V2V: Vector Embedding of a Graph and Applications

Trong Duc Nguyen
Department of Electical and Computer Engineering
Iowa State University
trong@iastate.edu

Srikanta Tirthapura
Department of Electical and Computer Engineering
Iowa State Univeristy
snt@iastate.edu

***Abstract*—We present V2V, a method for embedding each vertex in a graph as a vector in a fixed dimensional space. Inspired by methods for word embedding such as `word2vec`, a vertex embedding is computed through enumerating random walks in the graph, and using the resulting vertex sequences to provide the context for each vertex. This embedding allows one to use well-developed techniques from machine learning to solve graph problems such as community detection, graph visualization, and vertex label prediction. We evaluate embeddings produced by V2V through comparing results obtained using V2V with results obtained through a direct application of a graph algorithm, for community detection. Our results show that V2V provides interesting trade-offs among computation time and accuracy.**

I. Introduction

A graph is a fundamental abstraction in computing, and graph algorithms are among the most widely used methods for data analysis. One difficulty with many graph problems is that their algorithmic complexity can be quite high. For instance, various versions of graph community detection are NP-complete [1] – this includes the version that maximizes the *modularity* metric [2]. Hence, exact algorithms for such problems are not scalable to large graphs, and approximate algorithms are necessary. The time complexity of graph algorithms can be a bottleneck for approximate algorithms and heuristics also – the algorithms of Clauset et al. [3] and of Girvan and Newman [4] for community detection both have a time complexity of $O(mn)$ in the worst case, where m is the number of edges and n the number of vertices in the graph (though in typical cases, the algorithm of [3] is expected to be faster). Further, graph algorithms may be sensitive to small errors in inputs, or to missing data. In addition, data may include, in addition to edges and hyperedges, directions on the edges, timestamps, weights and labels. In such cases, it may not be easy to modify a graph algorithm designed for an undirected and unweighted graph to incorporate additional properties. Overall, the complexity of graph problems has led to a rich literature on graph algorithms, with techniques tailored to the context and to the problem.

Machine learning (ML) is an approach to data analysis where there are a few general problems overall, such as classification, regression, and clustering, and there is a vast repository of methods that can be applied, based on the type of data involved, the data distribution, and the result desired. The techniques typically do not come with approximation guarantees, but are quite flexible to work with different types of data.

In this work, we consider how we can leverage ML methods to solve problems on data that are naturally represented as graphs. In particular, we consider embedding a graph into a vector space, representing each vertex in a graph through a vector. If this can be done, we can leverage the power of various ML techniques to analyze data that was originally structured as a graph. We investigate the power and weaknesses of this approach, and try to understand to what extent such a translation from graphs to a vector space can help in solving a graph problem, and how such methods compared with direct graph algorithms.

We present V2V (Vertex to Vector), an approach for learning the vector representation for graph-based data, and its applications. By vectorizing graph-based data, we are able to take advantage of the machine learning techniques to: (1) solve problems on graph-based data with useful tradeoffs between runtime and accuracy, when compared with exact graph algorithms; (2) gain a new perspective on data that is useful in tasks such as visualization, and (3) handle errors in data in a natural manner, through the use of robust machine learning techniques.

Inspired by Word2Vec [5], a vector representation of words that are learnt from the linguistic context of each word in a large corpus, we leverage the Continuous Bag of Words - CBOW [6] neural network model, to learn the embedded vector for each vertex in the graph, in which each vertex is represented by a fixed-dimensional vector embedding in a continuous space. We show the utility of V2V in a variety of tasks, including community detection, graph visualization, and feature prediction among vertices, through evaluating it on synthetic as well as real-world networks. We note that there have been multiple prior works on finding vector embeddings of graphs [7], [8], [9], [10], [11] – a comparison is presented in Section VI. A main difference with our work is that previous works do not have a detailed study of applications of such vector embeddings, especially a comparison with a direct graph-based approach, like we do here.

Community detection is a key analysis task in understanding the structure of complex networks. Due to its diverse applications, community detection is well studied problem [4], [1], [3], [12], [13], [14]. In contrast with existing algorithms

that work directly on the graph, we consider an approach that uses a clustering algorithm in the embedding space to derive communities that can be mapped back to the original graph space. Clustering in a vector space is well studied, and there are many efficient algorithms that scale to large data sets, for instance, Lloyd's algorithm for k-means clustering [15], and `k-means++` [16], [17]. Our finding is that the V2V approach to community detection yields good quality communities. While these are not as precise as the communities that are discovered by graph-based algorithms [4], [3], they run in a fraction of the time taken by graph-based algorithms (20x faster on graphs that we considered, with 1000 vertices and 25000 edges).

Embeddings produced by V2V give us a new perspective on the data, which is also helpful in **graph visualization**. This is a natural outcome of the vector representation of vertices, since the visualization of vectors is well-studied and there are principled approaches based on the Principal Component Analysis (PCA) [18] and t-SNE [19] to explore non-obvious aspects of the data. By projecting data along the first few principal components, we gain interesting visualizations that demonstrate relationships between vertices in the original graph.

In summary, our contributions include:

- V2V model, an approach to represent graph-based data in a low-dimensional vector space. V2V stands for a class of approaches which learn a vector representation of a vertex with the help of random walks in the network.
- A study on the application of V2V in various applications, including community detection, data visualization, and feature prediction, using machine learning approaches on the vector representation of vertices.
- An empirical study to evaluate the performance of V2V applied to community detection, when compared to direct graph-based approaches.

Clearly, the embedding approach applies to only a certain class of graph problems. We lose some basic graph structure by transforming to the vector representation, *e.g.,* we cannot exactly find the 1-hop neighbors for a given vertex, and there is not much reason to expect this representation to help identify shortest paths between vertices. However, this representation captures certain aspects of the global structure of the graph, such as graph communities, that can lead to useful tradeoffs between time and accuracy, and improved robustness for some of these problems.

