# GloVeNoR: GloVe for Node Representations with Second Order Random Walks

Shishir Kulkarni, Jay Ketan Katariya, and Katerina Potika
Department of Computer Science,
San Jose State University,
San Jose, USA
Email: shishir.kulkarni.2307@gmail.com, jayketan.katariya@sjsu.edu, katerina.potika@sjsu.edu

***Abstract*—We study the community detection problem by embedding the nodes of a graph into a $n$-dimensional space such that similar nodes remain close in their representations. There are many state-of-the-art methods, like node2vec and DeepWalk to compute node embeddings with the use of second order random walks. These techniques borrow methods like the Skip-Gram model, used in the domain of Natural Language Processing (NLP) to compute word embeddings. This paper explores the idea of porting the GloVe (Global Vectors for Word Representation) model, a popular technique for word embeddings, to a new method called GloVeNoR, to compute node embeddings in a graph, and creating a corpus with the use of second order random walks. We evaluate the model's quality by comparing it against node2vec and DeepWalk on the problem of community detection on five different data sets. We observe that GloVeNoR discovers similar or better communities than the other existing models on all the datasets based on the modularity score.**

**Keywords - Community detection, global vectors, word embeddings, node representation, graphs, random walks, clustering.**

## I. Introduction

A graph is a powerful way to represent a real world complex network of entities and relationships between them. For instance, a graph can be used to model protein to protein interactions in bioinformatics, social media users interactions, web pages with their hyperlinks, etc. Moreover, once modeled as a graph we can analyze that complex network to extract useful information and discover patterns of entities and relationships. In order to be able to apply Machine Learning (ML) techniques for the analysis it is required that the input data is represented as $n$-dimensional vectors of relevant features, called feature vectors or embeddings. The quality of the results depends on the number and quality of the selected features.

The task of selecting features using domain knowledge is called feature engineering, and is difficult in the case of graphs because of their size and unordered nature. There has been a lot of research in the past few years focused on computing representation of graph entities using Deep Learning (DL) and ML models.

### A. Natural Language Processing approaches used in graphs

Most of the recent research extends approaches of Natural Language Processing (NLP), that obtains vector representation of words (word embeddings), but in the case of graphs with some tweaks. As a result, some state-of-the-art methods like node2vec [1] have been developed which borrows many of its concepts like the Skip-Gram technique from word2vec [2], a popular word embedding technique used for NLP applications. The word2vec is a predictive model that learns embeddings by trying to minimize the loss of predicting a word given another word. There are other popular techniques, like GloVe [3] and Gaussian embeddings [4] in the NLP domain which follow a different approach for learning word embeddings. Using ideas from the node2vec approach, we research these techniques and examine their portability to graphs.

IEEE/ACM ASONAM 2020, December 7-10, 2020
978-1-7281-1056-1/20/$31.00 

### B. Motivation and problem definition

The Global Vectors for Word Representation (GloVe) is another popular word embedding technique which uses linear arithmetic methods and fundamental statistical properties of a text corpus for preserving semantic relationships between the words. Unlike word2vec, GloVe is transparent and uses global co-occurrence statistics which are fundamental to finding analogies amongst words. Compared to word2vec the model can be easily parallelized and is computationally inexpensive.

The primary objective of this work is to port the GloVe algorithm to generate node embeddings of a graph. We call our method GloVeNoR (Global Vectors for Node Representations). Moreover, we solve the community detection problem by reducing it to a clustering problem, where the nodes of a graph are represented as vectors. We use the k-means clustering algorithm for that purpose. To evaluate the formed clusters (communities) we use two metrics: the modularity score used on the formed communities of the graph, and the silhouette score applied to the clusters of data points (vectors) that represent the graph nodes.

This paper is organized as follows. Section II provides some terminology. In Section III we describe four popular node embedding techniques. Section IV briefly discusses the GloVe algorithm and we introduce GloVeNoR with the implementation details. Section V details the datasets of the experiments and the results. The final Section VI concludes and discusses the interpretation of results.

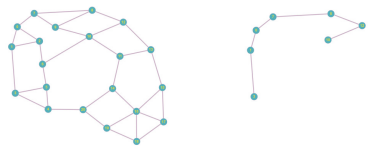

Fig. 1: An example of a random walk

## II. Terminology

Let us start with some terms and notations we will use. A graph is formally defined as a $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges. A graph can be represented using an adjacency matrix or adjacency lists.

