Graph Embedding: A Methodological Survey

Joseph R. Barr
MI Lab, Acronis SCS,
Scottsdale, Arizona, USA
barr.jr@gmail.com

Peter Shaw
Oujiang Laboritory,
Wenzhou, Zhejiang, China
petershaw@ojlab.ac.cn

Faisal N. Abu-Khzam
Lebanese American University
Beirut, Lebanon
faisal.abukhzam@lau.edu.lb

Tyler Thatcher
MI Lab, Acronis SCS,
Scottsdale, Arizona, USA
tyler.thatcher@acronisscs.com

Toby Dylan Hocking
Northern Arizona University
Flagstaff, Arizona, United States of America
toby.hocking@nau.edu

***Abstract*—Embedding a high dimensional combinatorial object like tokens in text or nodes in graphs into a lower dimensional Euclidean space is a form of (lossy) data compression. We will demonstrate a class of procedures to embed vertices of a (connected) graph into a low-dimensional Euclidean space. We explore two kinds of embedding, one *node2vec*, similar to *word2vec*, which deploys a shallow network and a recurrent network which remembers past moves and takes [sic] spatial correlations into an account. We also explore the extent in which graph embedding preserves information and the practicality of using the information stored in a compressed form to discern meaningful patterns. With growth in their popularity, we too make an extensive use of the neural networks computational frameworks; we propose the usage of various neural network architectures to implement an encoder-decoder scheme to learn 'hidden' features. Since training a network requires data, we describe various sampling techniques including novel methods to sample from a graph; one using a *vertex cover* and another is an *Eulerian tour* of a (possibly) modified graph.**

***Index Terms*—Graph Embedding ; Node2vec ; Encode-Decoder ; Sampling ; Data Compression ; Vertex Cover ; Cluster Editing**

I. Introduction

We assume the reader is familiar with graphs. In this context graphs are undirected and possess no loops. In many ways graphs are peculiar in that they are simultaneously simple and complex. In fact, many simple-to-state problems are known to evade an easy solution. A demonstration of this fact is the notorious *Four-Color Problem* which after a century of struggles and failed 'proofs', a (genuine) proof was discovered (Appel & Haken in 1977 [1].) It is part of the folklore that the theory of *satisfiability*, (Knuth [2],) is often described in terms of graph-theoretic decision/search problems [3]. The NP-hard problems, those easily verifiable, but hard to solve are often stated in graph-theoretic terms. As of yet (the Summer of 2022) it is not known whether a solution of any representative of the NP-complete class can be found in polynomial time. Donald Knuth expressed his frustration [2]; he says *"Section 7.1.1 discussed the embarrassing fact that nobody has ever been able to come up with an efficient algorithm to solve the general satisfiability problem..."* Arguably, the domain of graphs is an endless source of NP-hard problem with hundreds belonging to the class NP-complete. Graph embedding strives to translate aspects of graph attributes into analytical problems, i.e., analysis in a finite dimensional Euclidean spaces.

A. Graph Embedding Literature

There are a few previous surveys of algorithms for graph embedding [4,5], some with emphasis on link prediction [6], biological applications [7], and convolution [8]. Other approaches for detecting change-points in time series of graphs include non-embedding methods that attempt to optimize criteria such as Minimum Description Length [9].

Recently, the use of random walks became popular. In particular the use of random walks for computing GNN [10].

II. Lossless and Lossy Compression

Classical representations of a graphs as a matrix are

1) *Adjacency matrix* $A = (a_{uv})$ where

$$a_{uv} = \begin{cases} 1 & \text{if vertices } u \text{ and } v \text{ are connected} \\ 0 & \text{else.} \end{cases}$$

2) *Incidence matrix* $B = (b_{ve})$ where

$$b_{ve} = \begin{cases} 1 & \text{if } v \text{ is an endpoint of } e \\ 0 & \text{else.} \end{cases}$$

Either of those matrix representations are *lossless* as they preserve all the information. However, both are grossly inefficient because both matrices are generally sparse. If one is tolerant of information loss, then one might consider an approach which represents a graph G as a 'dense' matrix of lower order. Accordingly, an *embedding* is a mapping $\mathcal{G}_n \to \mathbb{R}^{n\times d}$ which maps a graph of order n into an $n \times d$ matrix with $d \ll n$: a lossy representation of order $O(n)$ rather than $O(n^2)$.