II. Learning V2V representation vector

In this section, we present our approach to learning vector embeddings of vertices. The goal is to represent each vertex by a vector in a low-dimensional (10-1000 dimensions) space, in which the structure of the graph is captured by the positions of the vectors in the embedding space. The representation vector of a vertex is learned from the *context* where that vertex appears. In general, the context of a node can be constructed through the interactions this node is involved in.

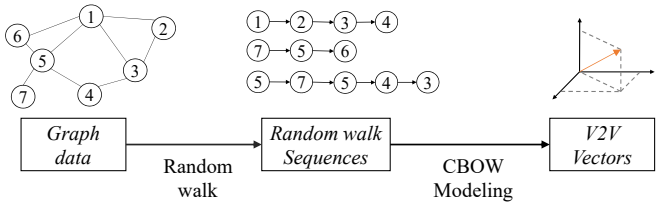


Fig. 1: Learning procedure overview. First, a biased random walk extracts random paths from graph-based data. Then the CBOW model is applied in random path sequences to learn the embedded vectors.

For instance, in a computer network consisting of client machines and workstations machines, when a workstation receives a request from a client, it may serve the request directly, or send a sub-request to other workstations, in order to serve the client's request. The "context" of a node in this scenario could be defined as the other nodes that are involved in servicing the same request. If the network is modeled as a graph, each request forms a path in the graph, and nodes that service a request appear in a sequence in the corresponding path. In this example, node contexts are already provided in data in the form of paths through the network.

For a general input graph where such path data is not available, we generate contexts for nodes with the help of random walks. Figure 1 shows an overview of our approach. We first generate a set of random walks in the graph – each random walk is a sequence of vertices. In these sequences, for each vertex, the surrounding vertices are considered as the context for the vertex. Using these sequences, we learn the vector embedding for each vertex through the Continuous Bag-Of-Words (CBOW) [5] model. The outcome is a vector embedding representing each vertex in the original graph.

A. Constrained Random Walks

Let G denote the input graph on edge set (V, E). We use random walks on G to generate vertex sequences. Let $t > 0$ be an integer parameter. Starting from each vertex in G, perform t independent random walks, for a total of $t \times |V|$ random walks. Each random walk is of length ℓ. Note that the results can be expected to be similar if we choose $t \times |V|$ random walks, each starting from a uniformly randomly chosen vertex in the graph. In our work, the default values of t and ℓ are set to 1000.

The basic random walk for an undirected graph starts from a vertex and moves to a randomly chosen neighboring vertex, and continues doing so for a specified number of steps. This can be constrained to take into account a variety of graph properties such as:

– Edge direction: If the original graph is directed, the random walk follows the direction of the edges in each step, by choosing a random outgoing edge from each vertex, rather than choosing a random neighbor. The random walk terminates when there is no outgoing edge from a vertex.

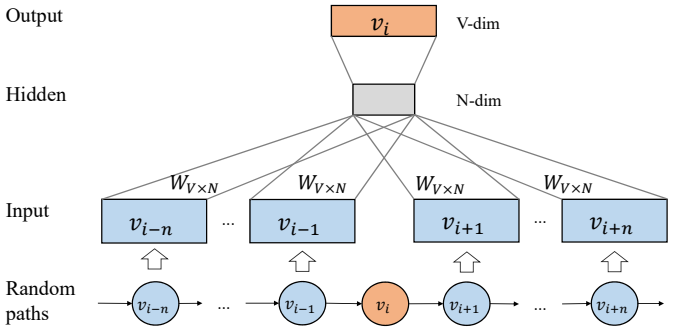


Fig. 2: CBOW applied to graphs: For each vertex v_i as the output, the surrounding vertices that appear within n steps from v_i in a random walk sequence, are fed to input layer of the neural network.

– Edge/vertex weight: Each edge may have a weight, which measures significance of the relationship between two vertices that the edge connects. *e.g.,* traffic between 2 nodes in a network; number of times an author cites the works of other author; *etc.*. It is possible to take weights into account in a random walk – for instance, we assign the probability of choosing an edge to be proportional to the edge weight. In case the edge is unweighted but the vertices have associated weights, we use a similar procedure: in each step, the probability of choosing an edge is proportional to the weight of the target vertex.

– Time stamps: If edges have timestamps associated with them, we can further constrain the random walk, by requiring them to obey the timestamp order. For each step of the random walk, the preceding steps are *earlier* in time. Such a random walk has a length that is bounded by the length of the longest time-increasing path in the network. We can also impose additional constraints on the random walk, by requiring that two consecutive edges in the random walk have timestamps that are within a *time window* threshold of each other. This creates paths that have greater relevance in a temporal sense.

Thus the context for each vertex can be created in a flexible manner, through customizing them according to the problem characteristics. This flexibility, of being able to work with different types of graphs, including undirected, directed, and time-stamped graphs, is a strength of V2V for graph processing.

B. Learning V2V Vectors from Random Walks

Continuous Bag Of Words (CBOW), introduced by Mikolov et al. [6], [5], is a full connected feed-forward neural network that was originally proposed and used in natural language processing. Once it has been trained on a large text corpus, CBOW can produce for each unique word, a vector in a fixed-dimensional space in which linguistic contexts of words are preserved. It represents words by encoding the contexts of their surrounding words into vectors. CBOW is used in the popular Word2Vec word embedding. See Fig. 2 for the use of CBOW in our work.