**Random Walk**: A random walk is a stochastic process that describes a path consisting of successive random steps on a state space. For example, a random walk on an integer number line will contain a sequence of numbers with an equal probability of moving to the left or right by 1 unit. A random walk can also be considered as a Markov chain. A random walk [5] starting at vertex $v_i$ is denoted by $W_{v_i}$. The length of the walk $\gamma$ is a hyperparameter. A state at any time in a random walk is given by random variables $W_{v_i^1}, W_{v_i^2} \ldots$. These variables represent the stochastic probability of selecting a vertex to advance the walk. The vertices are chosen randomly such that $W_{v_i^{k+1}}$ is selected from the neighbors of $W_{v_i^k}$. Figure 1 shows an example of a random walk which starts at node 3 and has a length of 6.

**Second order random walk**: A random walk with two additional parameters used to interpolate its behavior between Depth First Search (DFS) and Breadth First Search (BFS) and control the revisit frequency of a given node. The first parameter $p$ is called the return parameter. It governs how frequently we revisit a node in the walk. The second parameter $q$ is called the input parameter. It interpolates the behavior of the walk between DFS and BFS. If $q < 1$, the walk behaves more like BFS. If $q > 1$, the walk behaves more like DFS. These two parameters are used to generate a search bias, $\alpha$, that is used to select the next node in the walk non-uniformly. The walk starts at a random node $u$, and it generates a sequence of nodes of length $l$.

**Log-Bilinear model**: In language modeling, a log-bilinear model is a model that predicts the representation of a word based on context words using linear combination and computes the distribution of that word using similarity between prediction and representation of other words.

**Community Structure**: Community structure in a graph can be defined as groups of nodes, such that nodes within a group have denser connections with each other and sparser connections with nodes outside the group.

**Modularity (community) score** [6]: Modularity is the difference between the probability of an edge present in a community $i$ and the probability of a random edge to be present in $i$. Mathematically, it is expressed as:

$$Q = \sum_{i=1}^{k} e_{ii} - a_i^2$$

where,

$e_{ii}$ = Fraction of edges present in community $i$, $a_i$ = Fraction of edges that have one end in community $i$, and $k$ = number of communities

**Silhouette (clustering) score**: The silhouette value is a measure of how similar a data point is to its own cluster when compared to other clusters. It has a range from $-1$ to $+1$, where higher values indicate better clustering. Negative values demonstrate that data points have been assigned to the wrong cluster.

## III. Related Work

Let us briefly describe four state-of-the-art techniques used for computing node embeddings in a graph: node2vec, DeepWalk, DNGR, and Core2Vec. It also discusses the methods that these techniques use for extracting relationships between the nodes of a graph.

**DeepWalk [7]** uses deep neural networks for computing node embeddings. The main contribution is the use of DL in the domain of network analysis for the first time. The algorithm is efficient, scalable, and parallelizable. It introduces the idea of using random walks to capture the relationships between nodes of a graph. It is based on the hypothesis that a substantial number of such random walks starting at different nodes can successfully capture the community structure of the network. We will use that method to compare it against our approach. Additionally, the generation of these random walks can be parallelized with different walks exploring different sections of the graphs at the same time. They are flexible to accommodate any changes in the graph because only the walks containing changed sections have to be modified. This makes the approach scalable.

Although DeepWalk is much better than conventional algorithms, the techniques it employs for information extraction have certain shortcomings. For instance, random walks sample the next node from the given neighbors uniformly, and to distinguish them we call them first order. However, some neighbors might be better probable candidates for advancing the walk. Next we discuss the node2vec algorithm, which uses a different technique to solve this problem.

**node2vec [1]** Similar to DeepWalk it employs the method of local search for extracting neighborhoods and generating sequences from the graph nodes. It introduces second order random walks for exploring node neighborhoods. This technique is very similar to the first order random walks but it is more flexible and provides controls so that the walk behavior can be biased. This is the other method we will compare against our method. The walk starts at a random node $u$, and it generates a sequence of nodes of length $l$. Let's say the walk was at node $t$ and it selected node $v$ as the next node and is

currently at node $v$. There are 3 choices for the next node: $x_1, x_2, x_3$. The probability of choosing the next node, $c_i$, is given by the following equation:

$$P\big(c_i = x | c_{i-1} = v\big) = \begin{cases} \pi_{vx}/Z & if(v,x) \in E \\ 0 & otherwise \end{cases}$$

where:

$\pi_{vx}$: unnormalized transition probability between v and x

$Z$: normalizing constant

There are problems with using the normalized edge weight as transition probability. So a search bias $\alpha$ is used which is given by:

$$\alpha_{pq} = \begin{cases} 1/p & d_{tx} = 0 \\ 1 & d_{tx} = 1 \\ 1/q & d_{tx} = 2 \end{cases}$$

where:

*t: node preceding v*

Figure 2 shows how the search bias is computed using these 2 parameters using the above equation:

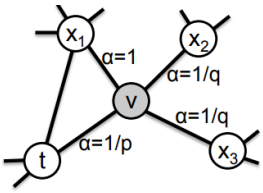


Fig. 2: Search bias $\alpha$ [1]

It is evident that neighborhood exploration strategy plays the most important role in the quality of generated embeddings. Both DeepWalk and node2vec operate on the premise that searching local neighborhoods is enough for extracting relationships amongst nodes. However, next we describes a method that relies on global statistics for information extraction.

**DNGR [8]** (Deep neural networks for learning graph representations) explores different approaches other than sampling and provides a way for capture structural information more accurately for directed graphs. It explores alternative approaches for learning representations based on matrix factorization techniques. DNGR uses a special type of neural network called autoencoders [9]. DNGR model uses the random surfer technique instead of random walks for extracting the relationships amongst the nodes. This technique generates a node co-occurrence matrix.

**Core2Vec [10]** uses a onion-like structure to design a flexible biased random-walk procedure which finds nodes from similar cores as candidates for walk progression. Similar to our approach the intuition of GloVe to utilize global statistics of a network to derive semantic relationships.

**GVNR** Global Vectors for Node Representation) [11] is a model for creating node representations that is based on the GloVe algorithm similar to ours. The difference with our approach is that the they don't use second order random walks to construct the corpus and their approach considers co-occurrence as well as non co-occurrence into account. Additionally, the extension GVNR-t is presented that incorporates the text that nodes might have. In our approach we only consider the structural properties of the nodes in the graph and assume that we have no text for the nodes.

**Graph convolution-based approaches** aggregate information from a node's neighborhood to create an embedding for that node. Some examples of such approaches: Graph Convolutional Networks (GCN) [12], the improved Fast GCN [13], and GraphSAGE [14]. For a unified framework see [15]. The advantage of graph convolution-based approaches is that they utilize node features or attributes in order to generate embeddings. In the networks we use in our experiments node attributes are not present, and therefore we will not use them in our comparison.

## IV. GloVeNoR for node embeddings

GloVe [3] was originally developed in 2014 for computing vector representations of words as a part of Stanford NLP research. GloVe is a global, unsupervised log-bilinear regression model that takes into account both, the local context window similarity and global context similarity for computing the embeddings. The intuition behind GloVe is that global word-word co-occurrence statistics can potentially reveal information about word similarities. As a result, GloVe word embeddings excel at tasks like finding words with similar meaning (nearest neighbors). This is ideal for community detection problems as communities can be viewed as a collection of nodes which are semantically similar to each other.

### A. GloVe for word embeddings

Techniques like skip-gram and context window used in word2vec have a disadvantage of not learning from global statistics. Core2Vec [16] also supports this hypothesis and uses the global core-periphery structure to find similar nodes. As a result, word repetitions and bigger patterns might not be learned with these methods. The way GloVe considers global statistics is by building a word-word co-occurrence matrix. The Algorithm then maximizes the probability of given context word appearing within a window of another word called as the main or center word. The objective of the model is a weighted least squares function. The model also uses a weight function to deal with outliers and co-occurrences that are seen very rarely. Algorithm 1 describes the steps of GloVe algorithm.

The cost function for the model is given as:

$$J = \sum_{i,j=1}^{V} f\big(X_{ij}\big)\big(w_i^T \cdot \tilde{w_j} + b_i + \tilde{b_j} - logX_{ij}\big)^2$$

where: $i$ = main word, $j$ = context word, $w_i$ = main word vector, $w_j$ = context word vector,$X$ = co-occurrence matrix, $b_i$, $b_j$ = bias vectors, $V$ = vocabulary size, $f\big(X_{ij}\big)$ = weight function.