An *autoencoder* utilizes a neural networks, an *encoder-decoder* architecture to produce the embedding. In recent years, various autoencoder techniques have become a staple of the study of unstructured data. For example, images, Hinton, et al. [11], text, Mikolos, et al., [12], source code, Alon, et al.,

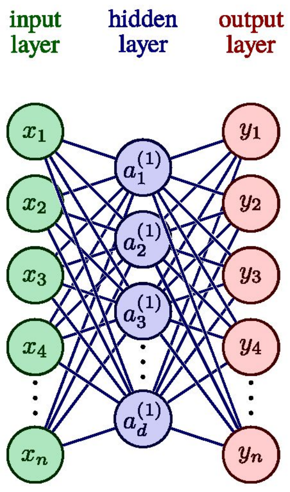

Figure 1: Node2vec feed-forward neural network architecture.

[13], and social media Barr, et al., [14]. The dimension $n \times d$ of the vector space $\mathbb{R}^{n\times d}$ is the dimension of the embedding. It seems reasonable to aspire for an efficient embedding, to select d as small as possible while maintaining as much structural information about the underline graph as possible. In the spirit of The Moody Blues' 1970 *Question of Balance*, graph embedding strives to gain maximal efficiency using as few parameters as possible while minimizing information loss. In that sense, graph embedding falls into the category of heuristics. We use analytical procedures to represent a graph as a dense matrix of order $O(n)$ using an *encoder-decoder* neural network architectures.

Invariably, the extent to which graph embedding preserves information must be tested against some ground truth. We expect that matrix embedding of graphs to possess a certain measure of continuity: that if two graphs G and G' are similar (in some sense,) then so are their embedding, and conversely. We would expect that a matrix resulting from the embedding of a dense graph, say K_n[1], is fundamentally different from the matrix resulting from a sparse graph, say a tree. In that sense, graph embedding heuristics are no different from any other data models where a model's performance is tested against a truth, often putative truth.

To fix notation, let $G = (V, E)$ be a graph with $|V| = n$ the *order* of G and a labelling of the vertices, i.e., a mapping $V \to \{1, 2, \cdots, n\}$; so we identify $v \in V$ with its label k where $v \to k$. Fix a labeling of the vertices of the graph.

A ONE-HOT ENCODING of V is the mapping $\phi : V \to \{0,1\}^n$ with $\phi(v_1) = (1, 0, 0, ..., 0)$, $\phi(v_2) = (0, 1, 0, 0, ..., 0)$ and in general $\phi(v_k) = (0, 0, ..., 0, 1, 0, ..., 0)$ with 1 in the kth spot and 0 elsewhere.

We should mention that one-hot encoding of G as points in $\mathbb{R}^n$ do not suppose to be a meaningful representation of the graph: any two one-hot vectors are orthogonal, and the distance between any two is $\sqrt{2}$.

III. NODE2VEC

As the much-used meme suggests, `node2vec` is a class of algorithms resulting in a vector representation of the nodes of a graph. Keeping faithful to tradition and relying on the seminal work in [12] and [15], we represent a *skip-node* procedure. We consider two versions: a shallow feed-forward network *S-skip-node*, as well as variations thereof *R-skip-node* which implement recurrent networks.

In a shallow feed-forward architecture, the network estimates two matrices $U \in \mathbb{R}^{d\times n}$ and $V \in \mathbb{R}^{n\times d}$. In a trained network, the columns of U are the embedding: the first column of U is the embedding of v_1, the second column of v_2, etc.

In a recurrent neural network (RNN) scheme, or more precisely *long short-term memory (LSTM)* network, three matrices are estimated, $U \in \mathbb{R}^{d\times n}$ from input to the hidden layer, $V \in \mathbb{R}^{n\times d}$ from hidden to output, and $W \in \mathbb{R}^{d\times d}$ from hidden to itself. The network estimates U, V, and W. Just like in the shallow feed-forward network framework, the columns of U are the embedding of the vertices of the graph.

SKIP-NODE predicts the neighbors of a *pivot node* v. The skip-node produces a probability vector $(p_1, p_2, \ldots, p_n)$ with p_j is the probability that vertex v_j is adjacent to the *pivot node* v.

Both models involve a hyper-parameter d, $d \ll n$, which is the size of the hidden layer; optimizing d requires empirical evidence.

As noted above, the *skip-node* procedure predicts all vertices adjacent to a pivot vertex. For vertex v_k, as pivot, the training pair associated with v_k is the one-hot representation of v_k, $x = (0, \ldots, 0, 1, 0, \ldots, 0)$, and the sum of all one-hot encoding of the neighbors of v_k, i.e., $y = v_{k(1)} + \cdots + v_{k(m_k)}$ where $\deg(v_k) = m_k$ and $v_{k(j)}$ are adjacent to v_k.