The CBOW neural network architecture has three layers: input, hidden, and output. The input layer takes in a window of n words preceding the current word w_i and n words succeeding w_i in a sentence. The output layer is for w_i. Each word is encoded into the model using its index in a vocabulary. The index vector for a word is an $1 \times D$ vector with D being the vocabulary's size, and only the index of that word is 1 while the other positions of the index vector are 0s. The embedded vector for each word w_i is the trained weight in the hidden layer with N dimensions, which is the number of the dimensions of the vector space. To compute embedded vector for w_i, CBOW first takes the sum of the vectors of the 2n input context words, and computes the product of the sum vector and the input-to-hidden weight matrix $W_{V\times N}$ (shared for all words):

$$V(w_i) = \sum_{j=(i-n)\ldots(i+n), j\neq i} w_j \cdot W_{V\times N}$$

$V(w_i)$ is the embedded vector for w_i. The window size is $2n$. $W_{V\times N}$ is the input-to-hidden weight matrix. The training criterion is to derive the input-to-hidden weight matrix $W_{V\times N}$ and the hidden-to-output weight matrix $W'_{N\times V}$ such that CBOW correctly classifies the current word $w = w_i$ for all words. Further details can be found in [5].

We adapt the CBOW model to learn vertex embeddings as follows. The vocabulary is the set of all vertices in the graph, each "word" is a vertex, and each "sentence" is a path. The training data consists of a sequence of paths. Since we use stochastic gradient descent for training, the training time depends on the size of the training corpus, which depends on the number of random walks and the length of the random walks. We set the window size $n = 5$ in default, while we various the number of dimensions in our experiments.

III. V2V for Community Detection

Community structure is an important property of networks, which can often be split into groups of vertices that are closely knit with strong connections within a group, but with weaker connections across groups. Identifying the community structure in a network is an important task that is widely studied in different types of networks, including social networks [20], [21], collaboration networks [22], the Internet and the Web [23], [24]. This can be posed as an unstructured machine learning problem.

There is much prior work on detecting community structure within a network. As an example of the different approaches considered, Clauset *et al.* [3] presented a top-down approach to decompose a graph into dense portions by starting with the entire graph as a single cluster, followed by recursive partitioning of clusters. Girvan and Newman [4] presented a bottom-up approach that starts out with each vertex in its own cluster, and arrives at the final result through recursive merging. A combination of top-down and bottom-up approaches has been proposed by Sobolevsky *et al.* [12].

We propose a novel approach to community detection based on V2V. The idea at a high level is simple: we first generate

the vector embeddings for each vertex in the graph using V2V; it is expected that vertices that belong to dense communities in the original graph are close to each other in the embedding space. We then apply a clustering algorithm in the embedding space to cluster the vectors. Vertices whose embeddings belong to the same cluster are grouped together as a community in the graph.

More concretely, let $V = v_1, v_2, ..., v_n$ be the vertex embeddings. We use k-means clustering, which partitions V into $k < n$ clusters $S = \{S_1, S_2, ..., S_k\}$ such that the sum of the squares of the distances to the cluster centers are minimized. Let μ_i denote the cluster center for cluster S_i. The problem is to minimize:

$$\min_S \sum_{i=1}^{k} \sum_{v \in S_i} \|v - \mu_i\|^2$$

where $\mu_i = \frac{1}{|S_i|} \sum_{v \in S_i} v$. We use Lloyd's algorithm [25] to solve the k-means optimization. Since Lloyd's algorithm may lead to a local minimum, to improve the quality of the results, we repeat the algorithm 100 times and choose the best solution (using the above metric).

We conducted experiments to evaluate the effectiveness of V2V representation on a synthetic dataset, in which we have the ground truth for the community structure.

A. Synthetic Dataset

In order to observe the performance of V2V on graphs with various densities, we generated a sequence of graphs, each with a well-defined community structure, but with different levels of strength within a community. Each graph has 1000 vertices that are partitioned into 10 groups (communities), $G_1, G_2, ..., G_{10}$, with 100 vertices per group. Each group G_i is converted into an α quasi-clique, by generating uniformly at random $\alpha \times |G_i|(|G_i| - 1)$ edges connecting vertices in that group. Note that $|G_i|(|G_i| - 1)$ is number of edges needed to make G_i a clique; for parameter $0 < \alpha < 1$, a subgraph is an α *quasi-clique* when it contains an α fraction the number of edges required to make it a clique. $\alpha = 0$ leads to a purely random graph where the density of each subgraph approximately equal the density of the whole graph. Meanwhile, $\alpha = 1$ makes each group a clique. In addition to intra-group edges, there are 200 edges connecting vertices between different groups.

Figure 3 shows a visualization of graphs with different values of α, using the ForceAtlas algorithm [26]. The whole network is connected, but the connections are heavily within a community. In Figure 3(c), α is 1.0, which makes each group a clique, while smaller values of α, such as $\alpha = 0.1$ in Figure 3(a) and 0.5 in Figure 3(b) give us a weaker connection within each group. The algorithms are given these graphs, but are not told which vertices belong to which community.

B. Evaluation

We conducted experiments to evaluate how well the V2V vector embeddings capture the community structure of vertices as they are in the graph. Ideally, the community structure in the graph maps closely to the clusters in the embedding space. After applying V2V to learn the vector embeddings from the generated graph, we use Principal Component Analysis (PCA) [18] to visualize the arrangement of the learned vectors. Figure 4 shows the projection of the vectors along the first and second principal components with the setting of $k = 10$, vector embeddings are 50 dimensions, that are learned from the graph with $\alpha = 0, 1$. The centroid μ_j of each cluster as well as the boundary between clusters are highlighted. We colored the vectors according to the group the vertex belonged to in the graph (the ground truth). Note that the color was not provided as input to the algorithm – they were added during visualization for presentation purposes only.

