A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications

Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang

Abstract—Graph is an important data representation which appears in a wide diversity of real-world scenarios. Effective graph analytics provides users a deeper understanding of what is behind the data, and thus can benefit a lot of useful applications such as node classification, node recommendation, link prediction, etc. However, most graph analytics methods suffer the high computation and space cost. Graph embedding is an effective yet efficient way to solve the graph analytics problem. It converts the graph data into a low dimensional space in which the graph structural information and graph properties are maximumly preserved. In this survey, we conduct a comprehensive review of the literature in graph embedding. We first introduce the formal definition of graph embedding as well as the related concepts. After that, we propose two taxonomies of graph embedding which correspond to what challenges exist in different graph embedding problem settings and how the existing work address these challenges in their solutions. Finally, we summarize the applications that graph embedding enables and suggest four promising future research directions in terms of computation efficiency, problem settings, techniques and application scenarios.

Index Terms—Graph embedding, graph analytics, graph embedding survey, network embedding

1 Introduction

Graphs naturally exist in a wide diversity of real-world scenarios, e.g., social graph/diffusion graph in social media networks, citation graph in research areas, user interest graph in electronic commerce area, knowledge graph etc. Analysing these graphs provides insights of how to make good use of the information hidden in graphs, and thus has received significant attention in the last few decades. Effective graph analytics can benefit a lot of applications, such as node classification [1], node clustering [2], node retrieval/recommendation [3], link prediction [4], etc. For example, by analysing the graph constructed based on user interactions in a social network (e.g., retweet/comment/follow in Twitter), we can classify users, detect communities, recommend friends, and predict whether an interaction will happen between two users.

Although the problem of graph analytics is practical and essential, most graph analytics methods suffer the high computation and space cost. A lot of research efforts have been devoted to conducting the expensive graph analytics efficiently. Examples include the distributed graph data processing framework (e.g., GraphX [5], GraphLab [6]), new space-efficient graph storage which accelerate the I/O and computation cost [7], and so on.

In addition to above strategies, graph embedding provides another effective yet efficient way to solve the graph analytics problem. Specifically, graph embedding converts a graph into a low dimensional space in which the graph information is preserved. By representing a graph as a (or a set of) low dimensional vector(s), graph algorithms can then be computed efficiently. There are different types of graphs (e.g., homogeneous graph, heterogeneous graph, attribute graph, etc), so the input of graph embedding varies in different scenarios. The output of graph embedding is a low-dimensional vector representing a part of the graph (or a whole graph). Figure 1 shows a toy example of embedding a graph into a 2D space in different granularities. I.e., according to different needs, we may represent a node/edge/substructure/whole-graph as a low-dimensional vector. More details about different types of graph embedding input and output are provided in Sec. 3.

In the early 2000s, graph embedding algorithms were mainly designed to reduce the high dimensionality of the non-relational data by assuming the data lie in a low dimensional manifold. Given a set of non-relational high-dimensional data features, a similarity graph is constructed based on the pairwise feature similarity. Then each node in the graph is embedded into a low-dimensional space where connected nodes are closer to each other. Examples of this line of researches are introduced in Sec. 4.1. Since 2010, with the prosperity of graph in various fields, researches in graph embedding take a graph as the input and leverage the auxiliary information (if any) to facilitate the embedding. On the one hand, some of them focus on representing a part of the graph (e.g., node, edge, substructure) (Fig. 1(b)-1(d)) as one vector. To obtain such embedding, they either adopt the state-of-the-art deep learning techniques (Sec. 4.2) or design an objective function to optimize the edge reconstruction probability (Sec. 4.3). On the other hand, there is also some work concentrates on embedding the whole graph as one vector (Fig. 1(e)) for graph level application. Graph kernels (Sec. 4.4) are usually designed to meet this need.

The problem of graph embedding is related to two traditional research problems, i.e., graph analytics [8] and representation learning [9]. Particularly, graph embedding aims to **represent** a **graph** as **low dimensional** vectors while

- *H. Cai is with the Advanced Digital Sciences Center, Singapore. E-mail: hongyun.c@adsc.com.sg.*
- *V. Zheng is with the Advanced Digital Sciences Center, Singapore and UIUC, USA. Email: vincent.zheng@adsc.com.sg.*
- *K. Chang is with the Department of Computer Science, University of Illinois at Urbana-Champaign, USA. Email: kcchang@illinois.edu.*

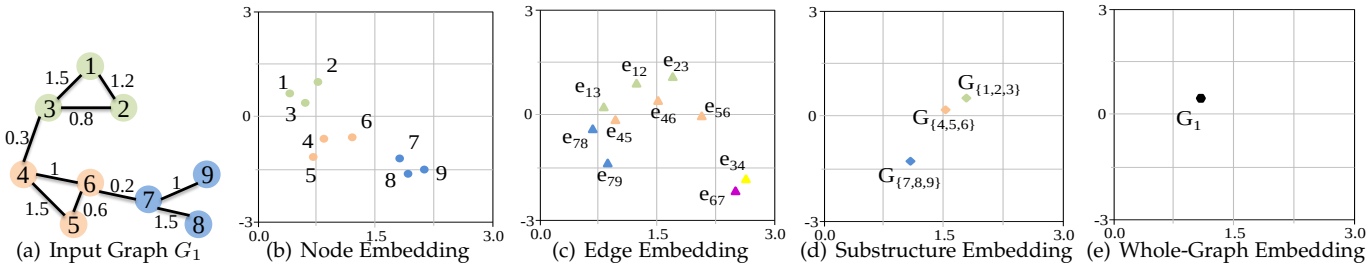


Fig. 1. Toy example of embedding a graph into 2D space with different granularities. $G_{\{1,2,3\}}$ denotes the substructure containing node v_1, v_2, v_3.

the graph structure are preserved. On the one hand, graph analytics aims to mine useful information from graph data. On the other hand, representation learning obtains data representations that make it easier to extract useful information when building classifiers or other predictors [9]. Graph embedding lies in the overlapping of the two problems and focuses on learning the low-dimensional representations. Note that we distinguish graph representation learning and graph embedding in this survey. Graph representation learning does not require the learned representations to be low dimensional. For example, [10] represents each node as a vector with the dimensionality equals to the number of nodes in the input graph. Every dimension denotes the geodesic distance of a node to each other nodes in the graph.

Embedding graphs into low dimensional space is not a trivial task. The challenges of graph embedding depend on the **problem setting**, which consists of embedding input and embedding output. In this survey, we divide the **input graph** into four categorizations, including *homogeneous graph, heterogeneous graph, graph with auxiliary information and graph constructed from non-relational data*. Different types of embedding input carry different information to be preserved in the embedded space and thus pose different challenges to the problem of graph embedding. For example, when embedding a graph with structural information only, the connections between nodes are the target to preserve. However, for a graph with node label or attribute information, the auxiliary information provides graph property from other perspectives, and thus may also be considered during the embedding. Unlike embedding input which is given and fixed, the **embedding output** is task driven. For example, the most common type of embedding output is node embedding which represents close nodes as similar vectors. Node embedding can benefit the node related tasks such as node classification, node clustering, etc. However, in some cases, the tasks may be related to higher granularity of a graph e.g., node pairs, subgraph, whole graph. Hence the first challenge in terms of embedding output is how to find a suitable embedding output type for the specific application task. We categorize four types of graph embedding output, including *node embedding, edge embedding, hybrid embedding and whole-graph embedding*. Different output granularities have different criteria of what a "good" embedding and face different challenges. For example, a good *node embedding* preserves the similarity to its neighbour nodes in the embedded space. In contrast, a good *whole-graph embedding* represents a whole graph as a vector so that the graph-level similarity is preserved.

In observations of the challenges faced in different problem settings, we propose two taxonomies of graph embedding work, by categorizing graph embedding literature based on the problem settings and the embedding techniques. These two taxonomies correspond to what challenges exist in graph embedding and how existing studies address these challenges. In particular, we first introduce different settings of graph embedding problem as well as the challenges faced in each setting. Then we describe how existing studies address these challenges in their work, including their insights and their technique solutions.

Note that although a few attempts have made for graph embedding survey ([11], [12]), they have the following two limitations. First, they usually only proposed one taxonomy of graph embedding techniques. I.e., they categorized graph embedding work base on the embedding techniques only. None of them analyze graph embedding work from the perspective of problem setting, and nor did they summarize the challenges in each setting. Second, only a limited number of related work are covered in existing graph embedding surveys. And there is no analysis for the insights behind each graph embedding technique. A comprehensive review of existing graph embedding work and a high level abstract of the insight for each embedding technique option can foster the future researches in the field of graph embedding by providing an insightful guideline.