The weight function that was empirically found to be effective is:

**Algorithm 1:** GloVe algorithm for word embeddings [3]

```
1 function GloVe(C, w):
  Input  : C: text corpus,
           w: context window length,
           d: vector dimensions
  Output: V: word vectors
2 voc = buildVocab(C);
3 M = buildCooccur(C, voc);
4 V = initRandomVectors(d)
5 trainModel(M, V)
6 return V;
```

$$f(X_{ij}) = \begin{cases} (x/x_{max})^{\alpha} & x < x_{max} \\ 1 & otherwise \end{cases}$$

### B. Proposed Method - GloVeNoR

In our approach, that we call GloVe for Node Representation (GloVeNoR), we use second order random walks due to the flexibility of the neighborhood exploration. We wanted the sampling behavior of the random walks to be approximated between DFS and BFS so the value of $p$ and $q$ are set to 1. Algorithm 2 provides an outline of the proposed GloVeNoR model. The procedure on line 2 iterates over all the nodes of the input graph $G$, and generates $k$ random walks of length $l$ per node. Parameter $w$ is for the co-occurrence matrix and $d$ is the dimensions of the output vectors.

**Algorithm 2:** GloVeNoR for Graphs

```
1 Procedure GloVeNoR(G, w, d, l, k, p, q, i):
  Input  : G: input graph, w: context window length, d:
           vector dimensions, l: length of random walks,
           k: number of walks per node, p: input
           parameter, q return parameter, i: number of
           training iterations
  Output: V: node vectors
2 corpus = generate2RandomWalks(G, k, p, q, l)
3 M = buildCooccur(corpus, w);
4 V = initRandomVectors(d)
5 trainModel(M, V, i) return V;
```

The generated corpus is used by the co-occurrence matrix generator procedure on line 3. The co-occurrence matrix generator is the gist of the GloVeNoR model. In contrast to [1] and [8], it captures the global co-occurrence statistics for all the nodes. For every node $i$ in a walk, the procedure iterates over all the nodes, $j$, in a window of size $w$ from $i$, and updates the $[i, j]^{th}$ entry of $M$ with the reciprocal of distance between $i$ and $j$. Intuitively, the value for a pair of nodes $i$ and $j$ in $M$ is higher for the nodes co-occurring frequently in the corpus. The output of this procedure is a $|V| \times |V|$ matrix $M$, where $|V|$ is the number of nodes.

In the next step, vectors of dimension $d$ are initialized by the procedure on line 5. These vectors are initialized from a random distribution and optimized to obtain the final embeddings. The procedure on line 4 implements the objective function of GloVe and optimizes it using Gradient Descent for $i$ iterations. This step generates the final node embedding matrix $V$. The size of $V$ is $|V| \times d$ and every row of $V$ represents a vector corresponding to a node in the graph.

Figure 3 shows the workflow used for creating the node embeddings. A toy example of the process is given in the Appendix VI-A.

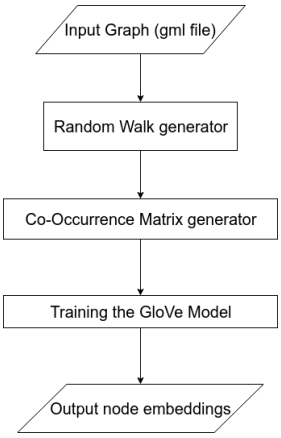


Fig. 3: GloVeNoR Workflow

## V. Implementation and Experiments

We discuss the experiments performed on different real-world data sets and compare our method against two other methods, the DeepWalk and the node2vec, since both also use random walks for the node embeddings. The experiments primarily focus on three different hyper-parameters, the length of random walks ($l$), the context window ($w$), and the vector dimensionality ($d$). Since we do not have any ground truth, we use the silhouette score and the modularity score as our quality metrics.

### A. Implementation Details

We use igraph and NetworkX for reading and performing operations on graph files. We use numpy for computing the co-occurrence matrix. The following list summarizes the implemented modules, as can be seen also in Figure 4:

1) Takes a graph file in GML format as input and generates a csv file containing random walks as the output (corpus). A random walk is a sequence of comma-separated node ids from the graph.
2) Takes this corpus of random walks as inputs, parsed it and generates an $n \cdot n$ co-occurrence matrix.
3) Implement the GloVeNoR model. Takes the co-occurrence file as input, trains the GloVeNoR model using Stochastic Gradient Descent, and creates an output file which contains the vectors for the graph nodes.