It can be seen that the vectors naturally separated into clusters, even when viewed through a 2-dimensional projection. Since the learning procedure for V2V is unsupervised, in which each vertex is treated equally and V2V has no information about the community structure of the graph, the arrangement of the vectors shows that the V2V embedding captures the community structure of the original graph. We showed further applications of V2V in data visualization in Section IV.

We conducted experiment to quantitatively measure the relationship between community structure in the graph and the arrangement of vectors in vector embedding space. In order to quantitatively measure how well the clusters in vector space capture the dense structures of the graph, we compare the ground truth community $G_1, G_2, ..., G_{10}$ of vertices in the original graph with the clusters $S_1, S_2, ..., S_k$ of the vector embeddings. We use pairwise *precision* and *recall* to measurement how accurate the clusters matches the communities. In other words, for a given pair of vertices that are from the same community G_i, whether or not the corresponding vectors belong to the same cluster S_j.

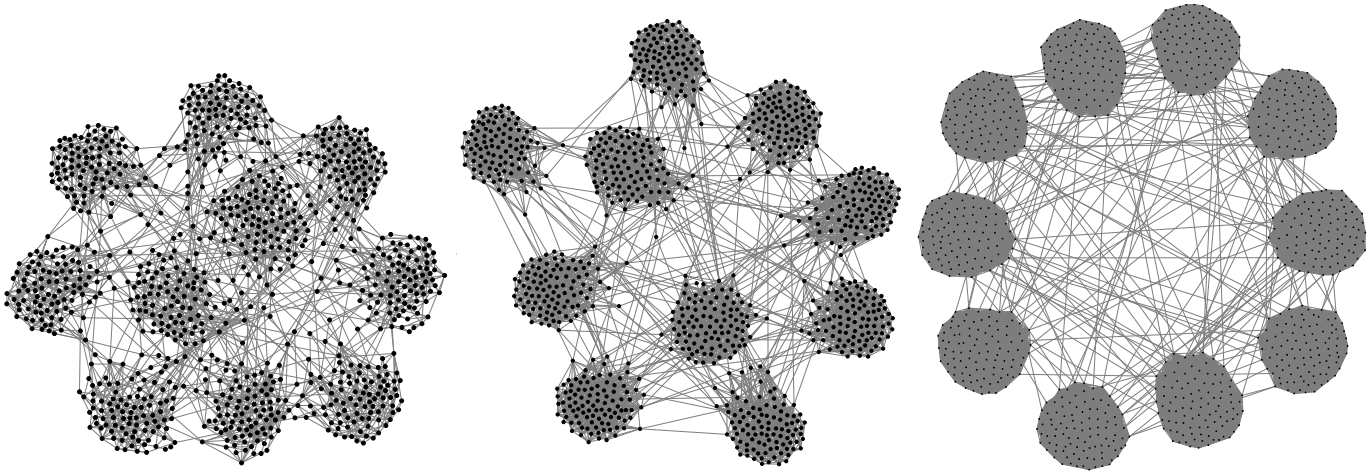
Precision is defined as the fraction of vector pairs that belong to the same community out of all vector pairs have been assigned to the same cluster. *Recall* is the fraction of vertex pairs that have been clustered together, out of all vertex pairs that belong within the same community (group).

$$Precision = \frac{|\{(v_i, v_j) \mid \exists G_t,\ S_k : v_i, v_j \in G_t; v_i, v_j \in S_k\}|}{|\{(v_i, v_j) \mid \exists S_k : v_i, v_j \in S_k\}|}$$

$$Recall = \frac{|\{(v_i, v_j) \mid \exists G_t,\ S_k : v_i, v_j \in G_t; v_i, v_j \in S_k\}|}{|\{(v_i, v_j) \mid \exists G_t : v_i, v_j \in G_t\}|}$$

The higher *precision* the more likely that vectors, which are clustered together, correspond to vertices that actually belong to the same group. Meanwhile, the higher *recall* the more likely that vertices belonging to the same group have vector embeddings that are clustered together.

Figures 5 and 6 show the precision and recall, respectively, with different settings of α and number of dimensions of the V2V vector space. Not surprisingly, as α increases, both the precision and the recall of our algorithm increase. That is because a higher α naturally leads to a stronger community structure, which should be easier to find.



(a) $\alpha = 0.1$ (b) $\alpha = 0.5$ (c) $\alpha = 1.0$

Fig. 3: Synthetic random graphs with 10 communities each. The strength of a community is controlled by α – each community is an α-maximal clique on 100 vertices.

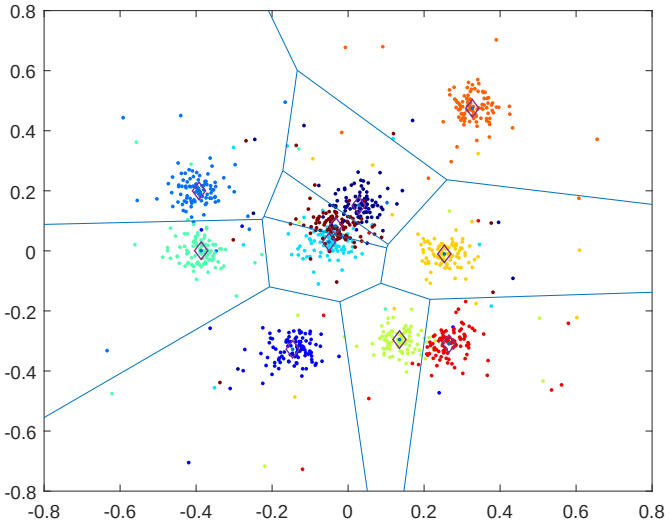


Fig. 4: PCA visualization of the V2V vector embeddings of the vertices (dimension = 50, the top two principal components are chosen) when $\alpha = 0.1$. The color of a vertex indicates the community it belongs to in the original graph.