1.1 Our Contributions

Below, we summarize our major contributions in this survey.

- We propose a taxonomy of graph embedding based on problem settings and summarize the challenges faced in each setting. We are the first to categorize graph embedding work based on problem setting, which brings new perspectives to understand existing work.
- We provide a detailed analysis of graph embedding techniques. Compared to existing graph embedding surveys, we not only investigate a more comprehensive set of graph embedding work, but also present a summary of the insights behind each technique. In contrast to simply listing how the graph embedding was solved in the past, the summarized insights answer the questions of why the graph embedding can be solved in this way. This can serve as an insightful guideline for future researches.
- We systematically categorize the applications that graph embedding enables and divide the applications as node related, edge related and graph related. For each category, we present the detailed application scenarios as the reference.
- We suggest four promising future research directions in the field of graph embedding in terms of computation

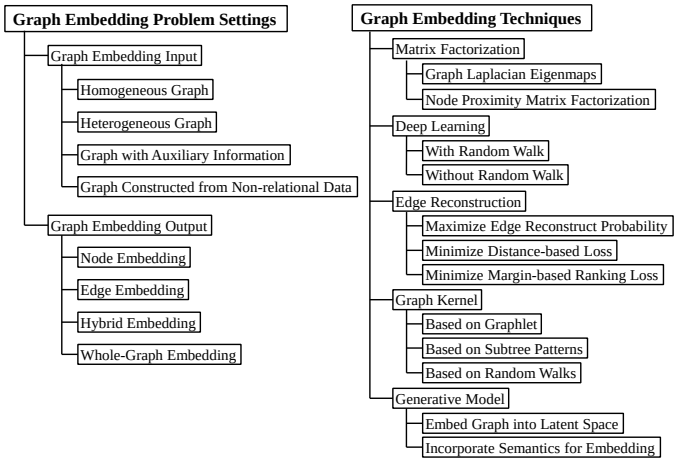


Fig. 2. Graph embedding taxonomies by problems and techniques.

efficiency, problem settings, techniques and applications. For each future direction, we provide a thorough analysis of the disadvantages or deficiency in current graph embedding work and propose the potential direction(s) in which the efforts can be devoted to.

1.2 Organization of The Survey

The rest of this survey is organized as follows. In Sec. 3.2, we introduce the definitions of the basic concepts required to understand the graph embedding problem, and then provide a formal problem definition of graph embedding. In the next two sections, we provide two taxonomies of graph embedding, where the taxonomy structures are illustrated in Fig. 2. Sec. 3 compares the related work based on the problem settings and summarize the challenges faced in each setting. In Sec. 4, we categorize the literature based on the embedding techniques.The insights behind each technique are abstracted, and a detailed comparison of different techniques is provided at the end. After that, we present the applications that graph embedding enables in Sec. 5. We then discuss four potential future research directions in Sec. 6 and concludes this survey in Sec. 7.

2 Problem Formalization

In this section, we first introduce the definition of the basic concepts in graph embedding, and then provide a formal definition of graph embedding problem.

2.1 Notation and Definition

The detailed descriptions of the notations used in this survey can be found in Table 1.

Definition 1. A **graph** is $\mathcal{G} = (V, E)$, where $v \in V$ is a node and $e \in E$ is an edge. $\mathcal{G}$ is associated with a node type mapping function $f_v : V \rightarrow \mathcal{T}^v$ and an edge type mapping function $f_e : E \rightarrow \mathcal{T}^e$.

$\mathcal{T}^v$ and $\mathcal{T}^e$ denote the set of node type set and edge type, respectively. Each node $v_i \in V$ belongs to one particular type, i.e., $f_v(v_i) \in \mathcal{T}^v$. Similarly, for $e_{ij} \in E$, $f_e(e_{ij}) \in \mathcal{T}^e$.

TABLE 1
Notations used in this paper.

Notations	Descriptions
$\|\cdot\|$	The cardinality of a set
$\mathcal{G} = (V, E)$	Graph $\mathcal{G}$ with nodes set V and edges set E
$\hat{\mathcal{G}} = (\hat{V}, \hat{E})$	A substructure of graph $\mathcal{G}$, where $\hat{V} \subseteq V, \hat{E} \subseteq E$
v_i, e_{ij}	A node $v_i \in V$ and an edge $e_{ij} \in E$ connecting v_i and v_j
A	The adjacent matrix of $\mathcal{G}$
A_i	The i-th row vector of matrix A
$A_{i,j}$	The i-th row and j-th column in matrix A
$f_v(v_i)$, $f_e(e_{ij})$	Type of node v_i and type of edge e_{ij}
$\mathcal{T}^v$, $\mathcal{T}^e$	The node type set and edge type set
$\mathcal{N}_k(v_i)$	The k nearest neighbours of node v_i
$X \in \mathbb{R}^{\|V\| \times N}$	A feature matrix, each row X_i is a N-dimensional vector for v_i
y_i, y_{ij}, $y_{\hat{\mathcal{G}}}$	The embedding of node v_i, edge e_{ij}, and structure $\hat{\mathcal{G}}$
d	The dimensionality of the embedding
(h, r, t)	A knowledge graph triplet, with head entity h, tail entity t and the relation between them r
$s_{ij}^{(1)}$, $s_{ij}^{(2)}$	First- and second-order proximity between node v_i and v_j
c	An information cascade
$\mathcal{G}^c = (V^c, E^c)$	A cascade graph which adopts the cascade c

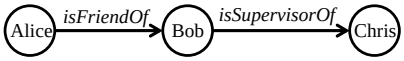


Fig. 3. A toy example of knowledge graph.

Definition 2. A **homogeneous graph** $\mathcal{G} = (V, E)$ is a graph in which $|\mathcal{T}^v| = |\mathcal{T}^e| = 1$. All nodes in $\mathcal{G}$ belong to a single type and all edges belong to one single type.

Definition 3. A **heterogeneous graph** $\mathcal{G} = (V, E)$ is a graph in which $|\mathcal{T}^v| > 1$ and/or $|\mathcal{T}^e| > 1$.

Definition 4. A **knowledge graph** $\mathcal{G} = (V, E)$ is a directed graph whose nodes are *entities* and edges are *subject-property-object* triple facts. Each edge of the form (*head entity*, *relation*, *tail entity*) (denoted as (h, r, t)) indicates that there is a relationship of r from entity h to entity t.

$h, t \in V$ are entities and $r \in E$ is the relation. In this survey, we call (h, r, t) a knowledge graph triplet. E.g., in Fig. 3, there are two triplets: $(Alice, isFriendOf, Bob)$ and $(Bob, isSupervisorOf, Chris)$. Note that the entities and relations in a knowledge graph are usually of different types [13], [14]. Hence, knowledge graph can be viewed as an instance of the heterogeneous graph.

The following proximity measures are usually adopted to quantify the graph property to be preserved in the embedded space. The first-order proximity is the local pairwise similarity only between the nodes connected by edges. It compares the direct connection strength between a node pair. Formally,

Definition 5. The **first-order proximity** between node v_i and node v_j is the weight of the edge e_{ij}, i.e., $A_{i,j}$.

Two nodes are more similar if they are connected by an edge with larger weight. Denote the first-order proximity between v_i and node node v_j as $s_{ij}^{(1)}$, we have $s_{ij}^{(1)} = A_{i,j}$. Let $s_i^{(1)} = [s_{i1}^{(1)}, [s_{i2}^{(1)}, \cdots, [s_{i|V|}^{(1)}]$ denote the first-order proximity between v_i and other nodes. Take the graph in Fig. 1(a) as an example, the first order between v_1 and v_2 is the weight of edge e_{12}, denoted as $s_{12}^{(1)} = 1.2$. And $s_1^{(1)}$ records the weight of edges connecting v_1 and other nodes in the graph, i.e., $s_1^{(1)} = [0, 1.2, 1.5, 0, 0, 0, 0, 0, 0]$.

The second-order proximity compares the similarity of the nodes' neighbourhood structures. The more similar two nodes' neighbourhoods are, the larger second-order proximity they have. Formally,

Definition 6. The **second-order proximity** between node v_i and v_j is the similarity between the neighbourhood structure of v_i and v_j, i.e., $s_{ij}^{(2)}$ is the similarity between $s_i^{(1)}$ and $s_j^{(1)}$.

