_{\boldsymbol{w}}^{(l-1)} \mid \boldsymbol{w} \in \mathcal{N}_{H,i}^{k\text{-WL}}(\boldsymbol{u})\}\!\}$ $(\forall \boldsymbol{u} \in U^k, i \in [k])$
    (e) $d_{\boldsymbol{u}}^{(l)} \leftarrow \textsc{Hash}(d_{\boldsymbol{u}}^{(l-1)}, d_{\boldsymbol{u},1}^{(l)}, d_{\boldsymbol{u},2}^{(l)}, \ldots, d_{\boldsymbol{u},k}^{(l)})$ $(\forall \boldsymbol{u} \in U)$
4. return "possibly isomorphic"
```

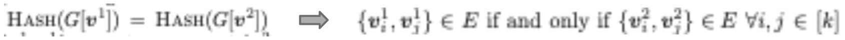


**$k$-dimensional folklore WL ($k$-FWL) algorithm**

```
1. $c_{\boldsymbol{v}}^{(0)} \leftarrow \textsc{Hash}(G[\boldsymbol{v}])$ $(\forall \boldsymbol{v} \in V^k)$
2. $d_{\boldsymbol{u}}^{(0)} \leftarrow \textsc{Hash}(H[\boldsymbol{u}])$ $(\forall \boldsymbol{u} \in U^k)$
3. for $l = 1, 2, \ldots$ (until convergence)
    (a) if $\{\!\{c_{\boldsymbol{v}}^{(l-1)} \mid \boldsymbol{v} \in V^k\}\!\} \neq \{\!\{d_{\boldsymbol{u}}^{(l-1)} \mid \boldsymbol{u} \in U^k\}\!\}$ return "non-isomorphic"
    (b) $c_{\boldsymbol{v},w}^{(l)} \leftarrow (c_{\boldsymbol{v}[0]\leftarrow w}^{(l-1)}, c_{\boldsymbol{v}[1]\leftarrow w}^{(l-1)}, \ldots, c_{\boldsymbol{v}[k]\leftarrow w}^{(l-1)})$ $(\forall \boldsymbol{v} \in V^k, w \in V)$
    (c) $c_{\boldsymbol{v}}^{(l)} \leftarrow \textsc{Hash}(c_{\boldsymbol{v}}^{(l-1)}, \{\!\{c_{\boldsymbol{v},w}^{(l)} \mid \boldsymbol{w} \in V\}\!\})$ $(\forall \boldsymbol{v} \in V^k)$
    (d) $d_{\boldsymbol{u},w}^{(l)} \leftarrow (d_{\boldsymbol{u}[0]\leftarrow w}^{(l-1)}, d_{\boldsymbol{u}[1]\leftarrow w}^{(l-1)}, \ldots, d_{\boldsymbol{u}[k]\leftarrow w}^{(l-1)})$ $(\forall \boldsymbol{u} \in U^k, w \in U)$
    (e) $d_{\boldsymbol{u}}^{(l)} \leftarrow \textsc{Hash}(d_{\boldsymbol{u}}^{(l-1)}, \{\!\{d_{\boldsymbol{u},w}^{(l)} \mid w \in U\}\!\})$ $(\forall \boldsymbol{u} \in U^k)$
4. return "possibly isomorphic"
```

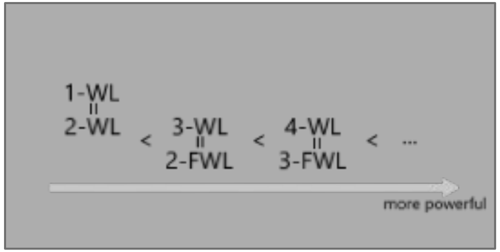


$$\mathcal{N}_{G,i}^{k\text{-WL}}((v_1, v_2, \ldots, v_k)) = \{(v_1, \ldots, v_{i-1}, w, v_{i+1}, \ldots, v_k) \mid w \in V\}$$



$$\textsc{Hash}(G[\boldsymbol{v}^1]) = \textsc{Hash}(G[\boldsymbol{v}^2]) \Rightarrow \{\boldsymbol{v}_i^1, \boldsymbol{v}_j^1\} \in E \text{ if and only if } \{\boldsymbol{v}_i^2, \boldsymbol{v}_j^2\} \in E \ \forall i, j \in [k]$$

# k-dim Weisfeiler-Lehman Costs

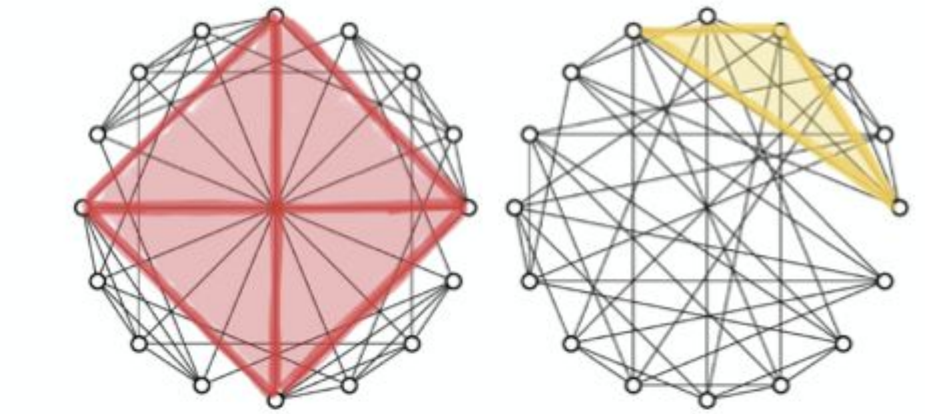


Each iteration in k-WL and k-FWL cost O(n^{k+1})

Even for the 2-FWL the cost of each iteration is O(n^3)

Not Practical

# Limitation of 2-FWL and 3-WL



Example of non-isomorphic strongly regular graphs with 16 vertices and node degree 6, where every two adjacent vertices have 2 mutual neighbours, and every two non-adjacent vertices also have 2 mutual neighbours. The 3-WL test fails on this example, while GSN with 4-clique structure can distinguish between them. In the graph on the left (known as the Rook's graph) each node participates in exactly one 4-clique. The graph on the right (Shrikhande graph) has maximum cliques of size 3 (triangles).

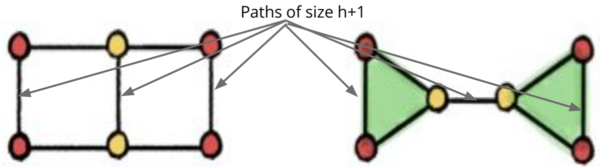
→ Bouritsas, G., Frasca, F., Zafeiriou, S., & Bronstein, M. M. (2020). Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*.

↓

Graphlet Feature in Graph neural Networks or iterative algorithms.