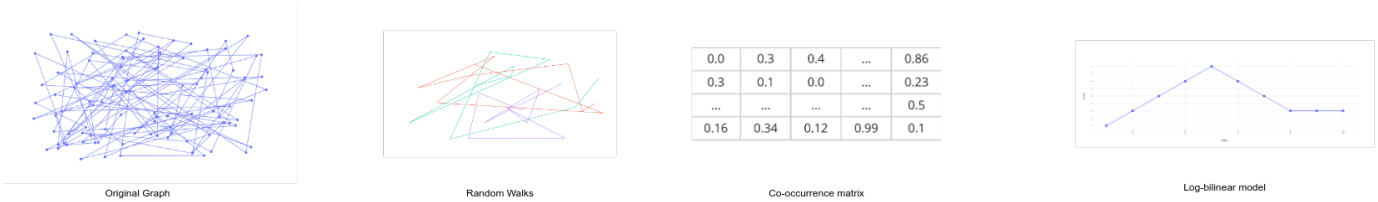


Fig. 4: GloVeNoR Phases

4) Use of utility scripts which perform the clustering operation on these vectors, and compute the DeepWalk and node2vec embeddings for evaluation.

As an application we solve the community detection problem by reducing it to a clustering problem on the node embeddings. Figure 5 shows the use of the generated embeddings for finding communities in the graph.

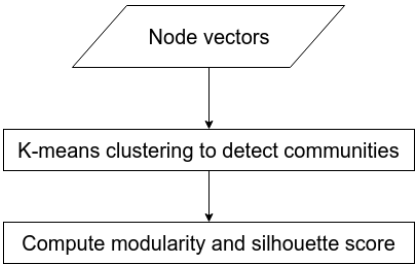


Fig. 5: GloVeNoR Applications

Let us now descibe the various datasets.

### B. Zachary's Karate Club dataset

The Zachary's Karate Club dataset [17] is a famous network dataset curated by Wayne W. Zachary. This dataset has 34 nodes representing members. There is an edge (total 78) between two members if they interact socially outside the club. An interesting fact about this data set is that there was a conflict between the instructor and the administrator which led the latter starting his own club. Thus, the users (nodes) were split into two clubs.

Table I summarizes the observations for different values of the hyperparameters. We used k-means clustering to generate the communities from the embeddings. The $k$ value (number of clusters) is varied from 2 to 8 and the one that produces the maximum modularity is chosen. It can be clearly seen that GloVeNoR has the best modularity values among all methods, with a s $0.4197$ score. It outperforms the modularity score using the Girvan Newman Algorithm [6] which is $0.4013$, and the Louvain Algorithm [18] which is $0.4188$ for this dataset. The number of communities that achieves the best modularity is with $k = 6$.

Another interesting observation is that the model tends to find the local minimum quickly if the size of the context window is closer to or greater than the diameter of the graph. Thus intuitively, we can use this as the window size for further experiments. Moreover, one can see that the walk length is not affecting the results. In most cases, GloVeNoR lags behind

TABLE I: Zachary's Karate Club: Various parameters, walk length $l$, vector dimensions $d$, window size $w$

| $l$ | GloVeNoR (Silhouette score, modularity) | DeepWalk (Silhouette score, modularity) | node2vec (Silhouette score, modularity) |
|---|---|---|---|
| 10 | 0.3940, **0.4197** | **0.3947**, 0.2287 | 0.3859, 0.2043 |
| 20 | 0.4189, **0.4197** | **0.6301**, -0.0210 | 0.4105, 0.2043 |
| 30 | 0.4098, **0.4197** | **0.5375**, -0.0670 | 0.4117, 0.1674 |
| $d$ | | | |
| 2 | 0.5720, **0.3600** | **0.6715**, -0.0670 | 0.6980, 0.1634 |
| 4 | 0.5475, **0.4197** | **0.5662**, -0.0891 | 0.5949, 0.1674 |
| 6 | 0.4180, **0.4197** | **0.5353**, -0.1180 | 0.4664, 0.1674 |
| $w$ | | | |
| 5 | 0.3986, **0.4197** | **0.5363**, -0.0670 | 0.3939, 0.2043 |
| 6 | 0.4307, **0.4197** | **0.5416**, -0.0545 | 0.3856, 0.2043 |
| 7 | 0.4392, **0.4197** | **0.5418**, -0.0545 | 0.3821, 0.1674 |


Fig. 6: Walk length vs Modularity and Silhouette score

in the silhouette score especially compared to DeepWalk. However, since our primary target is to community detection, a good modularity score is desirable.