Again, take Fig. 1(a) as an example: $s_{12}^{(2)}$ is the similarity between $s_1^{(1)}$ and $s_2^{(1)}$. As introduced before, $s_1^{(1)} = [0, 1.2, 1.5, 0, 0, 0, 0, 0, 0]$ and $s_2^{(1)} = [1.2, 0, 0.8, 0, 0, 0, 0, 0, 0]$. Adopt Cosine similarity as the similarity metric, $s_{12}^{(2)} = CosineSimilarity(s_1^{(1)}, s_2^{(1)}) = 0.43$. Likewise, we can calculate $s_{15}^{(2)} = CosineSimilarity(s_1^{(1)}, s_5^{(1)}) = 0$. We can see that the second-order proximity between v_1 and v_5 equals to zero as v_1 and v_5 do not share any common 1-hop neighbours. v_1 and v_2 share a common neighbour v_3, hence their second-order proximity $s_{12}^{(2)}$ is larger than zero.

The **higher-order proximity** can be defined likewise. For example, the k-th-order proximity between node v_i and v_j is the similarity between $s_i^{(k-1)}$ and $s_j^{(k-1)}$. Note that sometimes the higher-order proximities are also defined using some other metrics, e.g., Katz Index, Rooted PageRank, Adamic Adar, etc [11].

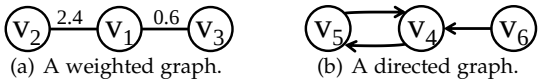
It is worth noting that, in some work, the first-order and second-order proximities are empirically calculated based on the joint probability and conditional probability of two nodes. More details are discussed in Sect. 4.3.2.

Problem 1. **Graph embedding:** Given the input of a graph $\mathcal{G} = (V, E)$, and a predefined dimensionality of the embedding d ($d \ll |V|$), the problem of graph embedding is to convert $\mathcal{G}$ into a d-dimensional space, in which the graph property is preserved as much as possible. The graph property can be quantified using the proximity measures such as the first- and higher-order proximity. Each graph is represented as either one d-dimensional vector (for a whole graph) or a set of d-dimensional vectors where each vector represents the embedding of a part of the graph (e.g., node, edge, substructure).

Fig. 1 shows a toy example of graph embedding with $d = 2$. Given an input graph (Fig. 1(a)), the graph embedding algorithms are applied to convert a node (Fig. 1(b))/ edge (Fig. 1(c)), substructure (Fig. 1(d))/ whole-graph (Fig. 1(e)) as a 2D vector (i.e., a point in a 2D space). In the next two sections, we provide two taxonomies of graph embedding, by categorizing the graph embedding literature based on problem settings and embedding techniques respectively.

3 Problem Settings of Graph Embedding

In this section, we compare existing graph embedding work from the perspective of problem setting, which consists of the embedding input and embedding output. For each setting, we first introduce different types of graph embedding input or output, and then summarize the challenges faced in each setting at the end.



(a) A weighted graph. (b) A directed graph.

Fig. 4. Examples of weighted and directed graphs.

We start with graph embedding input. As a graph embedding setting consists of both input and output, we use node embedding as an example embedding output setting during the introduction of different types of input. The reason is that although there exist various types of embedding output, the majority of graph embedding studies focus on node embedding, i.e., embedding nodes to a low dimensional space where the node similarity in the input graph is preserved. More details about node embedding and other types of embedding output are presented in Sec. 3.2.

3.1 Graph Embedding Input

The input of graph embedding is always a graph. In this survey, we generally divide input graphs in existing graph embedding studies into four categories: homogeneous graph, heterogeneous graph, graph with auxiliary information and constructed graph. Each type of graph carries different information, and thus poses new challenges to the graph embedding problem. Next we will introduce the four type of input graphs one by one and summarize the challenges faced in each embedding input setting at the end.

3.1.1 Homogeneous Graph

The first category of input graph is the homogeneous graph (Def. 2), in which both nodes and edges belong to one single type respectively. The homogeneous graph can be further categorized as the weighted (or directed) graph and the unweighted (or undirected) graph based on whether the edges are weighted (or directed) or not. Figure 4 gives a simple example of a weighted graph and a directed graph.

Undirected and unweighted homogeneous graph is the most basic graph embedding input setting. A number of studies work under this setting, e.g., [1], [15], [16], [17], [18]. They treat all nodes and edges equally, as only the basic structural information of the input graph is available.

Intuitively, the weights and directions of the edges provide more information of the graph, and may help to represent graph more accurately in the embedded space. For example, in Fig. 4(a), v_1 should be embedded closer to v_2 than v_3 because the weight of the edge between v_1 and v_2 is higher. Similarly, v_4 in Fig. 4(b) should be embedded closer to v_5 than v_6 as v_4 and v_5 are connected in both direction. The above information is lost in the unweighted and undirected graph. Noticing the advantages of exploiting the weight and direction property of the graph edges, the graph embedding community start to explore the weighted graph and/or the directed graph. Some of them focus on only one graph property, i.e., either edge weight or edge direction. On the one hand, the **weighted graph** is considered in [19], [20], [21], [22], [23], [24]. Nodes connected by higher-weight edges will be embedded closer. However, their work is still limited to undirected graphs. On the other hand, some work distinguishes directions of edge during the embedding process and preserve the direction information in the embedded space. One example of the **directed graph**

is the social network graph, e.g, [25]. Each user has both followership and followeeship with other users. However, the weight information is unavailable for the social user links. Recently, a more general graph embedding algorithm is proposed, in which both weight and direction properties are considered. In other words, these algorithms (e.g., [3], [26], [27]) can flexibly process **both directed and undirected, weighted and unweighted graph**.

Challenge: *How to capture the diversity of connectivity patterns observed in graphs.* Since only structural information is available in homogeneous graphs, the challenge of homogeneous graph embedding lies in how to preserve these connectivity patterns observed in the input graphs during embedding.

3.1.2 Heterogeneous Graph

The second category of input is the heterogeneous graph (Def. 3), which mainly exist in the below three scenarios.

Community-based question answering (cQA) sites. cQA is an Internet-based crowdsourcing service that enables users to post questions on a cQA website, which are answered by other users later [28]. Intuitively, there are different types of nodes in a cQA graph, e.g., question, answer, user. Existing cQA graph embedding methods distinguish from each other in terms of the links they exploit. For example, [29] exploits two types of links, user-user and user-question. [30] extends [29] by further considering the relative quality rank of question-answer links. In contrast to them, [28] does not utilize user-question links, instead it explores user-answer links in addition to question-answer and user-user links. [31] denotes a ranking pair as a ordered tuple (i, j, k, o, p), meaning that "the j-th answer provided by the k-th user obtains more votes (i.e., thumb-ups) than the o-th answer provided by the p-th user for the i-th question". The authors then exploit users' asymmetric following links as well as a set of pairwise rankings for embedding.

Multimedia Networks. The multimedia network is a network which contains multimedia data, e.g., image, text, etc. For example, both [32] and [33] embed the graphs containing two types of nodes (image and text) and three types of links (the co-occurrence of image-image, text-text and image-text). [34] processes a social curation with two types of nodes (user and image). Only user-image links are considered as it aims to embed users and images into the same space so that their similarity can be directly calculated for recommending images for users. In [35], a click graph is considered which contains image nodes and text query nodes. The image-query edge indicates a click of an image given a query. The click count serves as the edge weight.

Knowledge Graphs. Knowledge graph (Def. 4) is an example of a heterogeneous graph in which the entities (nodes) and relations (edges) are usually of different types. For example, in a film related knowledge graph constructed from Freebase [36], the types of entities can be "*production company*", "*director*", "*actor*", "*film*", etc. And the types of relations can be "*produce*", "*direct*", "*act_in*". A lot of efforts have been devoted to embed knowledge graphs (e.g., [37], [38], [39]). We will introduce them in details in Sec. 4.3.3.

Other heterogeneous graphs also exist. For instance, [40] and [41] work on the mobility data graph, in which the train station, train role and train company nodes are connected by three types of links (station-station, station-role and station-company). [42] embeds a Wikipedia graph with three types of nodes (entity, category and word) and three types of edges (entity-entity, entity-category and word-word). In [43], a geo-tagged social media graph is constructed in which four types of co-occurrence edges (word-word, word-time, word-location, time-location) and two types of neighbourhood edges (location-location and time-time) are established. In addition to the above graphs which are specific for one application scenario, there are some general heterogeneous graphs in which the types of nodes and edges are not specifically defined [44], [45], [46].