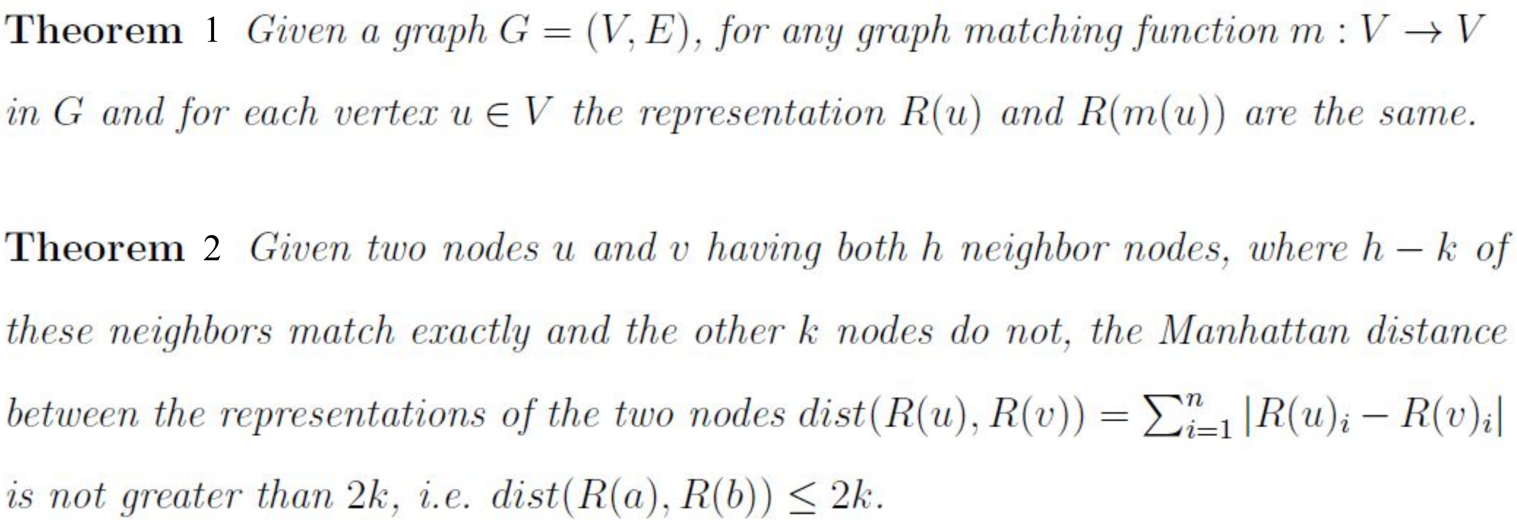
↓

Still limited to the order h of the graphlets



# Theoretical guarantees

- Theoretically proof that:

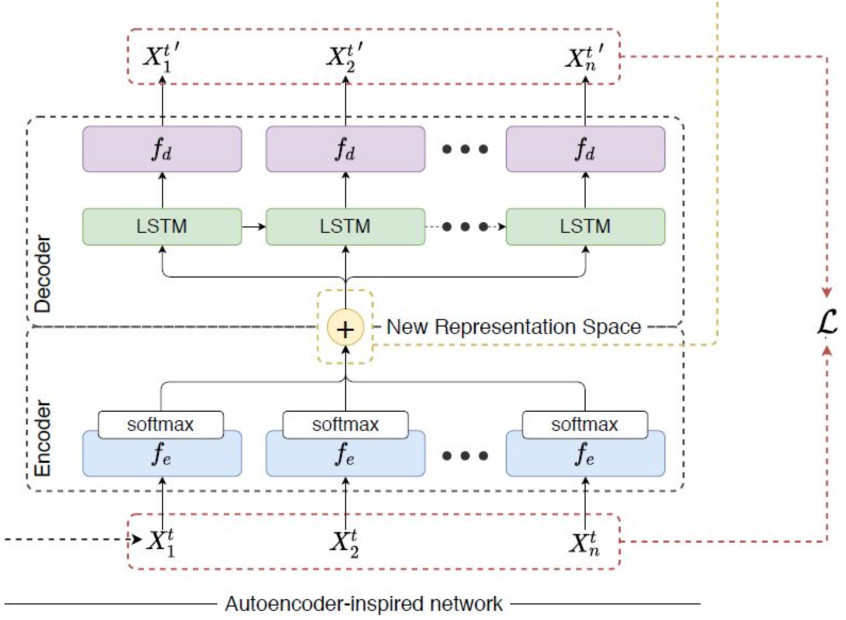


**Theorem 1** *Given a graph $G = (V, E)$, for any graph matching function $m : V \to V$ in $G$ and for each vertex $u \in V$ the representation $R(u)$ and $R(m(u))$ are the same.*

**Theorem 2** *Given two nodes $u$ and $v$ having both $h$ neighbor nodes, where $h - k$ of these neighbors match exactly and the other $k$ nodes do not, the Manhattan distance between the representations of the two nodes $dist(R(u), R(v)) = \sum_{i=1}^{n} |R(u)_i - R(v)_i|$ is not greater than $2k$, i.e. $dist(R(a), R(b)) \leq 2k$.*

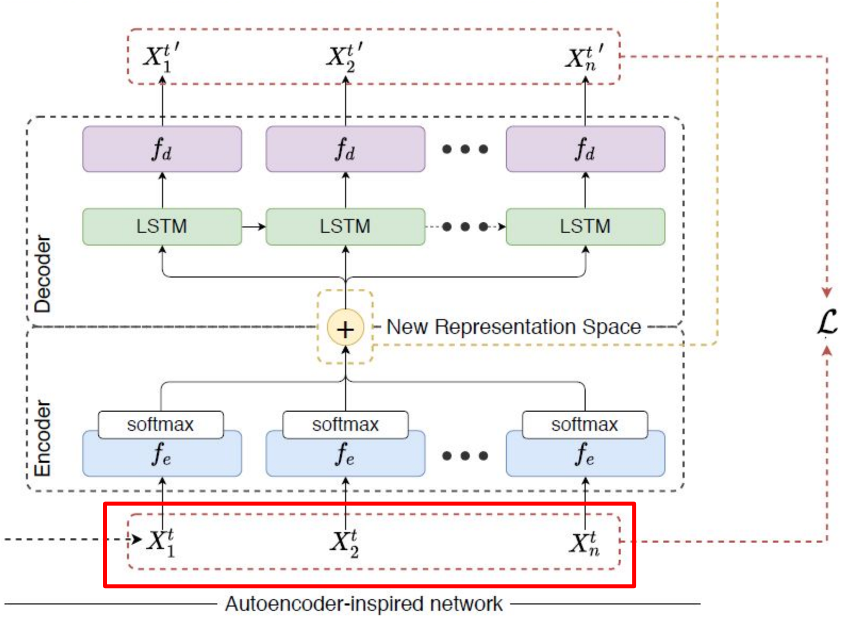
# SILA: Structural Iterative Lexicographic Autoencoded Node Representation

- Automatic vertex description:
  - Autoencoder neural network architecture
- Autoencoder architecture:
  - Input:
    - Representations of node neighborhoods
  - Transform to common space
  - Aggregation
  - Long Short-Term Memory
  - Minimize reconstruction error



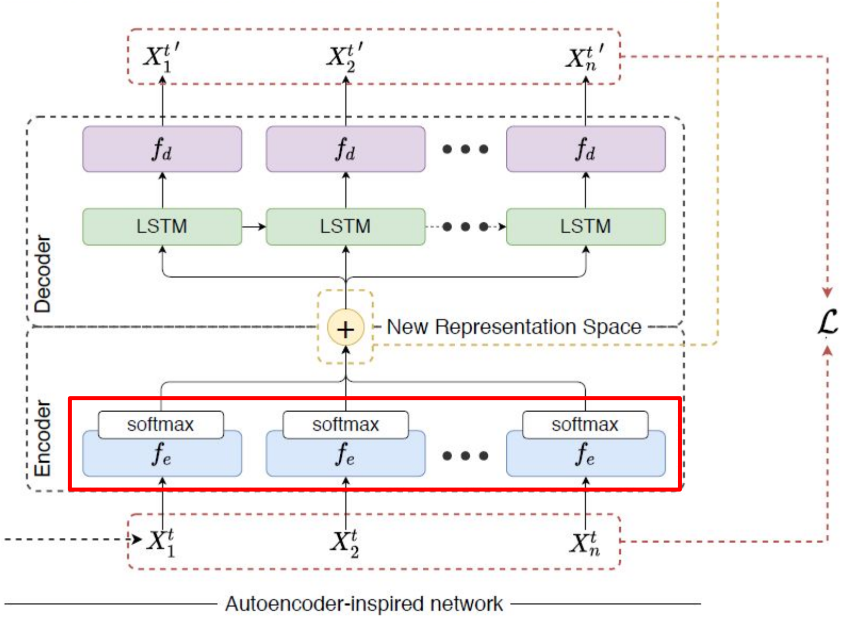
# SILA: Structural Iterative Lexicographic Autoencoded Node Representation

- Automatic vertex description:
  - Autoencoder neural network architecture
- Autoencoder architecture:
  - Input:
    - Representations of node neighborhoods