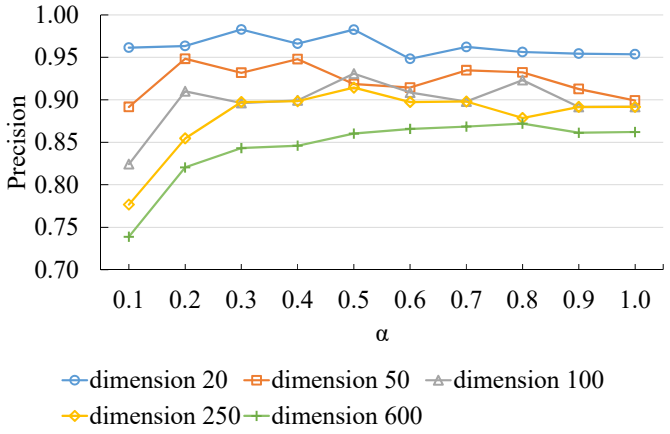


Fig. 5: Precision of V2V community detection as a function of α, the strength of a community.

Figure 7, shows how the training time of V2V is sensitive to the structure of the original graph. With small α, the input graph has weak community structure and V2V takes more training time to learn the vector embeddings. When α is increased, the input graph has stronger community structure, and V2V requires less training time to converge into a stationary stage and provide higher accuracies, both precision and recall, in detecting the communities.

C. Comparison with Graph-based Algorithms

We compared the results of V2V community detection, though clustering in vector space, with the results of the traditional graph-based algorithms that aimed to find the communities in the graph. We compared our approach, with two algorithms that work directly on the graphs, the CNM algorithm and the Girvan-Newman algorithm – these two methods are instances of the top-down and bottom-up methods for community detection, respectively. We used the implementation of CNM and Girvan-Newman from [27]. Table I shows the results comparing our approach using the V2V embedding, with graph-based algorithms.

– Accuracy: The graph-based algorithms have better results in term of precision and recall. Indeed, for most instances, CNM and Girvan-Newman can correctly detect the communities fitting the ground truth. Our approach is lower in precision and recall, averaging 95.2% and 98.6%, respectively, over the different values of α considered.

– Runtime: On the other hand, V2V tremendously outperforms the graph-based algorithms in running time. While CNM

TABLE I: Community Detection: Comparison of V2V on 10-dimensional vector space to CNM and Girvan-Newman algorithms. Time is shown in seconds.

α	V2V				CNM			Girvan-Newman		
	Precision	Recall	Trainning time	Running time	Precision	Recall	Running time	Precision	Recall	Running time
0.1	0.961	0.973	341.265	0.00765	0.998	0.998	464.0064	0.998	0.998	447.928
0.2	0.948	0.985	334.519	0.00743	1.00	1.00	1133.0545	1.00	1.00	1131.316
0.3	0.983	0.996	325.651	0.00746	1.00	1.00	2244.0256	1.00	1.00	2151.287
0.4	0.948	0.989	285.015	0.00722	1.00	1.00	3567.8170	1.00	1.00	3490.294
0.5	0.983	0.995	258.618	0.00664	1.00	1.00	5090.1243	1.00	1.00	4998.118
0.6	0.913	0.976	250.185	0.00725	1.00	1.00	6697.0080	1.00	1.00	6600.254
0.7	0.931	0.984	229.204	0.00716	1.00	1.00	8156.5426	1.00	1.00	8014.215
0.8	0.913	0.977	226.910	0.00700	1.00	1.00	9666.2279	1.00	1.00	9399.932
0.9	0.983	0.995	223.136	0.00722	1.00	1.00	10680.3451	1.00	1.00	10608.498
1.0	0.954	0.988	219.177	0.00706	1.00	1.00	11693.1782	1.00	1.00	11628.913
avg.	0.952	0.986	269.368	0.00721	1.000	1.000	5939.2330	1.000	1.000	5847.0755

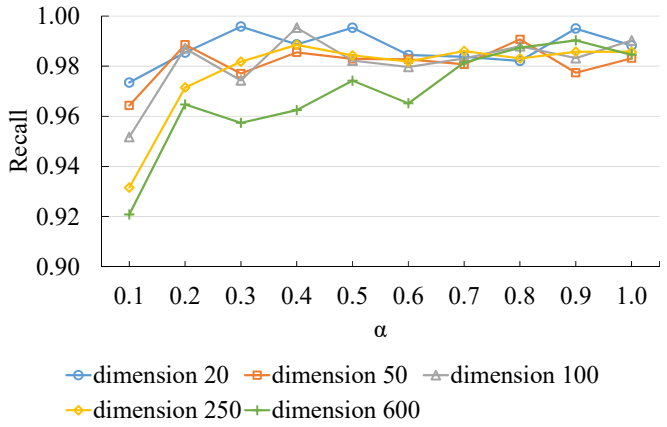


Fig. 6: Recall of V2V community detection as a function of α, the strength of a community.

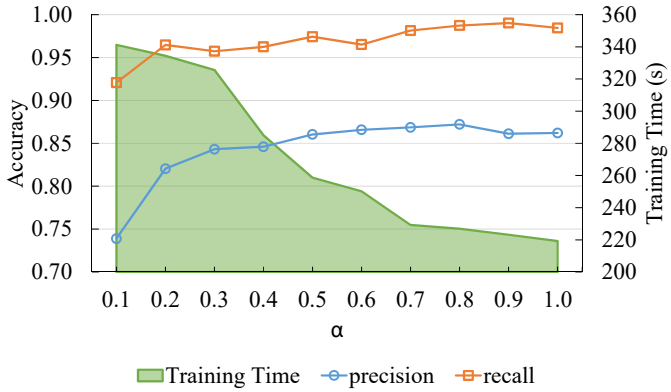


Fig. 7: Accuracy and training time of V2V, 600 dimensions, as a function of α. As α increases, the communities get stronger, and the training time decreases.

and Girvan-Newman require a run time of multiple hours, V2V takes an average of 4 minutes for the learning phrase, less than 0.01 seconds for the actual clustering! Further, we note that the learning phase is a one-time cost. Once vector embeddings are computed, they can be used without change for many further tasks, including community detection, visualization, etc, as we explain further.