[1]K_n is the complete graph on n vertices.

In other words y is a (0,1)-vector of weight m_k, with 1 in positions corresponding to labels of neighbors of v_k.

For example, say v_1 is the pivot vertex and suppose for definiteness, that the neighbors of v_1 are v_2, v_3, v_4 and v_5, then for the pivot v_1 we associate a training pair

$$\Big((1,0,0,0,0,\ldots,0)\ ,\ (0,1,1,1,1,0,0,\ldots,0)\Big)$$

where v_1 is encoded as $(1,0,0,0,0,\ldots,0)$.

In general, if the neighbors of v are $u, w, x, y, \ldots$ then the training pairs are of the form

$$\big(v\ ,\ u + w + x + y + \ldots\big).$$

A. Embedding with Recurrent Neural Networks (RNN)

We may replicate skip-node using a recurrent neural network rather than a shallow feed-forward one. In practice, to help ensure convergence, a *long short-term memory network (LSTM network)* is used. RNN architecture consists of n dimensional inputs (n is the order of the graph), n dimensional output, d hidden nodes and three matrices of parameters. $U \in \mathbb{R}^{d\times n}$ from input-to-hidden, $V \in \mathbb{R}^{n\times d}$ from hidden-to-output, and $W \in \mathbb{R}^{d\times d}$ from hidden-to-hidden (See Fig 2). The goal is to estimate the parameter matrices U, V, and W. The embedding of vertex v_k the kth column of U.

IV. Sampling vertices from a graph

A set of training pairs $\{(x_i, y_i), i = 1, 2, \ldots, k\}$, is an indispensable ingredient of machine learning. We propose several procedures to extract data.

A. Simulating training pairs with a random walk

A *walk* in a graph G is a sequence of vertices $v^1, v^2, v^3, ..., v_m$ so that for each $i = 1, 2, ..., m-1$, the pairs $v^i v^{i+1}$ are edges of G. A *random walk* (RW) on a graph G is a [sic] random traversal of the vertices along the edges of the graph (See Fig. 3). A random walk is initialized at a 'random' vertex $v = v^1$ (selected uniformly) and proceed to move randomly to a neighboring vertex. The probability of a move is uniform; if, say, v has 4 neighbors, w, u, x, y, then the move to either has a probability $1/4$. The random walk continues for a specified, predetermined number of moves. The number of moves q is a parameter controlled by the modeler. The random walk generates the training pairs, and so the sample size equals the number of moves. A realization of a random walk of size q is $v^1, v^2, v^3, \ldots, v^q$ where $v^k v^{k+1}$ is an edge, but the v^j may not be distinct; in fact, we design the walk to be sufficiently long to ensure vertices of degree greater than 1 are likely to be revisited multiple times.

A sample generated with a random walk is rather similar to bootstrapping a sample from data. Bootstrapping is a re-sampling method introduced by B. Efron more than 40 years ago [16].

We walk the edges of the graph, giving all the neighbors of a vertex equal probability. Each (random) move gives rise to a training pair, as described earlier. The number of simulated moves must be larger than the number of vertices to ensure the entire graph has a good chance of being represented in the sample. A consequence of the procedure is that a training pair may appear multiple times in the sample or perhaps not appear at all.

B. Simulating training pairs using Vertex Covers

A vertex cover of a graph $G = (V, E)$ is nothing but a set of vertices $C \subset V$ such that every edge of G has at least one endpoint in C. As such, the adjacency lists of all the elements of C cover all the edges, i.e., it captures the graph information in its entirety. It follows that finding a small vertex cover, if any, allows us to reduce the overall graph representation. A corresponding embedding via our skip-node approach (described above) can save valuable space while achieving a lossy compression/embedding.

Despite the NP-hardness of the Vertex Cover problem, recent algorithms based on fixed-parameter tractability methods proved to be highly efficient in practice [17], and methods for using high performance computing platforms, including GPUs, can solve some of the most recalcitrant instances in seconds [18]–[23]. Therefore, depending on the computed size of the vertex cover, this method could significantly improve performance. However, in general, it can be noted that it will reduce the size of the edge list by a factor of two.

C. Simulating training pairs with semi-Eulerian Walks

The purpose of a random walk is to obtain a vector representation that covers as many vertices as possible in as few steps as possible, thus capturing information (about the graph.)