  - Transform to common space
  - Aggregation
  - Long Short-Term Memory
  - Minimize reconstruction error



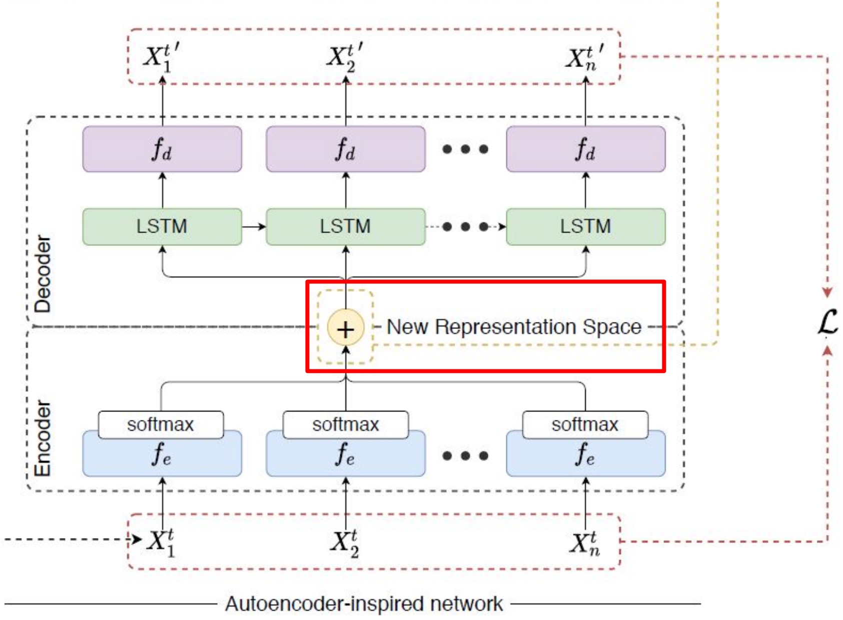
# SILA: Structural Iterative Lexicographic Autoencoded Node Representation

- Automatic vertex description:
  - Autoencoder neural network architecture
- Autoencoder architecture:
  - Input:
    - Representations of node neighborhoods
  - Transform to common space
  - Aggregation
  - Long Short-Term Memory
  - Minimize reconstruction error



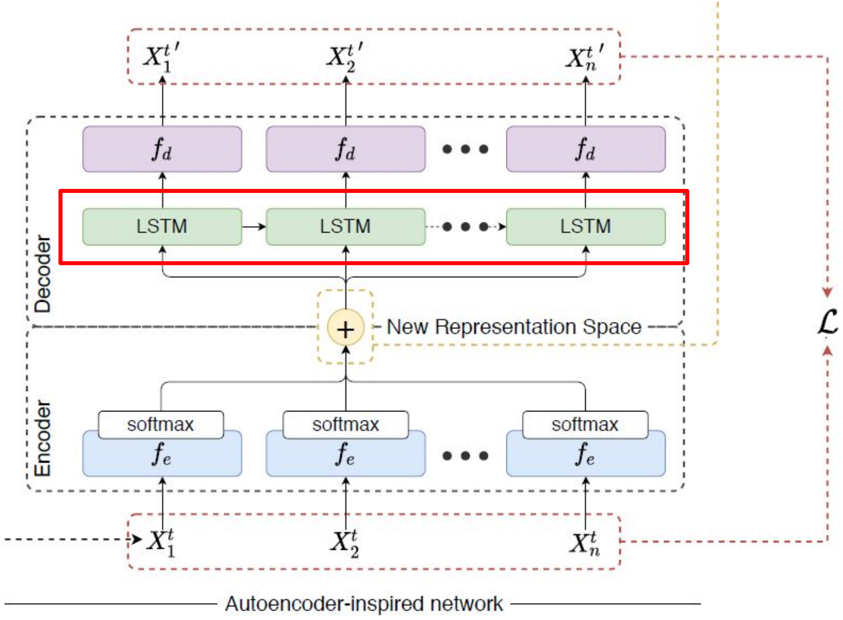
# SILA: Structural Iterative Lexicographic Autoencoded Node Representation

- Automatic vertex description:
  - Autoencoder neural network architecture
- Autoencoder architecture:
  - Input:
    - Representations of node neighborhoods
  - Transform to common space
  - Aggregation
  - Long Short-Term Memory
  - Minimize reconstruction error



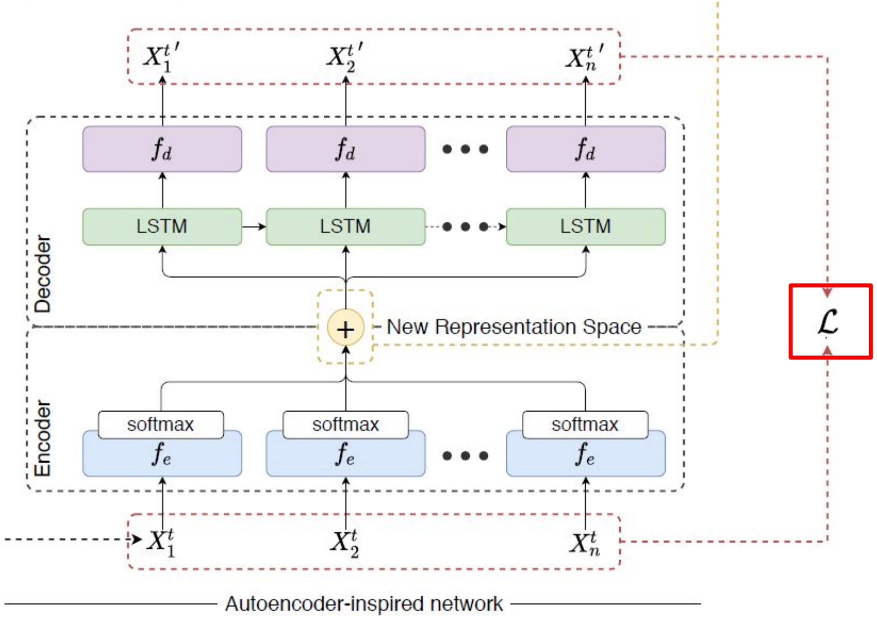
# SILA: Structural Iterative Lexicographic Autoencoded Node Representation

- Automatic vertex description:
  - Autoencoder neural network architecture
- Autoencoder architecture:
  - Input:
    - Representations of node neighborhoods
  - Transform to common space
  - Aggregation
  - Long Short-Term Memory
  - Minimize reconstruction error



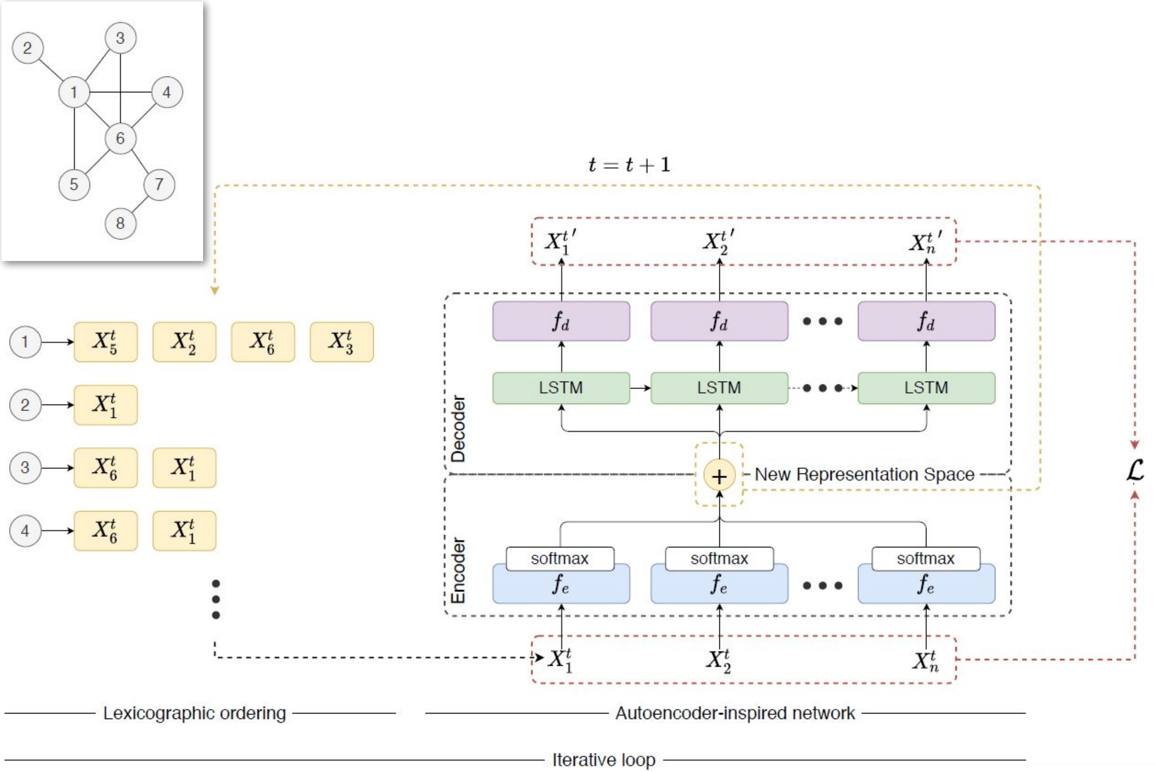