– **Graph Size:** We note that the runtimes of the graph-based algorithms increase as the number of edges increase. IN particular, as α changes from 0.1 to 1 (i.e. number of edges increases by a factor of 10), the runtimes increase by a factor of more than 20. In contrast, the training time for V2V decreases as α increases. This is because the stronger community structure within the graph allows the training to reach a (locally) optimal value quicker for the underlying gradient descent algorithms. Thus, we can expect this approach to community detection to scale more easily to larger graphs.

– **Errors:** We can also expect the V2V approach to be less sensitive to errors in data than the pure graph-based approaches. This aspect needs further investigation.

IV. V2V for Data Visualization

The V2V vector embeddings give us a new point of view to the data that captures the structure of the graph and lets us explore relationships between vertices. In this section, we demonstrate the usefulness of V2V vector embeddings in visualizing vertices in a graph, with the help of Principal Component Analysis (PCA) [18].

PCA is a statistical technique that is widely applied in data analysis. PCA projects data into a number of orthogonal components, called the *principal components*, which enables to view data in a low dimension. One importance usage of PCA is to display data as points [28], [29] in two or three dimensional space. We applied PCA on the generated V2V vector embeddings, to get the set of principal components. By projecting the vector space into the first and second components, we have the 2-dimensional (2D) visualization. Similar, with the first, second and third components, we have 3-dimensional (3D) visualization.

In Section III, we showed the utility of V2V visualization by visualizing data from a synthetic graph. As seen, the visualization allows us to see the community structure in data, even in a 2D representation. In this section, we further

demonstrate the usefulness of V2V visualization on a real-world dataset.

A. Visualizing the OpenFlights dataset

We collected a real-world dataset, which we call the OpenFlights dataset, from website OpenFlights.org [30]. The dataset contains information about different airlines and their flight routes connecting airports around the world. The flight route map can be viewed as a graph of more than 67 thousand directed edges (routes) and more than 10 thousand nodes (airports).

With the OpenFlights dataset, we first use V2V to derive the representation vectors for airports. The associate information of the airports, such that location, attitude, nation, etc., are not included in the learning phase. We consider the routes map as the directed graph where airports, as the vertices, are connected by flight routes as the directed edges. Then, we applied PCA [18] in the embedded vector space to generate the illustration of the airports around the world.

Figure 8(a) shows the 2D visualization and Figure 8(b) shows the 3D visualization of the 100-dimensions embedded vector space. In both figures, we color the vectors by the continents of the corresponding airports. Interestingly, representation vectors of airports from each continent are well grouped together, which show that the distances between the embedded vectors reflect geometric proximity between airports. The embedded vectors are learned from the nature of the air flight routes between airports. None of the geographic information of an airport such as its country, continent, latitude, and longitude, were included in the training input.

V. V2V for Feature Prediction

In this section, we evaluate V2V in the application of feature prediction. We aim to predict the unknown value of a particular feature of a vertex, using known values from other vertices. Feature prediction is practically useful in multiple scenarios. The most common use case is dealing with missing data. When the label of a vertex is lost, it can be recovered based on its relationships with other vertices, whose labels are known.

We formulate the problem of predicting the vertex's feature as a classification task of the corresponding vector embedding. In the embedded V2V vector space, we used the well-known k-nearest neighbors (k-NN) algorithm [31] to classify the vector's label, using supervised learning on the set of vectors whose labels are known. With the k-NN algorithm, the predicted label of an unlabeled vector is the majority vote from its k, $k > 0$, nearest neighbor vectors. In special case $k = 1$, the given vector is simply assigned the same label as its nearest neighbor. The proximity of vectors is measured by the cosine distance. Even k-NN is not the best accuracy classification algorithm, from our knowledge, though such a simple algorithm still yield impressive results.

We conducted an experiment using the OpenFlights dataset [30], in which each embedded vector are labeled by the country of the corresponding airport that vector represents. We hide the country labels of a portion of airports and try to predict those hidden information using the learned V2V vector embeddings and the known country labels of the others. We conducted 10-fold cross validation in which we randomly divided the airports into 10 folds that are equal in size. Each time, one testing fold hides its labels while the 9 other folds, with the visible national labels, are used to train the classifier. The classifier is used to predict the labels of airports in the testing fold. The result is the average of 10 runs, each with a different fold to be the testing fold. We measure the accuracy of the classification as the ratio between the number of airports that were correctly classified to the number of airports that were tested. We repeated the experiment 10 times and report the average results.

Figure 9 shows the accuracy of classification with different settings of V2V vector dimensions. For each selection of k - the number of neighbor that involves to the votes, the accuracy initially increases when we increase the number of dimensions. That is because the embedded vector with a low dimensional setting, 10 - 30 dimensions, cannot well capture the nature of the data. e have the best accuracy of 90% which the setting of 50 dimensions and $k = 3$. When we continue to increase the dimension of the embedded vector, the accuracy decreases. That is the overfiting phenomenon. The more complicated model, with higher dimension, requires more learning data to be trained properly. Meanwhile, we trained the V2V, with different settings of dimensions, in the same set of random walk paths. That leads to the overfitting in the high-dimensional models. Consequently, the accuracy decreases when we increase the dimension further. This dimensional-sensitivity shows how the number of dimensions of the vector embeddings affect the prediction's accuracy. While that is data-driven, in our experiment, the models with 40 - 70 dimensions give us the good results.