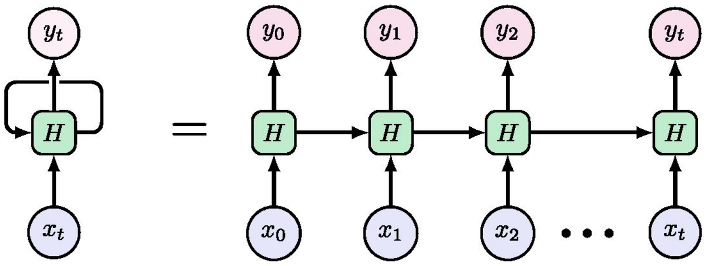

Figure 2: Node2vec RNN architecture.

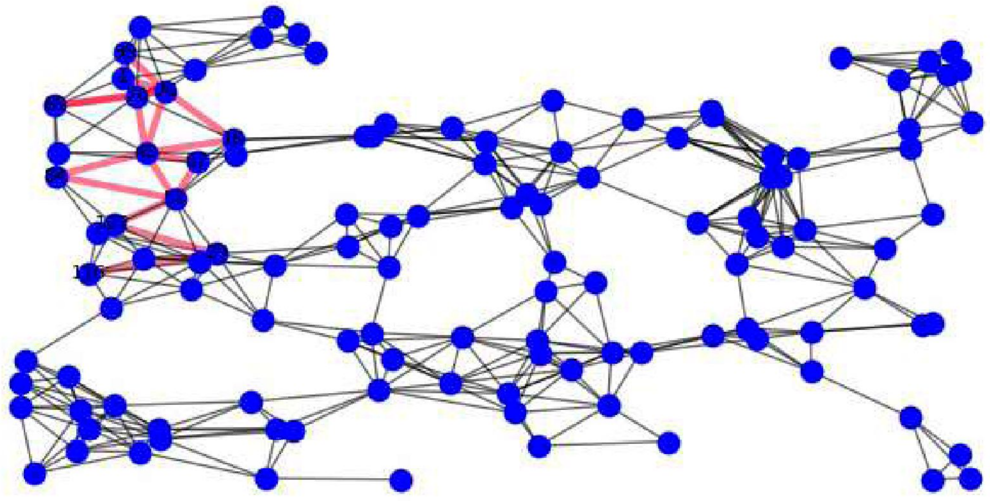

Figure 3: A (short) random walk (in red).

An alternative approach to this (partial) covering is to try to capture all the information by a walk that simply visits all the edges. Of course, one cannot avoid edge repetition unless the graph is Eulerian.

Our Semi-Eulerian Walk algorithm consists of (i) adding virtual edges between each pair of non-adjacent (but "close enough") odd-degree vertices to transform the input graph into an Eulerian graph and, (ii) visiting all the edges via an Eulerian walk of the (new) graph. The output "marks" the virtual edges via a special unique code. Such a mark marks a jump in the resulting vector.

The number of added virtual edges is at most $n/2$. Thus our resulting walk has a length at most $m + 3n/2$ where m, is the number of edges in the input graph.

V. Improving the random work using Cluster Editing

In computing a random walk that covers the graph, it is beneficial to start the random walks from a number of well separated locations. These starting points can be chosen randomly, however ideally the are spread over the clusters of the graph. The Cluster Editing problem provides a near ideal way to compute these subgraphs. Cluster Editing partitions the vertices of a graph $G = (V, E)$ such that after adding or removing a minimum number of edges this partitions $V = C_1, \dots, C_p$ into a union of disjoint cliques.

Computing a random walk using Cluster Editing:

1) Compute the cluster graph $C_1, \dots, C_p$
2) Compute a random walk starting from a vertex in each clique in the cluster graph with length of size $|C_i| + c$ i.e. $N[s_i] + c$ in the cluster graph.
3) *If for some reason the edges of C_i are not well covered a second vertex in C_i can be covered.*

It is worth noting that, while computing an optimum cluster graph is NP-hard, practical FPT methods proved to be efficient in practice [14, 24]–[26]. Moreover, the requirement to edit the graph into a disjoint union of cliques can sometimes be too restrictive. Other editing variants that relax this restriction can play an effective role in this context and might be worth exploring [27]–[29].

VI. Learning framework

We expand on the learning frameworks.

The auto-encoder architecture is a shallow feed-forward neural network. The embedding is a hidden layer with d nodes.

In a complete analogy of skip-gram, the Skip-node training pairs are $(v_k, u_1 + \cdots + u_m)$ where m is the degree of v and a vertex v_k and u_j are one-hot encoding of corresponding vertices.

The goal is to predict the neighbor's pivot vertex v_k. The prediction is in the form of a vector $(p_1, p_2, \dots, p_n)$ where $p_k \geq 0$ and $\sum_j p_j = 1$.