# SILA: Structural Iterative Lexicographic Autoencoded Node Representation

- Automatic vertex description:
  - Autoencoder neural network architecture
- Autoencoder architecture:
  - Input:
    - Representations of node neighborhoods
  - Transform to common space
  - Aggregation
  - Long Short-Term Memory
  - Minimize reconstruction error



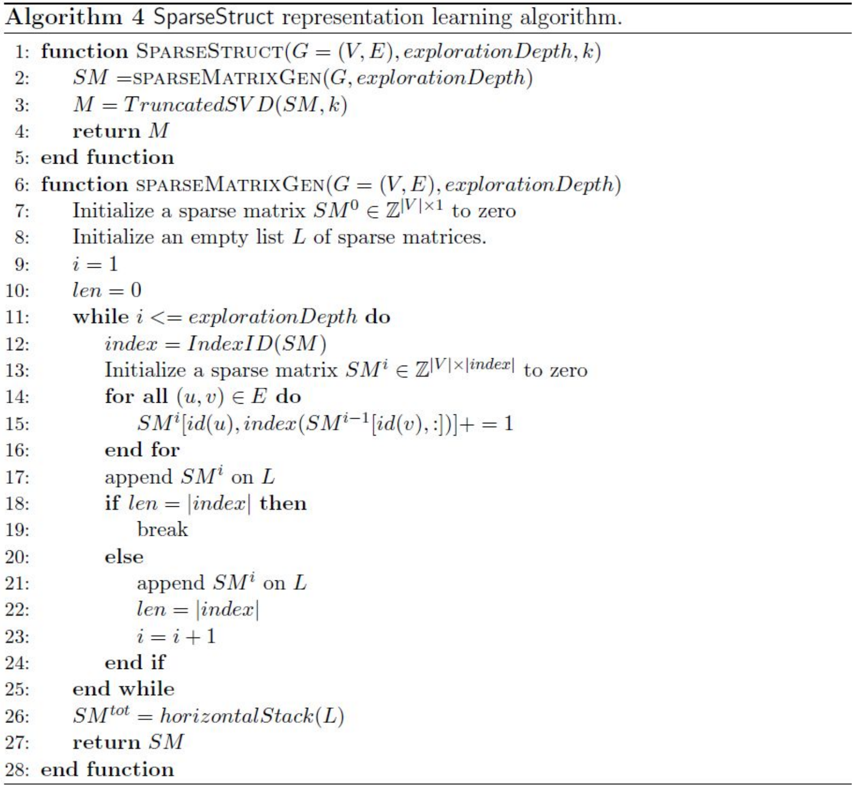
# SILA: Structural Iterative Lexicographic Autoencoded Node Representation

- Iterative process: 3 steps
  - Lexicographic ordering of current node representations
  - Train autoencoder
  - Extract new node representations
- Theoretically proof:
  - Same as SIR-GN



# SparseStruct: Sparse Structural Node Representation

- Inspired on Weisfeiler-Lehman Isomorphism Test
- Iterative steps:
  - Generate Index
  - Initialization
  - Sparse vertex description
  - Concatenation
- Truncated SVD
- Theoretically proof:
  - Same as SIR-GN
  - Proof of convergence

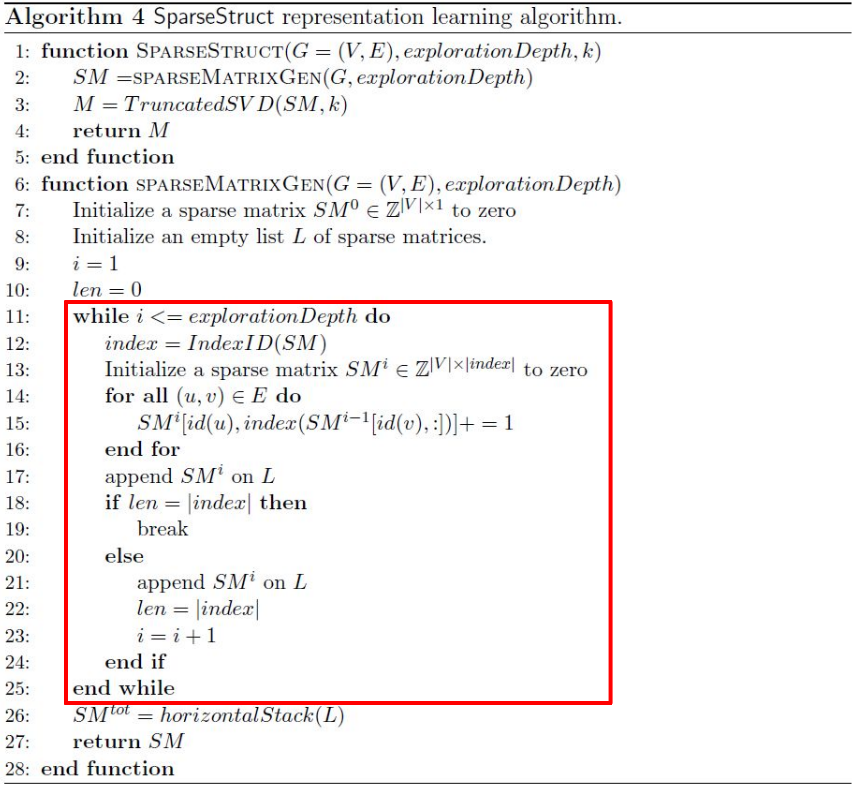


**Algorithm 4** SparseStruct representation learning algorithm.

```
function SPARSESTRUCT($G = (V, E), explorationDepth, k$)
    $SM$ =SPARSEMATRIXGEN($G, explorationDepth$)
    $M = TruncatedSVD(SM, k)$
    return $M$
end function
function SPARSEMATRIXGEN($G = (V, E), explorationDepth$)
    Initialize a sparse matrix $SM^0 \in \mathbb{Z}^{|V|\times 1}$ to zero