Challenge: *How to explore global consistency between different types of objects, and how to deal with the imbalances of different-typed objects, if any.* Different types of objects (e.g., nodes, edges) are embedded into the same space in heterogeneous graph embedding. How to explore the global consistency between them is a problem. Moreover, there may exist imbalance between objects of different types. This data skew should be considered during the embedding process.

3.1.3 Graph with Auxiliary Information

The third category of input graph contains auxiliary information of a node/edge/whole-graph in addition to the structural relations of nodes (i.e., E). Generally, there are the following five different types of auxiliary information.

Label: Label provides the categorical information about a node/edge. Intuitively, nodes belong to different labels should be embedded far away from each other. In order to achieve this, [47] and [48] jointly optimize the embedding objective function together with a classifier function. [49] calculates the pairwise node similarity with the consideration of the node labels and puts a penalty on the similarity between nodes with different labels. [50] considers node labels and edge labels when calculating different graph kernels. [51] and [52] embed the knowledge graph, in which the entity (node) has a semantic category. [53] embeds a more complicated knowledge graph with the entity categories in a hierarchical structure, e.g., the category "book" has two sub-categories "author" and "written_work".

Attribute: The differences between label and attribute information is that a label value is always categorical, while an attribute value can be both discrete and continuous. For example, [54] embeds a graph in which each node has a discrete attribute value (e.g., the atomic number in a molecule). In contrast, [4] represents the node attribute as a continuous high-dimensional vector (e.g., user attribute features in social networks). [55] deals with both discrete and continuous attributes for nodes and edges.

Node feature: Most node feature are text features, which are provided either as a feature vector associated with each node [56], [57] or as a document [58], [59], [60], [61]. For the latter, the documents are further processed to extract feature vectors using techniques such as bag-of-words [58], topic modelling [59], [60], or directly treating "word" as one type of node [61]. Other types of node feature are also possible. E.g., [32] provides both text and image features. These node features enhance the graph embedding performance by providing rich and unstructured information, which is available in many real-world graphs, e.g., user published

content in social graphs, author profiles/papers in citation graphs, etc. Another benefit of incorporating node feature is that it makes inductive graph embedding possible [62].

Information propagation: The cascade graph describes the paths of how the information is propagated in a graph. An example is the retweet in Twitter. Both [63] and [64] incorporate the information propagation knowledge in the their graph embedding methods, but in a different way. In [63], given an original graph $\mathcal{G} = (V, E)$, a cascade graph $\mathcal{G}^c = (V^c, E^c)$ is constructed for each cascade c. The nodes of $\mathcal{G}^c$ are the nodes in V that have adopted c within a duration after its origination. And the edges are those in E with both ends in V^c. Each cascade graph is then represented by a set of sampled node sequences. Their objective is to predict the increment of cascade size. In contrast to [63], [64] aims to embed the users and content information where the similarity between the embedding reflects the probability of the user pair diffuse the content. Hence the one-hop information diffusions are provided as a directed user pairs with the corresponding diffused content (i.e., a propagation edge).

Knowledge base: A knowledge base provides the text associated with or facts between the knowledge concepts. The popular knowledge bases include Wikipedia [65], Freebase [36], YAGO [66], DBpedia [67], etc. Take Wikipedia as an example, the concepts are entities proposed by users and text is the article associated with the entity. In [65], such kind of knowledge base is used to learn a social knowledge graph from a social network by linking each social network user to a given set of knowledge concepts. [68] represents queries and documents in the entity space (provided by a knowledge base) so that the academic search engine can understand the meaning of research concept in queries.

There are other types of auxiliary information in addition to the above five types, such as user check-in data (user-location) [69], user item preference ranking list [70], etc. Moreover, the auxiliary information incorporated in the graph embedding algorithms is not just limited to one type. For instance, [62] and [71] consider both label and node feature information. [72] utilizes node contents as well as node labels to assist the graph embedding process.

Challenge: *How to incorporate the rich and unstructured information so that the learnt embeddings are both representing the topological structure and discriminative in terms of the auxiliary information.* The auxiliary information helps to define node similarity in addition to graph structural information. The challenges of embedding graph with auxiliary information is how to combine these two information sources to define node similarity to be preserved.

3.1.4 Graph Constructed from Non-relational Data

As the last category of graph embedding input, the input graph is not explicitly provided but constructed from the non-relational input data by different strategies. This usually happens when the input data is assumed to lie in a low dimensional manifold.

In most cases, the input is a feature matrix $X \in \mathbb{R}^{|V| \times N}$ where each row X_i is an N-dimensional feature vector for the i-th training instance. A similarity matrix is constructed by calculating the similarity of every instance pair. There are usually two ways to construct graphs from the similarity matrix. A straightforward way is to directly treat the similarity matrix as the adjacency matrix of an invisible graph [73]. However, [73] has two major drawbacks. First, it is based on the Euclidean distance. Second, it doesn't consider any of the neighbouring nodes when calculating the pairwise node similarity. In other words, [73] will treat two nodes with small Euclidean distance as two near nodes. However, if the high-dimensional data lies on or near a curved manifold, the distance between those two nodes over the manifold is much larger than their Euclidean distance [12]. To address these issues, some other methods (e.g., [74], [75], [76]) construct K nearest neighbour (KNN) graph from the similarity matrix first and estimate a new adjacency matrix (node pairwise similarity matrix) based on the KNN graph. For example, Isomap [77] incorporates the geodesic distance for the new adjacency matrix. It first constructs a KNN graph using the Euclidean distance between datapoints, and then finds the shortest path between two nodes as a estimate of the geodesic distance between two nodes. Finally, it apply MDS [73] on the resulting matrix for node embedding.

Another way of graph construction is to extract elements from the input as nodes and then establish edges between nodes based on the co-occurrence of elements. For example, [78] constructs a graph from an image by treating pixels as nodes, and the edges represent spatial relations between pixels. [79] constructs a heterogeneous network by establishing edges of word-word, word-document and word-label co-occurrence. Similarly, [43] extracts three types of nodes (location, time and message) from the GTMS record and therefore forms six types of edges between these nodes. [80] generates a graph using entity mention, target type and text feature as nodes, and establishes three kinds of edges: mention-type, mention-feature and type-type.

In addition to the above edge establishment methods (i.e., pairwise feature similarity based and node co-occurrence based), other graph construction strategies are designed for different purposes. For example, [81] constructs an intrinsic graph to capture the intraclass compactness, and a penalty graph to characterize the interclass separability. The former is constructed by connecting each data point with its neighbouring points of the same class, while the latter connects the marginal points across different classes. [82] constructs a signed graph to exploit the label information. Two nodes are connected by a positive edge if they belong to the same class, and a negative edge if they are from two classes. [83] constructs a hyperedge for each label, and includes all instances with a common label into one hyperedge to capture their joint similarity. In [84], two feedback relational graphs are constructed to encode the pairwise relations in the feedback. In the positive feedback graph, two nodes are connected by an edge if they are both labelled as relevant in the feedback. In the negative feedback graph, two nodes are connected only when one node is relevant and the other is irrelevant. These two graphs help to gather together relevant pairs and keep away irrelevant ones after embedding.

Challenge: *How to construct a graph that encodes the pairwise relations between instances and how to preserve the generated node proximity matrix in the embedded space.* The first challenge

faced by embedding graphs constructed from non-relational data is how to compute the relations between the non-relational data and construct such a graph. After the graph is constructed, the challenge becomes the same as in other input graphs, i.e., how to preserve the node proximity of the constructed graph in the embedded space.

3.2 Graph Embedding Output

The output of graph embedding is a (set of) low dimensional vector(s) representing a (part of) graph. Based on the output granularity, we divide graph embedding output into four categories, including node embedding, edge embedding, hybrid embedding and whole-graph embedding. Different types of embedding facilitate different kinds of applications.