A. *Shallow feed-forward architecture*

This simple architecture has an input of size n, a single hidden layer consisting of d nodes, and an output layer of n nodes. U is a $d \times n$ matrix of weights associated with the input-to-hidden layer, and the matrix associated with the hidden-to-output is an $n \times d$ matrix V. For example, see Fig. 1.
The input-hidden activation is linear, i.e., Ux (matrix multiplication,) and the hidden-output activation is `softmax`. In other words, for input vector x_v, a (0,1)-vector of weight-1 we have output $\texttt{softmax}(VUx_v)$. To unravel, we have $Ux_v \in \mathbb{R}^d$

and $VUx_v \in \mathbb{R}^n$, say $VUx = (r_1, r_2, ..., r_n)^T$. The output therefore is

$$\texttt{softmax}(VUx_v) = \Big(\frac{\exp(r_1)}{\sum_j \exp(r_j)}, \frac{\exp(r_2)}{\sum_j \exp(r_j)}, \dots, \frac{\exp(r_n)}{\sum_j \exp(r_j)}\Big)^T. \quad (1)$$

The standard cost function for this problem is CROSS-ENTROPY $\rho(\eta, \mu) = -\sum_j \eta_j \log(\mu_j)$ and for skip-node model, cross-entropy is $\rho(v, \texttt{softmax}(VUx_v))$. To amplify, v is a one-hot pivot vector, with a corresponding vector the sum of the one-hot representations of the neighbors of v (or the (0,1) vector with 1's in positions corresponding to neighbors of v.)

B. Bag-of-nodes

In a complete analogy to the foregoing, the *bag-of-nodes* procedure predicts the nodes which are jointly adjacent to any subset of the nodes. Say, a bag of k nodes, $v_1, v_2, \dots, v_k$, the bag-of-nodes procedure produces a probability vector $(p_1, p_2, \dots, p_n)$ where p_j is the probability that node v_j is adjacent to $v_1, v_2, \dots, v_k$. Training pairs consist of (x, y) where x is the sum of the one-hot vertices of the bag and y is the sum of the one-hot vertices adjacent to the bag.

It should be clear that the data complexity of bag-of-nodes is far greater than that of skip-node. We believe that with the exception of graphs with density $O(n^2)$, (density = number of edges divided by the number of nodes), bag-of-nodes is not a viable approach to embed a graph.

C. Recurrent network

We now describe a recurrent network approach to node2vec. We consider the setting mentioned above; graph G of order n, with vertex set $\{v_1, v_2, \dots, v_n\}$, each one-hot encoded and a set of training pairs. To amplify any spatial correlation, in a recurrent neural network (RNN), training pairs are fed one-by-one as they appear in the data.

The simplest form of RNN involves three matrices $U \in \mathbb{R}^{d \times n}$, $V \in \mathbb{R}^{n \times d}$, and $W \in \mathbb{R}^{d \times d}$. As before, the embedding size d is a model's hyperparameter, representing the number of nodes in the hidden layer. Information flows from input to hidden as well as laterally, from hidden to hidden.

To fix notation, we may label training pairs with a time stamp t, $t = 1, 2, 3, \dots$, and so training pairs are labeled $(x_t, y_t), t = 1, 2, \dots$ where for skip-node, y_t is the sum of the one-hot nodes neighboring to x_t.

Training is defined recursively:
BASE CASE. A training pair (x_1, y_1) is fetched; the hidden value $h_1 = W_{hh}x_1$ the network outputs

$$\texttt{softmax}(W_{hy}h_1) = \texttt{softmax}(W_{hy}W_{xh}x_1).$$

RECURSIVE STEP. Suppose $(x_1, y_1), ..., (x_{t-1}, y_{t-1})$ were processed and (x_t, y_t) is fetched. The output is

$$\texttt{softmax}(W_{hh}h_{t-1} + W_{hy}W_{xh}x_{t-1}).$$

It's important to note that RNNs tend to suffer from the *vanishing* or *exploding* gradient problem, so the RNN model framework generally fails to converge. A better and more robust approach is the *long short-term memory (LSTM)* networks. To specify a model framework, one needs to explicitly describe all the computational nodes and the relations between those. This includes specifying data flow, activation functions, and any 'special purpose' gates required to enhance network robustness which guarantees convergence.