    Initialize an empty list $L$ of sparse matrices.
    $i = 1$
    $len = 0$
    while $i <= explorationDepth$ do
        $index = IndexID(SM)$
        Initialize a sparse matrix $SM^i \in \mathbb{Z}^{|V|\times|index|}$ to zero
        for all $(u, v) \in E$ do
            $SM^i[id(u), index(SM^{i-1}[id(v), :])] += 1$
        end for
        append $SM^i$ on $L$
        if $len = |index|$ then
            break
        else
            append $SM^i$ on $L$
            $len = |index|$
            $i = i + 1$
        end if
    end while
    $SM^{tot} = horizontalStack(L)$
    return $SM$
end function
```

# SparseStruct: Sparse Structural Node Representation

- Inspired on Weisfeiler-Lehman Isomorphism Test
- Iterative steps:
  - Generate Index
  - Initialization
  - Sparse vertex description
  - Concatenation
- Truncated SVD
- Theoretically proof:
  - Same as SIR-GN
  - Proof of convergence

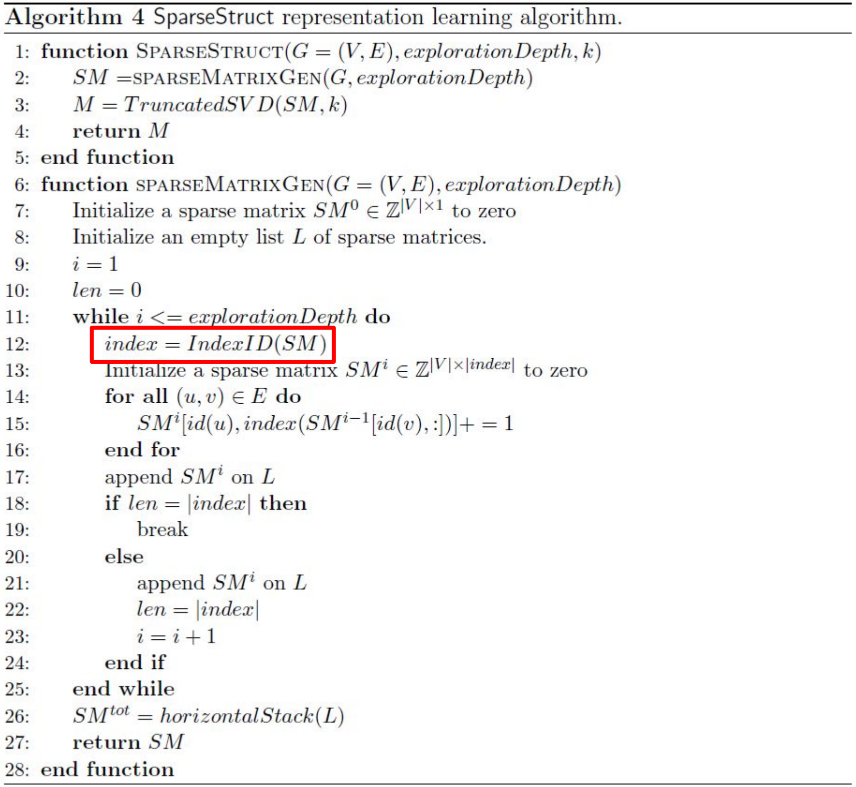


**Algorithm 4** SparseStruct representation learning algorithm.

```
function SPARSESTRUCT($G = (V, E), explorationDepth, k$)
    $SM$ = SPARSEMATRIXGEN($G, explorationDepth$)
    $M = TruncatedSVD(SM, k)$
    return $M$
end function
function SPARSEMATRIXGEN($G = (V, E), explorationDepth$)
    Initialize a sparse matrix $SM^0 \in \mathbb{Z}^{|V| \times 1}$ to zero
    Initialize an empty list $L$ of sparse matrices.