The information in the tables is graphically shown in the following visualizations in Figures 6, 7, 8, and 9.

The observations about the hyperparameters from the Zachary's Karate Club dataset can be used as the basis for experimentation on the other datasets.

### C. Harry Potter Dataset

The Harry Potter dataset [19] contains $178$ nodes and $2,453$ edges. Each node represents a character in the Harry Potter


Fig. 7: Number of communities vs Modularity and silhouette score

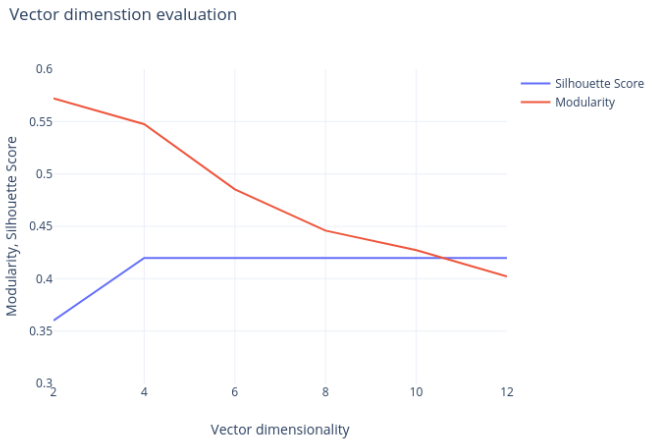


Fig. 8: Vector dimension vs Modularity and Silhouette score


Fig. 9: Context window vs Modularity and Silhouette score

universe. Two nodes have an edge between them if they have some logical connection to each other in the books and the diameter of the network is 3.

TABLE II: Harry Potter dataset: Various parameters, walk length $l$, vector dimensions $d$, window size $w$

| $l$ | GloVeNoR (Silhouette score, modularity) | DeepWalk (Silhouette score, modularity) | node2vec (Silhouette score, modularity) |
|---|---|---|---|
| 10 | 0.3940, **0.2039** | **0.3941**, 0.0022 | 0.1825, 0.0012 |
| 20 | 0.2249, **0.1910** | **0.3972**, 0.0022 | 0.3016, 0.0095 |
| 30 | 0.2242, **0.2114** | **0.3941**, 0.0022 | 0.3102, 0.00431 |
| $d$ | | | |
| 2 | 0.4997, **0.2033** | **0.5986**, 0.0029 | 0.5325, 0.0121 |
| 4 | 0.4223, **0.2018** | **0.5986**, 0.00574 | 0.466, 0.0047 |
| 6 | 0.3653, **0.2109** | **0.4967**, 0.0071 | 0.3797, 0.0097 |
| $w$ | | | |
| 3 | 0.212, **0.2137** | **0.3686**, 0.0008 | 0.283, 0.0109 |
| 4 | 0.2531, **0.2083** | **0.3830**, 0.0025 | 0.2996, 0.0013 |
| 5 | 0.2630, **0.2137** | **0.3953**, 0.0037 | 0.3089, 0.0037 |

Table II summarizes the experiments performed on the Harry Potter dataset. We ran the k-means clustering with values of $k$ from 2 to 20. The results are in agreement with the experiments performed on the Zachary's Karate Club dataset. GloVeNoR has the best modularity score of all the three methods. The length of walks does not have a significant effect on the modularity. The modularity value stabilizes to a good value when the window size is close to the diameter of the graph. Additionally, the silhouette score values generated by GloVeNoR are comparable to the ones produced by node2vec. However, DeepWalk has the best silhouette score overall.

### *D. Facebook Dataset*

The Facebook dataset [20] contains $4,039$ nodes and $88,234$ edges. A node in this dataset represents a Facebook user and an edge joins two users that are friends on Facebook. This dataset was collected by surveying users using an application on Facebook. All the user information is randomized, replaced and anonymized to protect the privacy of the participants. We use the observations from our previous experiments and keep the window size for the co-occurrence matrix generation equal to $8$ which is equal to the diameter of the network. We also keep the walk length to $10$ which is slightly greater than the diameter and should be enough to cover the farthest vertices of the graph. The dimension of the embeddings is set to $10$. The number of random walks per node is $385$. The same values were used for all the three methods while performing the experiments. The second column of the Table III summarizes the observations. GloVeNoR outperforms all the other methods in the modularity score by a significant margin. The silhouette score value for GloVeNoR is comparable to node2vec but lags far behind that of DeepWalk.