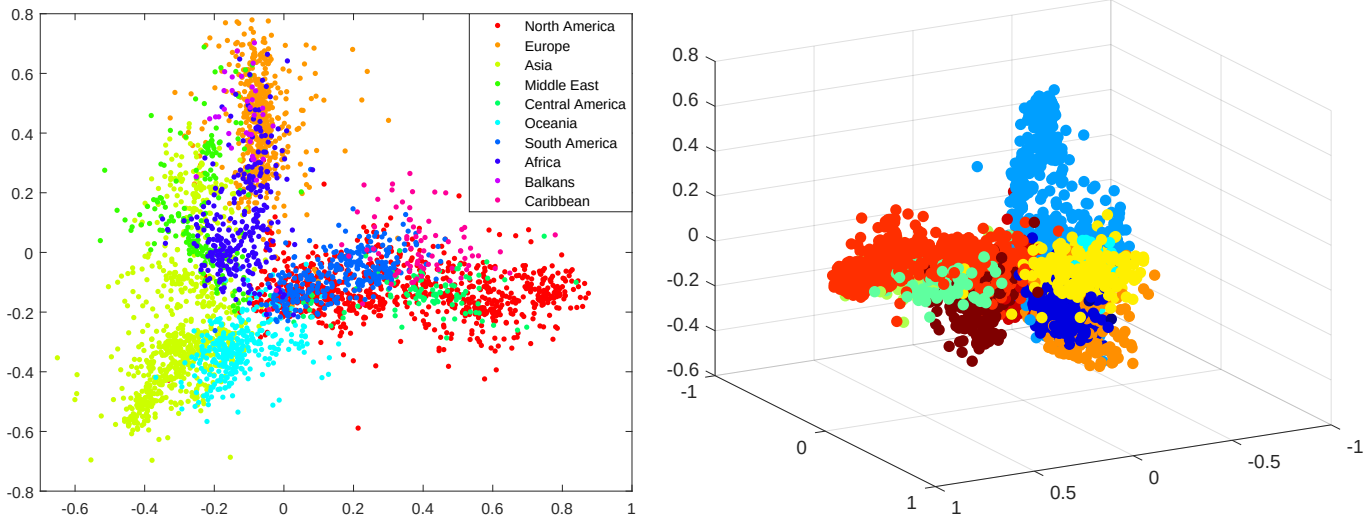
In using k-NN algorithm, the number k - the number of neighbors involving to the vote - is an essential parameter. Figure 10 show how the parameter k affects the classification's accuracy. For most of the settings of dimension, we have the best accuracy with $k = 3$, which is when we assign the country of a hidden airport by the majority vote of its 3 nearest neighbors, we have the best accuracy prediction.

In general, these high accuracy predictions, 85% to 90%, show that the V2V vector well captures the nature of the data, and enables prediction of vertex labels for this dataset.

VI. Related Work

Vectorization of graph-based data has been studied by multiple prior works. Tang *et al.* [32], [33] focus on social network data, and propose a latent representation for creating *social dimensions* of data. Starting from the social network as a graph, where vertices represent people and edges represent friendships, the authors use metadata of the relationships to learn features that help assign affiliations to vertices. This work is specific for social networks and uses metadata as important component to derive the representation.

Recent works [7], [8], [9], [10], [11], [34] follow an approach broadly similar to our work, by using paths in



(a) PCA 2D visualization.

(b) PCA 3D visualization.

Fig. 8: PCA Visualizations of the V2V vector (dimension = 50) embeddings of the OpenFlights dataset. The vector space is projected into top two and top three principal components to create the 2D and 3D display, respectively. Airports are colored by the continent for visualize purpose.

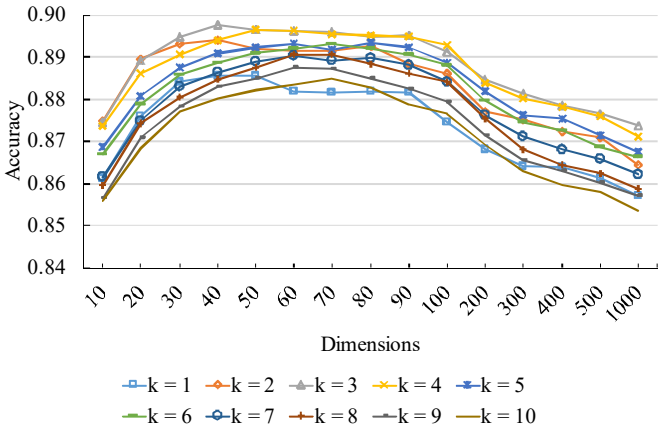


Fig. 9: Accuracy of predicting the country of airports as a function of the number of dimensions of embedded vectors. k is number of neighbors involving to the vote.

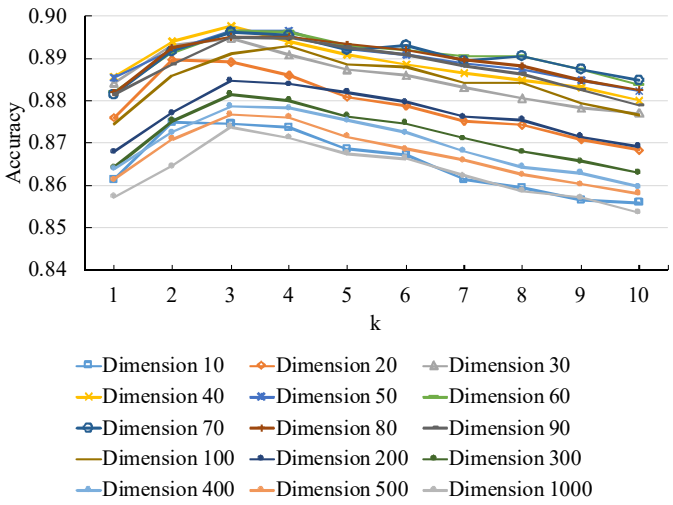


Fig. 10: Accuracy of predicting the country of airports as a function of k – number of neighbors involving to the vote.