VII. GRAPH MATRIX

Consider an embedding of vertices V of G into $\mathbb{R}^d$, $2 \leq d \ll |V| = n$. With the ordering of $V = \{v_1, v_2, \dots, v_n\}$ we construct a matrix $M_G \in \mathbb{R}^{n \times q}$ where column j of M_G is the embedding of vertex v_j.

$$M_G = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & . & . & . & m_{1,n} \\ m_{2,1} & m_{2,2} & m_{2,3} & . & . & . & m_{2,n} \\ m_{3,1} & m_{3,2} & m_{3,3} & . & . & . & m_{3,n} \\ . & . & . & . & . & . & \\ . & . & . & . & . & . & \\ . & . & . & . & . & . & \\ m_{d,1} & m_{d,1} & m_{d,3} & . & . & . & m_{d,n} \end{bmatrix}$$

In fact, M_G is a "fat and short" matrix; specifically, the idea of embedding to compress information means that d much smaller than n. One would expect that an effective $d = d(n)$ lies somewhere between $\log(n)$ and $\sqrt{n}$ and a good choice function of $d(n)$ is selected based on empirics.

A. Graph vector

Perhaps the goal of converting M_G into a 'low-dimensional' vector without losing information is not attainable, but converting it into a vector while losing as little information as possible may be. A plausible approach in projecting the columns of M_G into a subspace of the column space of M_G. As a matter of expediency, since a graph G is fixed, there's no need to signify it, and we write M for M_G. Principle components is one possible approach. Consider the matrix MM^T and an orthogonal $n \times n$ matrix U such that $U^T(X^TX)U = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_n)$ with $\lambda_1 \geq \lambda_2 \geq \dots \lambda_n \geq 0$. Write $u^1, u^2, ..., u^n \in \mathbb{R}^n$ the columns of U. Fix some $\alpha \geq 0.9$, say, and let

$$k = \min\Big\{j \leq n : \frac{\lambda_1 + \dots + \lambda_j}{\lambda_1 + \dots + \lambda_n} \geq \alpha\Big\}.$$

Define the k principal components $Mu^1, ..., Mu^k \in \mathbb{R}^q$ and the $n \times k$-matrix consisting of columns $Mu^1, ..., Mu^k$.

B. An illustration.

If G has 10,000 vertices and we embed G into $\mathbb{R}^8$ then we map G into a matrix M of order $8 \times 10,000$. Say three eigenvalues account for 91 percent of the variability, then M is further reduced to a $10,000 \times 3$ matrix, say M'. In other words, we've reduced the representation of G from $10,000^2 = 10^8$ to $30,000 = 3\times10^4$. Still, it's very high dimension. Is it possible to do better? To embed a graph with 10,000 nodes into a 20-dimensional vector? Perhaps, but a different approach must be invented.

C. Embedding of subgraphs

The idea is this: Consider a graph decomposition $V(G) = V(G_1) \cup V(G_2) \cup ... \cup V(G_q)$ where G_j is a dense subgraph and $V(G_j)$ is the vertex set of G_j. Consider an embedding of G into $M_G \in \mathbb{R}^{d\times n}$ as described above. Associate a vector x_{G_j} with a component G_j by averaging the embedded vertex vectors of $V(G_j)$. This results in q vectors, each of dimensionality d.

D. Lowering the complexity: partitioning into dense subgraphs

We repeat & expand the idea in the introductory remarks. Suppose G is partitioned into dense subgraph, this means V is partitioned into a disjoint collection of vertex set $V_1, V_2, \dots, V_q$, $q \geq 1$, with the induced graphs $G[V_j]$, subgraphs of G on a vertex set V_j dense, say clique.

Consider node2vec embedding of G into say, $d\times n$ matrix $M = M_G$. Now M can be further reduced by taking the average of the embedded vectors of group V_1, $\frac{1}{|V_1|}\sum_1^{k_1} x(v_j)$. For every $j = 1, 2, \dots, q$, if $|V_j| = l_j$, consider the average $a_j = \frac{1}{|l_j|}\sum_1^{l_j} x(l_j) \in \mathbb{R}^d$.

This results in q vectors $a_1, a_2, ..., a_q$, each of order d. The graph G is then encoded by a $d \times q$ matrix. Since, in general, $q < n$, this results in substantially more economical compression than node2vec.

This is in fact an embedding of the QUOTIENT GRAPH modulo $\sim$ where $u \sim v$ if u and v belong to the same part (clique.)

VIII. Things to investigate

Our aim is to investigate the embedding; specifically, to investigate whether the proposed embedding sufficiently preserves the information of the underline graph.

A few things we propose to investigate:

1) How graphical distance between any two vertices translate into corresponding Euclidean distance between their embedding?
2) Invariance under relabeling; whether changing the labeling changes the embedding.
3) Feasibility to reconstructing a graph from its embedding.
4) How small perturbation affects embedding?
5) Optimizing 'd', the dimensionality of the embedding.
6) Refining encoder-decoder architecture.
7) Optimizing the generation of training pairs.
8) How embedding can be integrated in making distinctions and/or answering a decision (yes/no) problems?