Unlike embedding input which is fixed and given, the embedding output is task driven. For example, node embedding can benefit a wide variety of node related graph analysis tasks. By representing each node as a vector, the node related tasks such as node clustering, node classification, can be performed efficiently in terms of both time and space. However, the graph analytics tasks are not always at node level. In some scenarios, the tasks may be related to higher granularity of a graph, such as node pairs, subgraph, or even a whole graph. Hence, the first **challenge** in terms of embedding output is *how to find a suitable type of embedding output which meets the needs of the specific application task.*

3.2.1 Node Embedding

As the most common embedding output setting, node embedding represents each node as a vector in a low dimensional space. Nodes that are "close" in the graph are embedded to have similar vector representations. The differences between various graph embedding methods lie in how they define the "closeness" between two nodes. First-order proximity (Def. 5) and second-order proximity (Def. 6) are two commonly adopted metrics for node pairwise similarity calculation. In some work, higher-order proximity is also explored to certain extent. For example, [20] captures the k-step ($k = 1, 2, 3, \cdots$) neighbours relations in their embedding. [1] considers nodes' community membership as one of the similarity indicators, i.e., two nodes belonging to the same community will be embedded closer.

Challenge: *How to define the node pairwise proximity in various types of input graph and how to encode the proximity in the learnt embeddings.* The challenges of node embedding mainly come from defining the node proximity in the input graph. In Sec 3.1, we have elaborated the challenges of node embedding with different types of input graphs.

Next, we will introduce other types of embedding output as well as the new challenges posed by these outputs.

3.2.2 Edge Embedding

In contrast to node embedding, the edge embedding aims to represent an edge as a low-dimensional vector. The edge embedding is useful in the following two scenarios.

Firstly, knowledge graph embedding (e.g., [85], [86], [87]) learns embedding for both nodes and edges. Each edge in a knowledge graph is a relation from a head entity to a tail entity, denoted as a triplet $< h, r, t >$. The embedding is learnt to preserve the relations between two entities in the embedded space, so that a missing entity/relation can be correctly predicted given the other two components in the triplet. Secondly, some work (e.g., [27], [64]) embeds a node pair as a vector feature to either make the node pair comparable to other nodes or predict the existence of a link between the node pair. For instance, [64] tries to propose a content-social influential feature for user-user interaction probability given a content. Hence it embeds the directed user pairs and content in a space that the distance between the user pair vector and the content vector is closer if they can be found in the training triplets and vice versa. [27] embeds a pair of nodes using a bootstrapping approach over the node embedding, to facilitate the prediction of whether a link exists between two nodes in a graph.

In summary, the edge embedding benefits the edge (/node pairs) related graph analysis, such as link prediction, knowledge graph entity/relation prediction, etc.

Challenge: *How to define the edge-level similarity and how to model the asymmetric property of the edges, if any.* The edge proximity is different from node proximity as an edge contains a pair of nodes and usually denotes the node pairwise relation. Moreover, unlike nodes, the edges may be directed, and this asymmetric property should be encoded in the learnt edge representations.

3.2.3 Hybrid Embedding

Hybrid embedding is the embedding of a combination of different types of graph components, e.g, node + edge (i.e., substructure), node + community.

Substructure embedding has been studied in a quantity of work. For example, [44] studies the problem of proximity embedding, which embeds the graph structure between two possibly distance nodes to support semantic proximity search. [88] learns the embedding for subgraphs (e.g., graphlets) so as to define the graph kernels for graph classification. [89] utilizes the knowledge base to learn to answer questions by embedding the words (in questions and answers), entities and relations (in knowledge base) in one space. To enrich the information about the answer, it encodes the representation of both path and subgraph from the entity mentioned in the question to the answer entity.

Compared to subgraph embedding, community embedding has only attracted limited attention. [1] proposes to consider a community-aware high-order proximity for node embedding. It jointly optimizes the community detection and community-aware node embedding in a unified framework. A node's embedding is similar to the embedding of the community which it belongs to. [90] also jointly solves node embedding, community detection and community embedding together. Rather than representing a community as a vector, it defines each community embedding as a multivariate Gaussian distribution in the low dimensional space so as to characterize how its member nodes distribute.

The embedding of substructure or community can also be derived by aggregating the individual node and edge embedding inside it. However, such kind of "indirect" approach is not optimized to represent the structure. Moreover, node embedding and community embedding can reinforce each other. I.e., better node embedding is learnt by incorporating the community-aware high-order proximity,

while better communities are detected when more accurate node embedding is generated.

Challenge: *How to generate the target substructure and how to embed different types of graph components in one common space.* In contrast to other types of embedding output, the target to embed in hybrid embedding (e.g., subgraph, community) is not given. Hence the first challenge is how to generate such kind of embedding target structure. Furthermore, different types of targets (e.g., community, node) may be embedded in one common space simultaneously. How to address the heterogeneity of the embedding target types is a problem.

3.2.4 Whole-Graph Embedding

The last type of output for graph embedding method is the embedding of a whole graph usually for small graphs, such as proteins, molecules, etc. In this case, a graph is represented as one vector and two similar graphs are embedded to be closer in the embedded space.

Whole-graph embedding benefits the graph classification task by providing a straightforward and efficient solution for calculating graph similarities [49], [55], [91]. In [49], [55], each graph is represented as one vector. To establish a compromise between the embedding time (efficiency) and the ability to preserve information (expressiveness), [91] designs a hierarchical graph embedding framework. It thinks that accurate understanding of the global graph information requires the processing of substructures in different scales. A graph pyramid is formed where each level is a summarized graph at different scales. Then the graph is embedded at all levels and embeddings from different levels are concatenated into one vector. In [63], a cascade graph is constructed. It learns the embedding for a whole cascade graph, and then trains a multi-layer perceptron to predict the increment of the size of the cascade graph in the future.

Challenge: *How to capture the properties of a whole graph and how to make a trade-off between expressivity and efficiency.* Embedding a whole graph requires to capture the property of a whole graph and is thus more time consuming compared to other types of embedding. One key challenge of whole-graph embedding is how to make a choice between the expressive power of the learnt embedding and the efficiency of the embedding algorithm.

4 Graph Embedding Techniques

In this section, we categorize graph embedding methods based on the techniques they used. Generally, the graph embedding aims to represent a graph in a low dimensional space which preserves as much graph property information as possible. The differences between different graph embedding algorithms lie in how they define the graph property to be preserved. Different algorithms have different insights of the node(/edge/substructure/whole-graph) similarities and how to preserve them in the embedded space. Next, we will introduce the insight of each graph embedding technique, as well as how they quantify the graph property and solve the graph embedding problem.

4.1 Matrix Factorization

Matrix factorization based graph embedding represent graph property (e.g., node pairwise similarity) in the form of a matrix and factorize this matrix to obtain node embedding [11]. The pioneer studies in graph embedding usually solve graph embedding in this way. In most cases, the input is a graph constructed from non-relational high dimensional data features as introduced in Sec. 3.1.4. And the output is a set of node embedding (Sec. 3.2.1). The problem of graph embedding can thus be treated as a structure-preserving dimension reduction method which assumes the input data lie in a low dimensional manifold. There are two types of matrix factorization based graph embedding. One is to factorize **graph Laplacian eigenmaps**, and the other is to directly factorize **the node proximity matrix**.

4.1.1 Graph Laplacian Eigenmaps

Insight: *The graph property to be preserved can be interpreted as pairwise node similarities, thus a larger penalty is imposed if two nodes with larger similarity are embedded far apart.*

Based on the above insight, the optimal embedding y can be derived by the below objective function [94].

$$y^* = \arg\min \sum\nolimits_{i \neq j} y_i - y_j{}^2 W_{ij} = \arg\min y^T L y \quad (1)$$

where W_{ij} is the "defined" similarity between node v_i and v_j, $L = D - W$ is the graph Laplacian. D is the diagonal matrix where $D_{ii} = \sum_{j \neq i} W_{ij}$. The bigger the value D_{ii} is, the more "important" y_i is [93]. A constraint $y^T D y = 1$ is usually imposed in Eq. 1 to remove an arbitrary scaling factor in the embedding. Eq. 1 then reduces to:

$$y^* = \arg \min_{y^T D y = 1} y^T L y = \arg\min \frac{y^T L y}{y^T D y} = \arg\max \frac{y^T W y}{y^T D y} \quad (2)$$

The optimal y's are the eigenvectors corresponding to the maximum eigenvalue of eigenproblem $Wy = \lambda D y$.