### *E. Wikivote dataset*

The Wikivote dataset [21] contains $7,115$ nodes and $100,762$ edges. The diameter of the network is 7. Wikipedia conducts elections to promote a user to administrator status.

A node in this network represents a wikipedia user and an edge between node $i$ and $j$ indicates that user $i$ voted for user $j$ in any election. The graph contains wikipedia voting data until $2008$. Similar to the Facebook dataset, we keep the hyperparameter values close to our observations from the previous results: the length of the walks is $8$, the window size is $7$, the dimensionality of the vectors is $10$ and the number of random walks per node is $218$.

The third column of Table III summarizes the results of our experiments. The results are consistent with our previous experiments. GloVeNoR produces the best modularity scores of all. However, one difference here is that the silhouette score for GloVeNoR is much lower than that of DeepWalk and node2vec.

### *F. Astro dataset*

The Astro dataset [22] is a co-authorship network of papers published on the topic of astrophysics from arXiv. A node represents an author. There is an undirected edge between two nodes, $i$ and $j$, if they have co-authored a paper. The dataset contains $18,772$ nodes and $198,110$ edges. The diameter of the network is $14$. For the parameters: the walk length of is $30$, the window size is $14$, the dimensionality of the embeddings is $10$, and the number of random walks per node is $12$. Observations from the fourth column of Table III are in agreement with the results from previous datasets. GloVeNoR consistently produces better modularity values and DeepWalk has the best silhouette score.

TABLE III: Method Comparison on each dataset, $d = 10$

| Comparison (Silhouette, modularity) | Facebook $w = 8$, $l = 10$ | Wikivote $w = 7$, $l = 10$ | Astro $w = 14$, $l = 30$ |
|---|---|---|---|
| GloVeNoR | 0.1945, **0.665** | 0.1005, **0.3585** | 0.0953, **0.5320** |
| DeepWalk | **0.5716**, 0.2785 | **0.8164**, 0.0011 | **0.7488**, 0.0134 |
| node2vec | 0.2725, 0.3297 | 0.759, 0.0015 | 0.1662, 0.4825 |

## VI. Conclusion

We propose the GloVeNoR model to generate node embeddings for graphs, and we evaluate the quality of the generated embeddings on the task of community detection. We conduct experiments on five different network datasets. We conclude that GloVeNoR consistently produced better modularity scores than node2vec and DeepWalk, both of these methods also use random walks, on all the datasets. GloVeNoR produces similar silhouette scores as that of node2vec, although DeepWalk produces the best silhouette scores of all. We target three hyperparameters in our experiments: length of walks, dimensionality of embeddings, and window size. We observed that the length of walks did not have a substantial effect on the modularity, and the silhouette scores if the value is close to or greater than the diameter of the graph. We also observed that the modularity value stabilized if the window size is equal to or greater than the diameter. Another interesting observation is that the GloVeNoR model trained quickly for higher dimensions.

## References


[1] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864, 2016.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR*, 2013.

[3] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[4] L. Vilnis and A. McCallum, "Word representations via gaussian embedding," in *3rd International Conference on Learning Representations, ICLR*, 2015.

[5] C. Avin and B. Krishnamachari, "The power of choice in random walks: An empirical study," in *Proceedings of the 9th ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, pp. 219–228, 2006.

[6] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, p. 026113, 2004.

[7] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 701–710, 2014.

[8] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 1145–1152, 2016.

[9] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biol. Cybern.*, vol. 59, pp. 291–294, Sept. 1988.

[10] S. Sarkar, A. Bhagwat, and A. Mukherjee, "Core2vec: A core-preserving feature learning framework for networks," in *IEEE/ACM 2018 International Conference on Advances in Social Networks Analysis and Mining, ASONAM*, pp. 487–490, 2018.

[11] R. Brochier, A. Guille, and J. Velcin, "Global vectors for node representations," in *The World Wide Web Conference*, pp. 2587–2593, 2019.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016.

[13] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," 2019.

[14] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *CoRR*, vol. abs/1706.02216, 2017.