    $i = 1$
    $len = 0$
    while $i <= explorationDepth$ do
        $index = IndexID(SM)$
        Initialize a sparse matrix $SM^i \in \mathbb{Z}^{|V| \times |index|}$ to zero
        for all $(u, v) \in E$ do
            $SM^i[id(u), index(SM^{i-1}[id(v), :])] += 1$
        end for
        append $SM^i$ on $L$
        if $len = |index|$ then
            break
        else
            append $SM^i$ on $L$
            $len = |index|$
            $i = i + 1$
        end if
    end while
    $SM^{tot} = horizontalStack(L)$
    return $SM$
end function
```

# SparseStruct: Sparse Structural Node Representation

- Inspired on Weisfeiler-Lehman Isomorphism Test
- Iterative steps:
  - Generate Index
  - Initialization
  - Sparse vertex description
  - Concatenation
- Truncated SVD
- Theoretically proof:
  - Same as SIR-GN
  - Proof of convergence

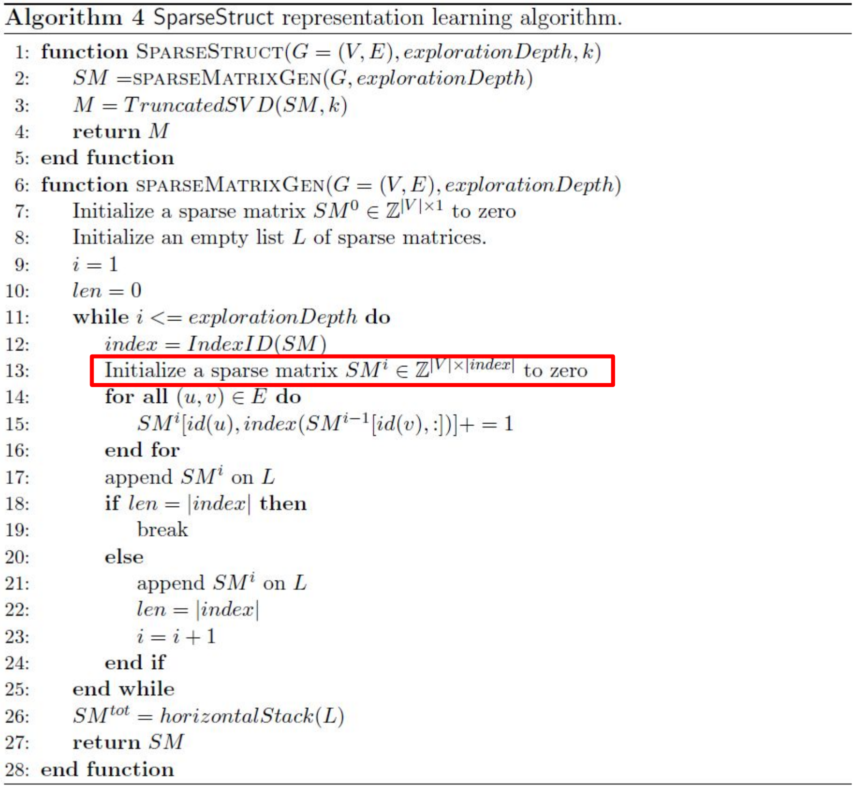


**Algorithm 4** SparseStruct representation learning algorithm.

```
function SparseStruct($G = (V, E), explorationDepth, k$)
    $SM$ =sparseMatrixGen($G, explorationDepth$)
    $M = TruncatedSVD(SM, k)$
    return $M$
end function
function sparseMatrixGen($G = (V, E), explorationDepth$)
    Initialize a sparse matrix $SM^0 \in \mathbb{Z}^{|V| \times 1}$ to zero
    Initialize an empty list $L$ of sparse matrices.
    $i = 1$
    $len = 0$
    while $i <= explorationDepth$ do
        $index = IndexID(SM)$
        Initialize a sparse matrix $SM^i \in \mathbb{Z}^{|V| \times |index|}$ to zero
        for all $(u, v) \in E$ do
            $SM^i[id(u), index(SM^{i-1}[id(v), :])] += 1$
        end for
        append $SM^i$ on $L$
        if $len = |index|$ then
            break
        else
            append $SM^i$ on $L$
            $len = |index|$
            $i = i + 1$
        end if
    end while
    $SM^{tot} = horizontalStack(L)$
    return $SM$
end function
```

# SparseStruct: Sparse Structural Node Representation

- Inspired on Weisfeiler-Lehman Isomorphism Test
- Iterative steps:
  - Generate Index
  - Initialization
  - Sparse vertex description
  - Concatenation
- Truncated SVD
- Theoretically proof:
  - Same as SIR-GN
  - Proof of convergence

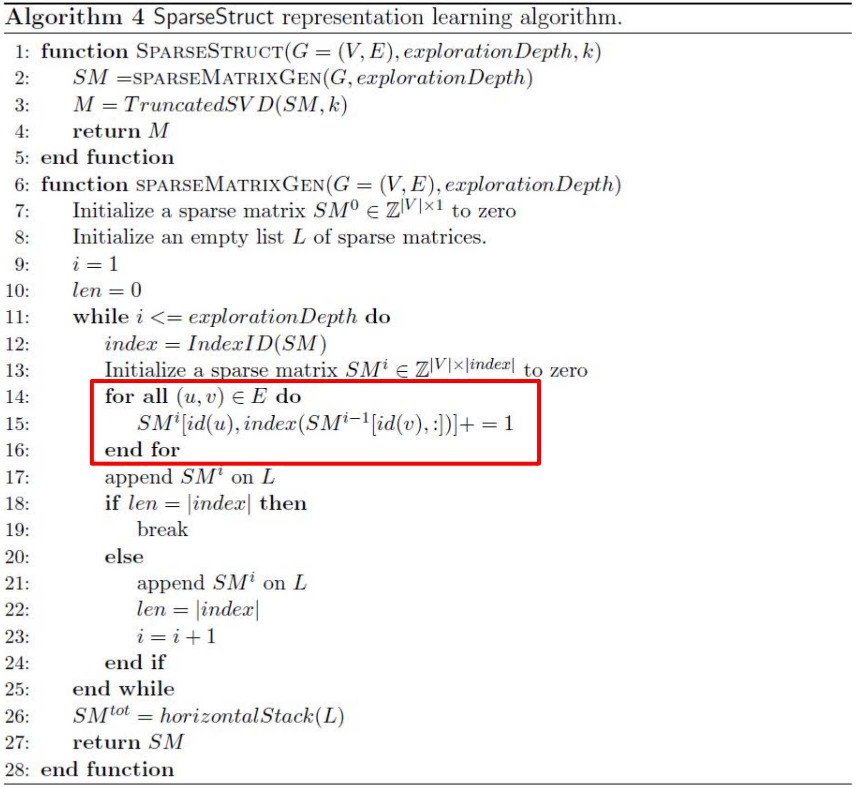


**Algorithm 4** SparseStruct representation learning algorithm.

```
function SPARSESTRUCT($G = (V, E), explorationDepth, k$)
    $SM$ = SPARSEMATRIXGEN($G, explorationDepth$)
    $M = TruncatedSVD(SM, k)$
    return $M$
end function
function SPARSEMATRIXGEN($G = (V, E), explorationDepth$)
    Initialize a sparse matrix $SM^0 \in \mathbb{Z}^{|V| \times 1}$ to zero
    Initialize an empty list $L$ of sparse matrices.