IX. Conclusion

We have demonstrated how vertices of graph are mapped to a Euclidean space $\mathbb{R}^d$ with $2 \leq d \ll n = |V(G)|$ resulting in a $d \times n$ matrix that represents the graph, and we proposed to study the workflow of embedding correlation clustering and aggregating vertex vectors by clusters which result in a $d \times k$ matrix (k = number of clusters), and examine how graphical attributes are preserved.

A related approach, for potential future work, would the use of graph mapping via dimensionality reduction, which was proposed in [30, 31] for graph compression.

X. Application

We propose using the data sets mentioned in [9], Ohio crime incidents, https://data.cincinnati-oh.gov/safety/PDI-Police-Data-Initiative-Crime-Incidents/k59e-2pvf, and

California temperature, https://prism.oregonstate.edu/explorer/map.php, which uses the PRISM model [32].

There are several items to address but all involve a sequence of graphs (networks) which represent temporal configurations. The idea is to use economical graph embedding to study the following.

1) *Investigate a sequential patterns "time series of graphs".*
2) *Identify outliers in the sequence.*
3) *Investigate trending.*
4) *Investigate relation between graph parameters (say, spectra, connectivity, etc.) and embedding.*

XI. Acknowledgements

Joseph Barr was supported by Acronis SCS and was greatly assisted by Acronis (Singapore.) Special thanks to Ms. Katerina Archangorodskaja of Acronis for her valuable suggestions and encouragements.

Peter Shaw was supported in part by the Oujiang Lab China's (Zhejiang Lab for Regenerative Medicine, Vision and Brain Health) startup fund.

References

[1] K. Appel and W. Haken, "Every planar map is four colorable," *Illinois Journal of Mathematics*, vol. 3, p. 429–490, 1977.
[2] D. Knuth, *Art of Computer Programming, The Satisfiability, Volume 4, Fascicle 6*. Addison-Wesley, 2015.
[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W H Freeman and Co., 1979.
[4] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *arXiv preprint arXiv:1705.02801*, 2017.
[5] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.
[6] M. Wang, L. Qiu, and X. Wang, "A survey on knowledge graph embeddings for link prediction," *Symmetry*, vol. 13, no. 3, p. 485, 2021.
[7] S. K. Mohamed, A. Nounu, and V. Nováček, "Biological applications of knowledge graph embedding models," *Briefings in bioinformatics*, vol. 22, no. 2, pp. 1679–1693, 2021.
[8] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, pp. 1–23, 2019.

[9] C. Xu and T. C. M. Lee, “Change point detection and node clustering for time series of graphs,” *IEEE Transactions on Signal Processing*, vol. 70, pp. 3165–3180, 2022.

[10] J. Callut, K. Françoisse, M. Saerens, and P. Dupont, “Classification in graphs using discriminative random walks,” in *International Workshop on Mining and Learning with Graphs*, vol. 7. Citeseer, 2008.

[11] G. Hinton, A. Krizhevsky, and S. Wang, “Transforming auto-encoders,” in *In International Conference on Artificial Neural Networks*. Springer, 2011, pp. 44–51.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *ArXiv*, 2013.

[13] ASTMiner, https://github.com/JetBrains-Research/astminer.

[14] J. R. Barr, P. Shaw, F. N. Abu-Khzam, and J. Chen, “Combinatorial text classification: the effect of multi-parameterized correlation clustering,” in *2019 First International Conference on Graph Computing (GC)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2019, pp. 29–36.

[15] M. L. Soutner D., “Application of lstm neural networks in language modelling.” *Lecture Notes in Computer Science*, vol. 8082, 2013.

[16] B. Efron, “Bootstrap methods: Another look at the jackknife,” *Ann. Statist*, vol. 7, no. 1, 1979.

[17] F. N. Abu-Khzam, M. A. Langston, A. E. Mouawad, and C. P. Nolan, “A hybrid graph representation for recursive backtracking algorithms,” in *Frontiers in Algorithmics, 4th International Workshop, FAW 2010, Wuhan, China, August 11-13, 2010. Proceedings*, ser. Lecture Notes in Computer Science, D. Lee, D. Z. Chen, and S. Ying, Eds., vol. 6213. Springer, 2010, pp. 136–147.

[18] F. N. Abu-Khzam, M. A. Langston, and P. Shanbhag, “Scalable parallel algorithms for difficult combinatorial problems: A case study in optimization,” in *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks, Innsbruck, Austria, February 17-19, 2004*, M. H. Hamza, Ed. IASTED/ACTA Press, 2004, pp. 649–654.