a graph to construct a vector representation for vertices. Scarselli *et al.* [7] proposed using a recursive neural networks model, trained from random walk sequences, to map a graph and one of its vertices to a vector of reals. Others [8], [9], [10] have used SkipGram [5], a single hidden layer feedforward neural network, to learn vector representations of vertices from their *usage contexts*. The *usage context* are defined in different ways. Perozzi *et al.* [8] proposed using truncated random walks; Tang *et al.* [9] used paths based on breadth first search; while Grover *et al.* [10] combined breadth first search and depth first search strategies into a random walk process.

Our work V2V uses the Continuous Bag of Word (CBOW) [6] neural network model to learn the vector embeddings for vertices. We train our model from the paths that are constructed using random walks that can be specialized according to graph properties, such as *e.g.,* edge direction, edge and vertex weights, and edge timestamps. Further, in existing works, embedded vectors have been applied mostly in classification. We showed that the V2V representation is useful in a broad range of applications that include both machine learning oriented applications, such as classification, clustering, feature prediction, *etc.*, as well as graph oriented applications, such as community detection. The comparison with direct graph algorithms shows that V2V provides interest-

ing trade-offs between runtime and accuracy, when compared with graph-based algorithms.

VII. Conclusion

We introduced V2V, an approach to represent vertices in the graph by vector embeddings, which are learnt from the contexts in which the vertices appear in constrained random walks through the graph. The vector embedding of different vertices captures many aspects of the global structure of the graph, which are useful in detecting communities, in predicting roles (labels) of vertices, and in predicting relationships between pairs of vertices. Our experiments indicate that methods based on vector embeddings provide useful tradeoffs when compared with direct graph-based methods. For instance, in community detection, their accuracy was high but worse that of direct graph-based algorithms. On the other hand, they ran much faster than direct graph-based algorithms.

There are many improvements possible and to be investigated, in addition to open questions. One is a principled manner of selecting the various parameters for representation learning – these should be chosen keeping in mind the time complexity of learning as well as their accuracy. Another is to find characterizations of the types of problems that can be solved using such embeddings, and the types of problems they are not good for. Another direction is experiments on larger scale networks, and on graphs with missing or incorrect data.

References

[1] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[2] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner, "On modularity clustering," *IEEE transactions on knowledge and data engineering*, vol. 20, no. 2, pp. 172–188, 2008.

[3] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, no. 6, p. 066111, 2004.

[4] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[5] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013 (NIPS'13).*, 2013, pp. 3111–3119.

[6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[7] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[8] 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*, ser. KDD '14. New York, NY, USA: ACM, 2014, pp. 701–710. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623732

[9] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077. [Online]. Available: https://doi.org/10.1145/2736277.2741093

[10] 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*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 855–864. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939754

[11] N. K. Ahmed, R. Rossi, J. Boaz Lee, X. Kong, T. L. Willke, R. Zhou, and H. Eldardiry, "Learning Role-based Graph Embeddings," *ArXiv e-prints*, Feb. 2018.

[12] S. Sobolevsky, R. Campari, A. Belyi, and C. Ratti, "General optimization technique for high-quality community detection in complex networks," *Phys. Rev. E*, vol. 90, p. 012811, Jul 2014. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.90.012811

[13] R. A. Rossi and N. K. Ahmed, "Role discovery in networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 4, pp. 1112–1131, April 2015.

[14] P. Pons and M. Latapy, "Computing communities in large networks using random walks," in *Computer and Information Sciences - ISCIS 2005*, p. Yolum, T. Güngör, F. Gürgen, and C. Özturan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 284–293.

[15] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Berkeley, Calif.: University of California Press, 1967, pp. 281–297. [Online]. Available: http://projecteuclid.org/euclid.bsmsp/1200512992

[16] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2007, pp. 1027–1035.

[17] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *PVLDB*, vol. 5, no. 7, pp. 622–633, 2012.

[18] I. Jolliffe, *Principal Component Analysis*. USA: Springer-Verlag, 1986.

[19] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[20] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*. Cambridge university press, 1994, vol. 8.

[21] J. Scott, *Social network analysis*. Sage, 2017.

[22] M. E. Newman, "The structure of scientific collaboration networks," *Proceedings of the national academy of sciences*, vol. 98, no. 2, pp. 404–409, 2001.

[23] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *ACM SIGCOMM computer communication review*, vol. 29, no. 4. ACM, 1999, pp. 251–262.

[24] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph structure in the web," in *Proceedings of the 9th International World Wide Web Conference on Computer Networks : The International Journal of Computer and Telecommunications Netowrking*. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 2000, pp. 309–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=347319.346290

[25] S. P. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, 1982.

[26] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian, "Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software," *PloS one*, vol. 9, no. 6, p. e98679, 2014.

[27] J. Leskovec and R. Sosič, "Snap: A general-purpose network analysis and graph-mining library," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 1, p. 1, 2016.

[28] I. Jolliffe, *Principal Component Analysis*. Springer, 2002.

[29] G. Saporta and N. Niang, "Principal component analysis: application to statistical process control," *Data analysis*, pp. 1–23, 2009.

[30] http://openflights.org.

[31] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[32] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 817–826.

[33] ——, "Leveraging social media networks for classification," *Data Mining and Knowledge Discovery*, vol. 23, no. 3, pp. 447–478, 2011.

[34] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *CoRR*, vol. abs/1709.05584, 2017. [Online]. Available: http://arxiv.org/abs/1709.05584