The above graph embedding is *transductive* because it can only embed the nodes that exist in the training set. In practice, it might also need to embed the new coming nodes that have not been seen in training. One solution is to design a linear function $y = X^T a$ so that the embedding can be derived as long as the node feature is provided. Consequently, for *inductive* graph embedding, Eq. 1 becomes finding the optimal a in the below objective function:

$$a^* = \arg\min \sum\nolimits_{i \neq j} \|a^T X_i - a^T X_j\|^2 W_{ij} = \arg\min a^T X L X^T a \quad (3)$$

Similar to Eq. 2, by adding the constraint $a^T X D X^T a = 1$, the problem in Eq. 3 becomes:

$$a^* = \arg\min \frac{a^T X L X^T a}{a^T X D X^T a} = \arg\max \frac{a^T X W X^T a}{a^T X D X^T a} \quad (4)$$

The optimal a's are the eigenvectors corresponding to the maximum eigenvalue of eigenproblem $XWX^T a = \lambda X D X^T a$.

The differences of existing studies mainly lie in how they calculate the pairwise node similarity W_{ij}, and whether they use linear function $y = X^T a$ or not. Some attempts [81], [97] have been made to summarize existing Laplacian eigenmaps based graph embedding methods using a general framework. But their surveys only cover a limited quantity of work. In Table 2, we summarize existing Laplacian eigenmaps based graph embedding studies and compare how they calculate W and what objective function they adopt.

The initial study MDS [73] directly adopted the Euclidean distance between two feature vectors X_i and X_j

TABLE 2
Graph Laplacian eigenmaps based graph embedding.

GE Algorithm	W	Objective Function
MDS [73]	W_{ij} = Euclidean distance (X_i, X_j)	Eq. 2
Isomap [77]	KNN, W_{ij} is the sum of edge weights along the shortest path between v_i and v_j	Eq. 2
LE [92]	KNN, $W_{ij} = \exp(\frac{\|X_i - X_j\|^2}{2t^2})$	Eq. 2
LPP [93]	KNN, $W_{ij} = \exp(\frac{\|X_i - X_j\|^2}{t})$	Eq. 4
ARE [84]	KNN, $W_{ij} = \exp(\frac{-\rho^2(X_i X_j)}{t})$, $W_{ij}^{ARE} = \begin{cases} -\gamma, X_i \in F^+ \& X_j \in F^+ \\ 1, \mathcal{L}(X_i) \neq \mathcal{L}(X_j) \\ 0, otherwise \end{cases}$ F^+ denotes the images relevant to a query, γ controls the unbalanced feedback	$a^* = \arg\max \frac{a^T X L^{ARE} X^T a}{a^T X L X^T a}$
SR [94]	KNN, $W_{ij} = \begin{cases} 1, \mathcal{L}(X_i) = \mathcal{L}(X_j) \\ 0, \mathcal{L}(X_i) \neq \mathcal{L}(X_j) \\ W_{ij}, otherwise \end{cases}$ $W_{ij}^{SR} = \begin{cases} 1/l_r, \mathcal{L}(X_i) = \mathcal{L}(X_j) = C_r \\ 0, otherwise \end{cases}$ C_r is the r-th class, l_r=$\|X_i \in X : \mathcal{L}(X_i) = C_r\|$	$a^* = \arg\max \frac{a^T X W^{SR} X^T a}{a^T X (D^{SR} + L) X^T a}$
HSL [83]	$S = I - L$, where L is normalized hypergraph Laplacian	$a^* = \arg\max tr(a^T X S X^T a)$, s.t. $a^T X X^T a = I_k$
MVU [95]	KNN, $W^* = \arg\max tr(W)$, s.t. $W \geq 0, \sum_{ij} W_{ij} = 0$ and $\forall i, j$, where $W_{ii} - 2W_{ij} + W_{jj} = \|X_i - X_j\|^2$	Eq. 2
SLE [82]	KNN, $W_{ij} = \begin{cases} 1/l_r, \mathcal{L}(X_i) = \mathcal{L}(X_j) = C_r \\ -1, \mathcal{L}(X_i) \neq \mathcal{L}(X_j) \end{cases}$ C_r is the r-th class, l_r=$\|X_i \in X : \mathcal{L}(X_i) = C_r\|$	Eq. 4
NSHLRR [75]	normal graph: KNN, $W_{ij} = 1$	Eq. 2
	hypergraph: $W(\mathbf{e})$ is the weight of a hyperedge $\mathbf{e}$ $h(v, \mathbf{e}) = \begin{cases} 1, v \in \mathbf{e} \\ 0, otherwise \end{cases}$, $d(\mathbf{e}) = \sum_{v \in \mathcal{V}} h(v, e)$	$y^* = \arg\min \sum_{\mathbf{e}} \|y_i - y_j\|^2 \frac{W(\mathbf{e})}{d(\mathbf{e})}$
[76]	$W_{ij} = \begin{cases} \frac{\|X_i - X_{m+1}\|_2^2 - \|X_i - X_j\|_2^2}{k\|X_i - X_{k+1}\|_2^2 - \sum_{m=1}^k \|X_i - X_k\|_2^2}, j \leq k \\ 0, j > k \end{cases}$	$y^* = \arg \min_{y^T y = 1} \sum_{i \neq j} W_{ij} \min(\|y_i - y_j\|_2^p, \theta)$
PUFS [74]	KNN, $W_{ij} = \exp(-\frac{\|X_i - X_j\|^2}{2t^2})$	Eq. 4 +(must-link and cannot link constraints)
RF-Semi-NMF-PCA [96]	KNN , $W_{ij} = 1$	Eq. 2 +$\mathcal{O}$(PCA) + $\mathcal{O}$(kmeans)

as W_{ij}. Eq. 2 is used to find the optimal embedding y's. MDS does not consider the neighbourhood of nodes, i.e., any pair of training instances are considered as connected. The follow-up studies (e.g., [77], [92], [93], [98]) overcome this problem by first constructing a k nearest neighbour (KNN) graph from the data feature. Each node is only connected with its top k similar neighbours. After that, different methods are utilized to calculate the similarity matrix W so as to preserve as much desired graph property as possible.

When auxiliary information (e.g., label, attribute) is available, the objective function is adjusted to preserve the richer information. E.g., [94] constructs an adjacency graph W and a labelled graph W^{SR}. The objective function then consists of two parts, one focus on preserving the dataset's local geometric structure as in LPP [93], and the other tries to get the embedding with the best class separability on the labelled training data. Similarly, [84] also constructs two graphs, an adjacency graph W which encodes local geometric structures and a feedback relational graph W^{ARE} that encodes the pairwise relations in users' relevance feedbacks. RF-Semi-NMF-PCA [96] simultaneously consider clustering, dimensionality reduction and graph embedding by constructing an objective function that consists of three components: PCA, k-means and graph Laplacian regularization.

Some other work thinks that W cannot be constructed by easily enumerating node pairwise relationships. Instead, they adopt semidefinite programming (SDP) to learn W. Specifically, SDP [99] aims to find an inner product matrix that maximizes the pairwise distances between any two inputs which are not connected in the graph while preserving the nearest neighbors distances. MVU [95] constructs such matrix and then applies MDS [73] on the learned inner product matrix. [2] proves that regularized LPP [93] is equivalent to regularized SR [94] if W is symmetric, doubly stochastic, PSD and with rank p. It constructs such kind of similarity matrix so as to solve LPP liked problem efficiently.

In addition to solving the above generalized eigenvalue problem, another line of studies tries to directly factorize node proximity matrix as will be introduced next.

4.1.2 Node Proximity Matrix Factorization

Insight: *Node proximity can be approximated in low-dimensional space using matrix factorization. The objective of preserving the node proximity is to minimize the loss during the approximation.*

Given the node proximity matrix W, the objective is:

$$\min \|W - YY^{cT}\| \tag{5}$$

where $Y \in \mathbb{R}^{|V| \times d}$ is the node embedding, and $Y^c \in \mathbb{R}^{|V| \times d}$ is the embedding for the context nodes [20].