    $i = 1$
    $len = 0$
    while $i <= explorationDepth$ do
        $index = IndexID(SM)$
        Initialize a sparse matrix $SM^i \in \mathbb{Z}^{|V| \times |index|}$ to zero
        for all $(u, v) \in E$ do
            $SM^i[id(u), index(SM^{i-1}[id(v), :])] += 1$
        end for
        append $SM^i$ on $L$
        if $len = |index|$ then
            break
        else
            append $SM^i$ on $L$
            $len = |index|$
            $i = i + 1$
        end if
    end while
    $SM^{tot} = horizontalStack(L)$
    return $SM$
end function
```

# SparseStruct: Sparse Structural Node Representation

- Inspired on Weisfeiler-Lehman Isomorphism Test
- Iterative steps:
  - Generate Index
  - Initialization
  - Sparse vertex description
  - Concatenation
- Truncated SVD
- Theoretically proof:
  - Same as SIR-GN
  - Proof of convergence

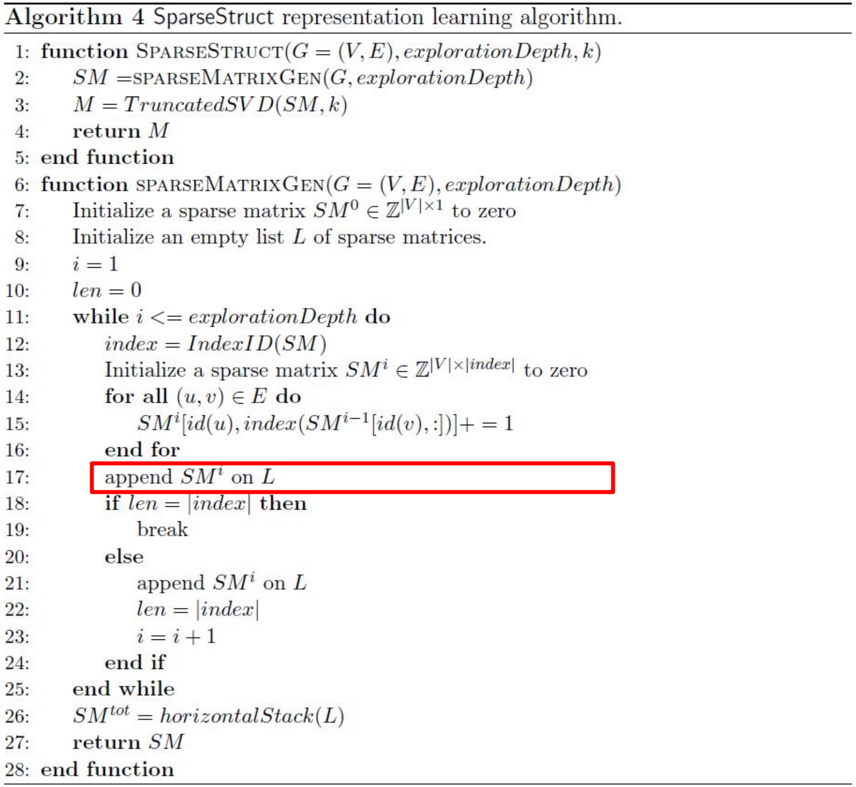


**Algorithm 4** SparseStruct representation learning algorithm.

```
function SPARSESTRUCT($G = (V, E), explorationDepth, k$)
    $SM$ =SPARSEMATRIXGEN($G, explorationDepth$)
    $M = TruncatedSVD(SM, k)$
    return $M$
end function
function SPARSEMATRIXGEN($G = (V, E), explorationDepth$)
    Initialize a sparse matrix $SM^0 \in \mathbb{Z}^{|V| \times 1}$ to zero
    Initialize an empty list $L$ of sparse matrices.
    $i = 1$
    $len = 0$
    while $i <= explorationDepth$ do
        $index = IndexID(SM)$
        Initialize a sparse matrix $SM^i \in \mathbb{Z}^{|V| \times |index|}$ to zero
        for all $(u, v) \in E$ do
            $SM^i[id(u), index(SM^{i-1}[id(v), :])] += 1$
        end for
        append $SM^i$ on $L$
        if $len = |index|$ then
            break
        else
            append $SM^i$ on $L$
            $len = |index|$
            $i = i + 1$
        end if
    end while
    $SM^{tot} = horizontalStack(L)$
    return $SM$
end function
```

# SparseStruct: Sparse Structural Node Representation

- Inspired on Weisfeiler-Lehman Isomorphism Test
- Iterative steps:
  - Generate Index
  - Initialization
  - Sparse vertex description
  - Concatenation
- Truncated SVD
- Theoretically proof:
  - Same as SIR-GN
  - Proof of convergence

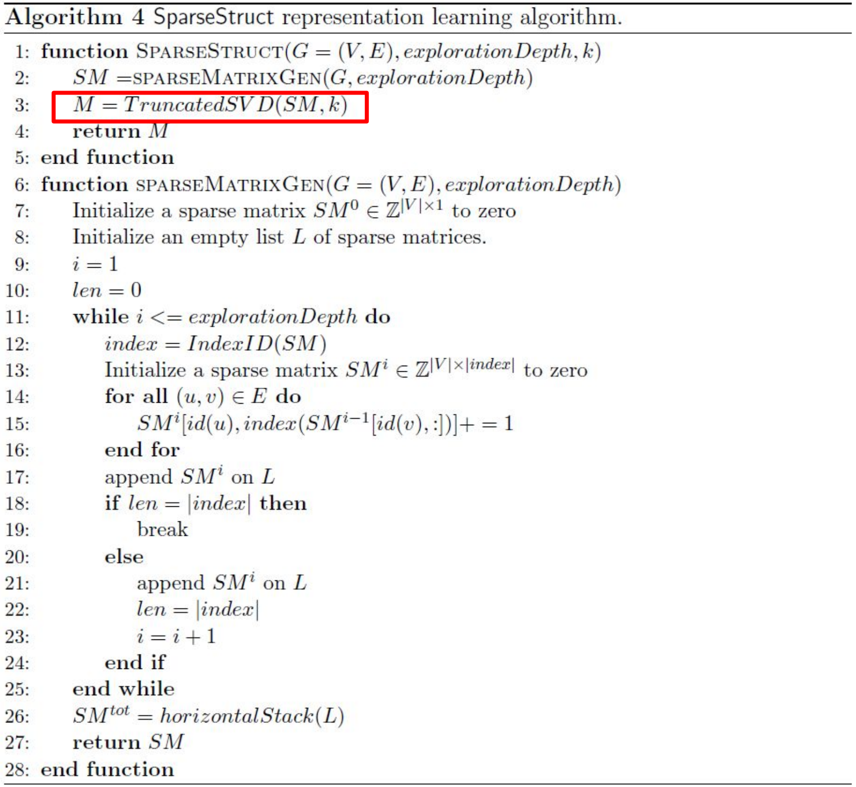