[15] S. Mithe and K. Potika, "A unified framework on node classification using graph convolutional networks," in *IEEE International Conference on Transdisciplinary AI (TransAI 2020)*, pp. 67–74, 2020.

[16] M. Rombach, M. Porter, J. Fowler, and P. Mucha, "Core-periphery structure in networks," *SIAM Journal on Applied Mathematics*, vol. 74, no. 1, pp. 167–190, 2014.

[17] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[18] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[19] D. Martin, "Networks." https://github.com/dpmartin42/Networks, 2014.

[20] J. Leskovec and J. J. Mcauley, "Learning to discover social circles in ego networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 539–547, Curran Associates, Inc., 2012.

[21] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Signed networks in social media," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1361–1370, 2010.

[22] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Trans. Knowl. Discov. Data*, vol. 1, Mar. 2007.

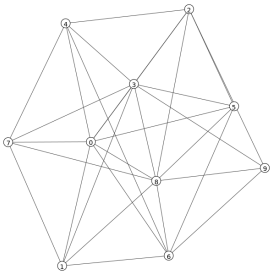

Fig. 10: A sample Erdős-Renyi graph with 10 nodes and 30 edges

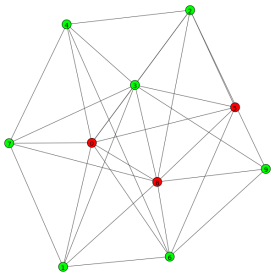

Fig. 11: Generated communities

### A. An example of GloVeNoR

Here we illustrate a sample execution of the algorithm on an example graph. In Figure 10 we have a random Erdős-Renyi graph with $p = 1/2$ (note that this is different than our $p$ in the random walk). It is an unweighted, undirected graph with 10 nodes and 30 edges. The first step of the algorithm it to generate the random walks corpus. We used the values for number of walks per node, $k = 1$ and length of a walk, $l = 5$. A sample of the output of this step is as follows: $0, 8, 2, 3, 5$

$1, 7, 4, 2, 3$
$2, 3, 5, 9, 8$
$3, 0, 8, 1, 7$
$4, 0, 7, 1, 0$
$5, 3, 2, 4, 7$
$6, 8, 0, 3, 4$
$7, 0, 8, 3, 4$
$8, 3, 0, 1, 6$
$9, 5, 2, 4, 7$

The next step generates the co-occurrence matrix using this corpus. We used a window size $w = 1$ for the above corpus. The generated co-occurrence matrix is given by:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 2.0 | 0.0 | 3.0 | 1.0 | 0.0 | 0.0 | 2.0 | 4.0 | 0.0 |
| 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 3.0 | 1.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 4.0 | 3.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 3.0 | 0.0 | 4.0 | 0.0 | 2.0 | 3.0 | 0.0 | 0.0 | 2.0 | 0.0 |
| 1.0 | 0.0 | 3.0 | 2.0 | 0.0 | 0.0 | 0.0 | 3.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 1.0 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 |
| 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 2.0 | 3.0 | 0.0 | 0.0 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4.0 | 1.0 | 1.0 | 2.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 0.0 | 0.0 | 1.0 | 0.0 |

After obtaining the co-occurrence matrix, a random vector of $d$ dimensions is initialized for every node. These vectors are optimized in the final step using the co-occurrence matrix and the cost function. The final step trains the GloVeNoR model using the co-occurrence matrix. This step generates $d$-dimensional embeddings of the graph nodes. For the above example, the dimensionality, $d$, of the embeddings is 2. The generated vectors are given by:

| | |
|---|---|
| 0.00032430148178421857 | 0.7938853234927894 |
| 0.47120730004410316 | 0.07747351511500729 |
| 0.6995203607385199 | 0.22675897204604747 |
| 0.40398774088300593 | 0.3402914202062996 |
| 0.3684166382730359 | −0.0466443816755288 |
| −0.09899558914305986 | 0.3546439090539501 |
| 0.32795573825915464 | 0.32323114306831036 |
| 0.3240960932604696 | 0.21676116252743483 |
| −0.15893379881151024 | 0.7265621806582748 |
| 0.30177281552368374 | 0.3173293382685133 |

These vectors are clustered using the k-means clustering algorithm. The generated clusters are communities in the graph. In the above example, we are using a k value of 2. The generated communities are shown in Figure 11.