Eq. 5 aims to find an optimal rank-d approximation of the proximity matrix W (d is the dimensionality of the embedding). One popular solution is applying SVD (Singular Value Decomposition) on W [105]. Formally,

$$W = \sum_{i=1}^{|V|} \sigma_i u_i u_i^{cT} \approx \sum_{i=1}^{d} \sigma_i u_i u_i^{cT} \tag{6}$$

where $\{\sigma_1, \sigma_2, \cdots, \sigma_{|V|}\}$ is the singular values sorted in decreasing order, u_i and u_i^c are singular vectors of σ_i. The optimal embedding is obtained using the largest d singular values and corresponding singular vectors as follows:

$$\begin{aligned} Y &= [\sqrt{\sigma_1} u_1, \cdots, \sqrt{\sigma_d} u_d] \\ Y^c &= [\sqrt{\sigma_1} u_1^c, \cdots, \sqrt{\sigma_d} u_d^c] \end{aligned} \tag{7}$$

Depending on whether the asymmetric property is preserved or not, the embedding of node i is either $y_i = Y_i$ [20], [50], or the concatenation of the Y_i and Y_i^c, i.e., $y_i = [Y_i, Y_i^c]$ [101]. There exist other solutions for Eq. 5, such as regularized Gaussian matrix factorization [23], low-rank matrix

TABLE 3
Node proximity matrix factorization based graph embedding. $\mathcal{O}(*)$ denotes the objective function; *e.g.*, $\mathcal{O}$(SVM classifier) denotes the objective function of a SVM classifier.

GE Algorithm	W	Objective Function
[50]	$W_{ij} = \begin{cases} 1, e_{ij} \in E \\ 0, otherwise \end{cases}$.	Eq. 5
SPE [100]	KNN, $W^* = \arg\max_{k \geq 0} tr(W\hat{A})$, s.t. $D_{ij} > (1 - \hat{A}_{ij}) \max_m (\hat{A}_{im} D_{im})$ $\hat{A}$ is a connectivity matrix that describes local pairwise distances	Eq. 5
HOPE [101]	Katz Index $W = (\mathbf{I} - \beta A)^{-1} \beta A$; Personalized Pagerank $W = (1-\alpha)(\mathbf{I} - \alpha P)^{-1}$ Common neighbors $W = A^2$; Adamic-Adar $W = A(1/\sum_j (A_{ij} + A_{ji}))A$	Eq. 5
GraRep [20]	$W_{ij}^k = \log(\frac{\hat{A}_{ij}^k}{\sum_t \hat{A}_{ij}^k}) - \log(\frac{\lambda}{\lvert V \rvert})$, where $\hat{A} = D^{-1}S$, $D_{ij} = \begin{cases} \sum_p A_{ip}, i = j \\ 0, i \neq j \end{cases}$	Eq. 5
CMF [42]	PPMI	Eq. 5
TADW [56]	PMI	Eq. 5 with text feature matrix
[23]	A	$y^* = \arg\min \sum_{e_{ij} \in E} (A_{ij} - < y_i, y_j >)^2 + \frac{\lambda}{2} \sum_i \lVert y_i \rVert^2$
MMDW [48]	PMI	Eq. 5 + $\mathcal{O}$(SVM classifier)
HSCA [57]	PMI	$\mathcal{O}$(MMDW) + (1^{st} order proximity constraint)
MVE [102]	KNN, $W^* = \arg\min tr(W(\sum_{i=1}^d v_i v_i^T + \sum_{i=d+1}^N v_i v_i^T))$ where v_i is eigenvector of a pairwise distance matrix, d is embedding dimensionality	Eq. 5
M-NMF [1]	$W = s^{(1)} + 5s^{(2)}$	Eq. 5 + $\mathcal{O}$(community detection) + (community proximity constraint)
ULGE [2]	$W = Z\Delta^{-1}Z^T$, where $Z^* = \underset{z_i^T 1 = 1, z_i \geq 0}{\arg\min} \sum_{j=1}^m \lVert X_i - u_j \rVert_2^2 z_{ij} + \gamma \sum_{j=1}^m z_{ij}^2$	$a^* = \arg\min \lVert a^T X - F_p \rVert_F^2 + \alpha \lVert a \rVert_F^2$ whereF_p is the top p eigenvectors of W
LLE [98]	KNN, $W^* = \arg\min \sum_i \lVert X_i - \sum_j W_{ij} X_j \rVert^2$	$y^* = \arg\min \sum_i \lVert y_i - \sum_j W_{ij} y_j \rVert^2$
RESCAL [103]	$W_{ijk} = \begin{cases} 1, (h_i, r_j, t_k) \text{ exists} \\ 0, otherwise \end{cases}$	$\min \sum_k \lVert W_k - Y R_k Y^T \rVert_F^2 + \lambda(\lVert Y \rVert_F^2 + \sum_k \lVert R_k \rVert_F^2)$
FONPE [104]	KNN, $W^* = \arg\min \sum_i \lVert X_i - \sum_j W_{ij} X_j \rVert^2$	$\min \lVert F - FW^T \rVert_F^2 + \beta \lVert P^T X - F \rVert_F^2$, s.t. $P^T P = \mathbf{I}$

factorization [56], or adding other regularizers to enforce more constraints [48]. We summarize all the node proximity matrix factorization based graph embedding in Table 3.

Summary: Matrix factorization is mostly used to embed the graph constructed from non-relational data (Sec. 3.1.4) for node embedding (Sec. 3.2.1) as this is the original setting of the graph Laplacian eigenmaps based problems. In a few work [23], [50], it is also used to embed homogeneous graphs (Sec. 3.1.1).

4.2 Deep Learning

Deep learning (DL) has shown its outstanding performance in a wide variety of research fields, such as computer vision, language modeling, etc. DL based graph embedding applies DL models on graphs. These models are either a direct adoption from other fields or a new neural network model specifically designed for embedding graph data. The input is either paths sampled from a graph or the whole graph itself. Consequently, we divide the DL based graph embedding into two categories based on whether the random walk is adopted to sample paths from a graph.

4.2.1 DL based Graph Embedding with Random Walk

Insight: *The second-order proximity in a graph can be preserved in the embedded space by maximizing the probability of observing the neighbourhood of a node conditioned on its embedding.*

In the first category of deep learning based graph embedding, a graph is represented as a set of random walk paths sampled from it. The deep learning methods are then applied to the sampled paths for graph embedding which preserves graph properties carried in the paths.

In view of above insight, DeepWalk [16] adopts a neural language model (SkipGram) for graph embedding. SikpGram [108] aims to maximize the co-occurrence probability among the words that appear within a window w. DeepWalk first samples a set of paths from the input graph using truncated random walk (i.e., uniformly sample a neighbour of the last visited node until the maximum length is reached). Each path sampled from the graph corresponds to a sentence from the corpus, where a node corresponds to a word. Then SkipGram is applied on the paths to maximize the probability of observing a node's neighbourhood conditioned on its embedding. In this way, nodes with similar neighbourhood (have larger second-order proximity) will share similar embedding. The objective function of DeepWalk is as follows:

$$\min_y -\log P(\{v_{i-w}, \cdots, v_{i-1}, v_{i+1}, \cdots, v_{i+w}\} | y_i) \quad (8)$$

where w is the window size which restricts the size of random walk context. SkipGram removes the ordering constraint, and Eq. 8 is transformed to:

$$\min_y -\log \sum_{-w \leq j \leq w} P(v_{i+j} | y_i) \quad (9)$$

where $P(v_{i+j}|y_i)$ is defined using the softmax function:

$$P(v_{i+j}|y_i) = \frac{\exp(y_{i+j}^T y_i)}{\sum_{k=1}^{|V|} \exp(y_k^T y_i)} \quad (10)$$

Note that calculating Eq. 10 is not feasible as the normalization factor (i.e., the summation over all inner product with every node in a graph) is expensive. There are usually two solutions to approximate the full softmax: hierarchical softmax [109] and negative sampling [109].

Hierarchical softmax: To efficiently solve Eq. 10, a binary tree is constructed in which the nodes are assigned to the leaves. Instead of enumerating all nodes as in Eq. 10, only the path from the root to the corresponding leaf needs to be evaluated. The optimization problem becomes maximizing the probability of a specific path in the tree. Suppose the path to leaf v_i is a sequence of nodes $(b_0, b_1, \cdots, b_{log(|V|)})$, where $b_0 =$ root, $b_{log(|V|)} = v_i$. Eq. 10 then becomes:

TABLE 4
Deep Learning based graph embedding *with* random walk paths.