**Algorithm 4** SparseStruct representation learning algorithm.

```
function SPARSESTRUCT($G = (V, E), explorationDepth, k$)
    $SM$ =SPARSEMATRIXGEN($G, explorationDepth$)
    $M = TruncatedSVD(SM, k)$
    return $M$
end function
function SPARSEMATRIXGEN($G = (V, E), explorationDepth$)
    Initialize a sparse matrix $SM^0 \in \mathbb{Z}^{|V|\times 1}$ to zero
    Initialize an empty list $L$ of sparse matrices.
    $i = 1$
    $len = 0$
    while $i <= explorationDepth$ do
        $index = IndexID(SM)$
        Initialize a sparse matrix $SM^i \in \mathbb{Z}^{|V|\times|index|}$ to zero
        for all $(u, v) \in E$ do
            $SM^i[id(u), index(SM^{i-1}[id(v), :])] += 1$
        end for
        append $SM^i$ on $L$
        if $len = |index|$ then
            break
        else
            append $SM^i$ on $L$
            $len = |index|$
            $i = i + 1$
        end if
    end while
    $SM^{tot} = horizontalStack(L)$
    return $SM$
end function
```

# SparseStruct: Sparse Structural Node Representation

- Inspired on Weisfeiler-Lehman Isomorphism Test
- Iterative steps:
  - Generate Index
  - Initialization
  - Sparse vertex description
  - Concatenation
- Truncated SVD
- Theoretically proof:
  - Same as SIR-GN
  - Proof of convergence

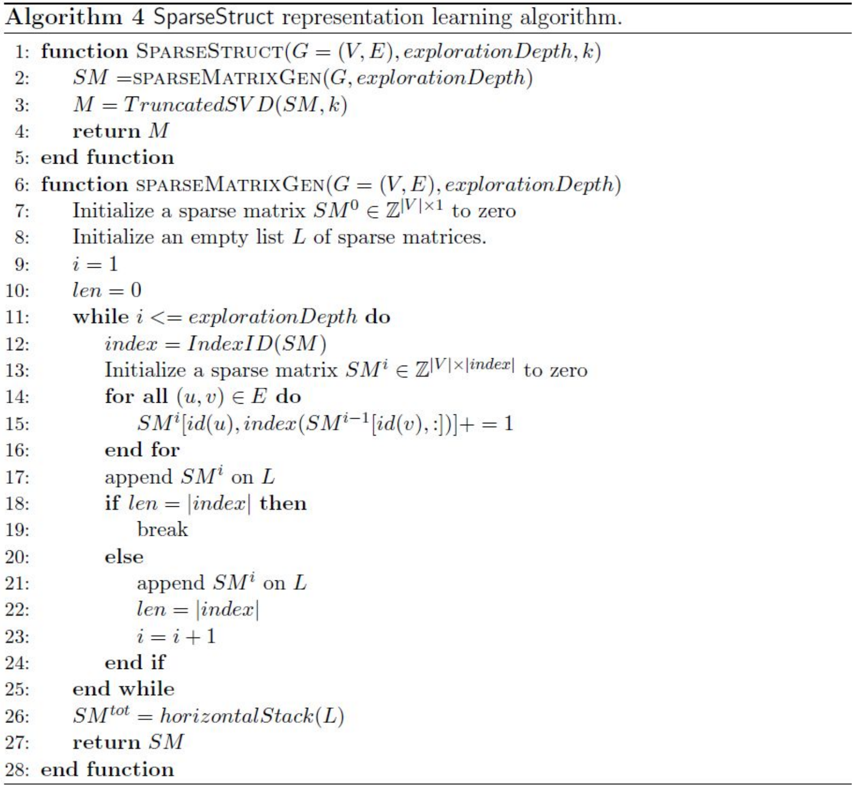


**Algorithm 4** SparseStruct representation learning algorithm.

```
function SparseStruct($G = (V, E), explorationDepth, k$)
    $SM =$ sparseMatrixGen($G, explorationDepth$)
    $M = TruncatedSVD(SM, k)$
    return $M$
end function
function sparseMatrixGen($G = (V, E), explorationDepth$)
    Initialize a sparse matrix $SM^0 \in \mathbb{Z}^{|V|\times 1}$ to zero
    Initialize an empty list $L$ of sparse matrices.
    $i = 1$
    $len = 0$
    while $i <= explorationDepth$ do
        $index = IndexID(SM)$
        Initialize a sparse matrix $SM^i \in \mathbb{Z}^{|V|\times|index|}$ to zero
        for all $(u, v) \in E$ do
            $SM^i[id(u), index(SM^{i-1}[id(v), :])] + = 1$
        end for
        append $SM^i$ on $L$
        if $len = |index|$ then
            break
        else
            append $SM^i$ on $L$
            $len = |index|$
            $i = i + 1$
        end if
    end while
    $SM^{tot} = horizontalStack(L)$
    return $SM$
end function
```

# SparseStruct: Sparse Structural Node Representation

- Inspired on Weisfeiler-Lehman Isomorphism Test
- Iterative steps:
  - Generate Index
  - Initialization
  - Sparse vertex description
  - Concatenation
- Truncated SVD
- Theoretically proof:
  - Same as SIR-GN
  - Proof of convergence



**Algorithm 4** SparseStruct representation learning algorithm.

```
function SPARSESTRUCT($G = (V, E), explorationDepth, k$)
    $SM$ =SPARSEMATRIXGEN($G, explorationDepth$)
    $M = TruncatedSVD(SM, k)$
    return $M$
end function
function SPARSEMATRIXGEN($G = (V, E), explorationDepth$)
    Initialize a sparse matrix $SM^0 \in \mathbb{Z}^{|V|\times 1}$ to zero
    Initialize an empty list $L$ of sparse matrices.
    $i = 1$
    $len = 0$
    while $i <= explorationDepth$ do
        $index = IndexID(SM)$
        Initialize a sparse matrix $SM^i \in \mathbb{Z}^{|V|\times|index|}$ to zero
        for all $(u, v) \in E$ do
            $SM^i[id(u), index(SM^{i-1}[id(v), :])] + = 1$
        end for
        append $SM^i$ on $L$
        if $len = |index|$ then
            break
        else
            append $SM^i$ on $L$
            $len = |index|$
            $i = i + 1$
        end if
    end while
    $SM^{tot} = horizontalStack(L)$
    return $SM$
end function
```