[19] F. N. Abu-Khzam, M. A. Langston, P. Shanbhag, and C. T. Symons, “Scalable parallel algorithms for FPT problems,” *Algorithmica*, vol. 45, no. 3, pp. 269–284, 2006. [Online]. Available: https://doi.org/10.1007/s00453-006-1214-1

[20] F. N. Abu-Khzam, K. Daudjee, A. E. Mouawad, and N. Nishimura, “On scalable parallel recursive backtracking,” *J. Parallel Distributed Comput.*, vol. 84, pp. 65–75, 2015. [Online]. Available: https://doi.org/10.1016/j.jpdc.2015.07.006

[21] T. Akiba and Y. Iwata, “Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover,” *Theoretical Computer Science*, vol. 609, pp. 211–225, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030439751500852X

[22] F. N. Abu-Khzam, D. Kim, M. Perry, K. Wang, and P. Shaw, “Accelerating vertex cover optimization on a gpu architecture,” in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 616–625.

[23] P. Yamout, K. Barada, A. Jaljuli, A. E. Mouawad, and I. E. Hajj, “Parallel vertex cover algorithms on gpus,” in *2022 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022, Lyon, France, May 30 - June 3, 2022*. IEEE, 2022, pp. 201–211. [Online]. Available: https://doi.org/10.1109/IPDPS53621.2022.00028

[24] F. N. Abu-Khzam, “On the complexity of multi-parameterized cluster editing,” *Journal of Discrete Algorithms*, vol. 45, pp. 26–34, 2017.

[25] P. Shaw, J. R. Barr, and F. N. Abu-Khzam, “Anomaly detection via correlation clustering,” in *16th IEEE International Conference on Semantic Computing, ICSC 2022, Laguna Hills, CA, USA, January 26-28, 2022*. IEEE, 2022, pp. 307–313. [Online]. Available: https://doi.org/10.1109/ICSC52841.2022.00057

[26] J. R. Barr, P. Shaw, F. N. Abu-Khzam, T. Thatcher, and S. Yu, “Vulnerability rating of source code with token embedding and combinatorial algorithms,” *Int. J. Semantic Comput.*, vol. 14, no. 4, pp. 501–516, 2020. [Online]. Available: https://doi.org/10.1142/S1793351X20500087

[27] F. N. Abu-Khzam, J. Egan, S. Gaspers, A. Shaw, and P. Shaw, “Cluster editing with vertex splitting,” in *Combinatorial Optimization - 5th International Symposium, ISCO 2018, Marrakesh, Morocco, April 11-13, 2018, Revised Selected Papers*, ser. Lecture Notes in Computer Science, J. Lee, G. Rinaldi, and A. R. Mahjoub, Eds., vol. 10856. Springer, 2018, pp. 1–13. [Online]. Available: https://doi.org/10.1007/978-3-319-96151-4_1

[28] F. N. Abu-Khzam, N. Makarem, and M. Shehab, “An improved fixed-parameter algorithm for 2-club cluster edge deletion,” *CoRR*, vol. abs/2107.01133, 2021. [Online]. Available: https://arxiv.org/abs/2107.01133

[29] F. N. Abu-Khzam, J. R. Barr, A. Fakhereldine, and P. Shaw, “A greedy heuristic for cluster editing with vertex splitting,” in *2021 4th International Conference on Artificial Intelligence for Industries (AI4I)*. IEEE, 2021, pp. 38–41.

[30] F. N. Abu-Khzam and R. H. Mouawi, “Concise fuzzy representation of big graphs: a dimensionality reduction approach,” *CoRR*, vol. abs/1803.03114, 2018. [Online]. Available: http://arxiv.org/abs/1803.03114

[31] F. N. Abu-Khzam, A. H. Ahmad, and R. H. Mouawi, “Concise fuzzy representation of big graphs: A dimensionality reduction approach,” in *Data Compression Conference, DCC 2020, Snowbird, UT, USA, March 24-27, 2020*, A. Bilgin, M. W. Marcellin, J. Serra-Sagristà, and J. A. Storer, Eds. IEEE, 2020, p. 356. [Online]. Available: https://doi.org/10.1109/DCC47342.2020.00056

[32] C. Daly, R. P. Neilson, and D. L. Phillips, “A statistical-topographic model for mapping climatological precipitation over mountainous terrain,” *Journal of Applied Meteorology and Climatology*, vol. 33, no. 2, pp. 140–158, 1994.