GE Algorithm	Ransom Walk Methods	Preserved Proximity	DL Model
DeepWalk [16]	truncated random walk	2^{nd}	SkipGram with hierarchical softmax (Eq. 11)
[33]	truncated random walk	2^{nd} (word-image)	
GenVector [65]	truncated random walk	2^{nd} (user-user & concept-concept)	
Constrained DeepWalk [24]	sampling with edge weight	2^{nd}	
DDRW [47]	truncated random walk	2^{nd} + class identity	
TriDNR [72]	truncated random walk	2^{nd} (among node, word & label)	
node2vec [27]	BFS + DFS	2^{nd}	SkipGram with negative sampling (Eq. 12)
UPP-SNE [106]	truncated random walk	2^{nd} (user-user & profile-profile)	
Planetoid [62]	sampling node pairs by labels and structure	2^{nd} + label identity	
NBNE [18]	sampling direct neighbours of a node	2^{nd}	
DGK [88]	graphlet kernel: random sampling [107]	2^{nd} (by graphlet)	SkipGram (Eqs. 11–12)
metapath2vec [46]	meta-path based random walk	2^{nd}	heterogeneous SkipGram
ProxEmbed [44]	truncate random walk	node ranking tuples	LSTM
HSNL [28]	truncate random walk	2^{nd} + QA ranking tuples	
RMNL [29]	truncated random walk	2^{nd} + user-question quality ranking	
DeepCas [63]	Markov chain based random walk	information cascade sequence	GRU
MRW-MN [35]	truncated random walk	2^{nd} + cross-modal feature difference	DCNN+ SkipGram

$$P(v_{i+j}|y_i) = \prod_{t=1}^{\log(|V|)} P(b_t|y_i) \quad (11)$$

where $P(b_t)$ is a binary classifier: $P(b_t|v_i) = \sigma(y_{b_t}^T y_i)$. $\sigma(\cdot)$ denotes the sigmoid function. y_{b_t} is the embedding of tree node b_t's parent. The hierarchical softmax reduces time complexity of SkipGram from $\mathcal{O}(|V|^2)$ to $\mathcal{O}(|V|log(|V|))$.

Negative sampling [109]: The key idea of negative sampling is to distinguish the target node from noises using logistic regression. I.e., for a node v_i, we want to distinguish its neighbour v_{i+j} from other nodes. A noise distribution $P_n(v_i)$ is designed to draw the negative samples for node v_i. Each $\log P(v_{i+j}|y_i)$ in Eq. 9 is then calculated as:

$$\log \sigma(y_{i+j}^T y_i) + \sum_{t=1}^{K} E_{v_t \sim P_n}[\log \sigma(-y_{v_t}^T y_i)] \quad (12)$$

where K is the number of negative nodes that are sampled. $P_n(v_i)$ is a noise distribution, e.g., a uniform distribution ($P_n(v_i) \sim \frac{1}{|V|}, \forall v_i \in V$). The time complexity of SkipGram with negative sampling is $\mathcal{O}(K|V|)$.

The success of DeepWalk [16] motivates many subsequent studies which apply deep learning models (e.g., SkipGram or Long-Short Term Memory (LSTM) [110]) on the sampled paths for graph embedding. We summarize them in Table 4. As shown in the table, most studies follow the idea of DeepWalk but change the settings of either random walk sampling methods ([24], [27], [62], [62]) or proximity (Def. 5 and Def. 6) to preserve ([33], [47], [62], [65], [72]). [46] designs meta-path-based random walks to deal with heterogeneous graphs and a heterogeneous SkipGram which maximizes the probability of having the hetegeneous context for a given node. Apart from SkipGram, LSTM is another popular deep learning model adopted in graph embedding. Note that SkipGram can only embed one single node. However, sometimes we may need to embed a sequence of nodes as a fixed length vector, e.g., represent a sentence (i.e., a sequence of words) as one vector. LSTM is then adopted in such scenarios to embed a node sequence. For example, [28] and [29] embed the sentences from questions/answers in cQA sites, and [44] embeds a sequence of nodes between two nodes for proximity embedding. A ranking loss function is optimized in these work to preserve the ranking scores in the training data. In [63], GRU [111] (i.e., a recurrent neural network model similar to LSTM) is used to embed information cascade paths.

4.2.2 DL based Graph Embedding without Random Walk

Insight:*The multi-layered learning architecture is a robust and effective solution to encode the graph into a low dimensional space.*

The second class of deep learning based graph embedding methods applies deep models on a whole graph (or a proximity matrix of a whole graph) directly. Below are some popular deep learning models used in graph embedding.

Autoencoder: Autoencoder aims to minimize the reconstruction error of the output and input by its encoder and decoder. Both encoder and decoder contain multiple nonlinear functions. The encoder maps input data to a representation space and the decoder maps the representation space to a reconstruction space. The idea of adopting autoencoder for graph embedding is similar to node proximity matrix factorization (Sec. 4.1.2) in terms of neighbourhood preservation. Specifically, the adjacency matrix captures node's neighbourhood. If we input the adjacency matrix to autoencoder, the reconstruction process will make the nodes with similar neighbourhood have similar embedding.

Deep Neural Network: As a popular deep learning model, Convolutional Neural Network (CNN) and its variants have been widely adopted in graph embedding. On the one hand, some of them directly use the original CNN model designed for Euclidean domains and reformat input graphs to fit into it. E.g., [55] uses graph labelling to select a fixed-length node sequence from a graph and then assembles nodes' neighbourhood to learn a neighbourhood representation with the CNN model. On the other hand, some other work attempts to generalize the deep neural model to non-Euclidean domains (e.g., graphs). [112] summarizes the representative studies in their survey. Generally, the differences between these approaches lie in the way they formulate a convolution-like operation on graphs. One way is to emulate the Convolution Theorem to define the convolution in the spectral domain [113], [114]. Another is to treat the convolution as neighborhood matching in the spatial domain [71], [115], [116].

Others: There are some other types of deep learning models. E.g., [34] proposes DUIF, which uses a hierarchical softmax as a forward propagation to maximize the modularity. HNE [32] utilizes deep learning techniques to capture the interactions between heterogeneous components, e.g., CNN for image and FC layers for text. ProjE [39] designs a neural network with a combination layer and a projection layer. It defines a pointwise loss (similar to multi-class classification) and a listwise loss (i.e., softmax regression loss) for knowledge graph embedding.

We summarized all deep learning based graph embedding methods (random walk free) in Table 5, and compare the models they use as well as the input for each model.

TABLE 5
Deep learning based graph embedding *without* random walk paths.

GE Algorithm	Deep Learning Model	Model Input
SDNE [19]	autoencoder	A
DNGR [22]	stacked denoising autoencoder	PPMI
SAE [21]	sparse autoencoder	$D^{-1}A$
[55]	CNN	node sequence
SCNN [113]	Spectral CNN	graph
[114]	Spectral CNN with smooth spectral multipliers	graph
MoNet [78]	Mixture model network	graph
ChebNet [115]	Graph CNN a.k.a. ChebNet	graph
GCN [71]	Graph Convolutional Network	graph
GNN [116]	Graph Neural Network	graph
[117]	adapted Graph Neural Network	molecules graph
GGS-NNs [118]	adapted Graph Neural Network	graph
HNE [32]	CNN + FC	graph with image and text
DUIF [34]	a hierarchical deep model	social curation network
ProjE [39]	a neural network model	knowledge graph
TIGraNet [119]	Graph Convolutional Network	graph constructed from images

Summary: Due to its robustness and effectiveness, deep learning has been widely used in graph embedding. Three types of input graphs (except for graph constructed from non-relational data (Sec. 3.1.4)) and all the four types of embedding output have been observed in deep learning based graph embedding methods.

4.3 Edge Reconstruction based Optimization

Overall Insight: *The edges established based on node embedding should be as similar to those in the input graph as possible.*

The third category of graph embedding techniques directly optimizes an edge reconstruction based objective functions, by either maximizing edge reconstruction probability or minimizing edge reconstruction loss. The later is further divided as distance-based loss and margin-based ranking loss. Next, we introduce the three types one by one.

4.3.1 Maximizing Edge Reconstruction Probability

Insight: *Good node embedding maximizes the probability of generating the observed edges in a graph.*

Good node embedding should be able to re-establish edges in the original input graph. This can be realized by maximizing the probability of generating all observed edges (i.e., node pairwise proximity) using node embedding.

The direct edge between a node pair v_i and v_j indicates their *first-order proximity*, which can be calculated as the joint probability using the embedding of v_i and v_j:

$$p^{(1)}(v_i, v_j) = \frac{1}{1+\exp(-y_i{}^T y_j)} \tag{13}$$

The above first-order proximity exists between any pair of connected nodes in a graph. To learn the embedding, we maximize the log-likelihood of observing these proximities in a graph. The objective function is then defined as:

$$\mathcal{O}^{(1)}_{max} = \max \sum_{e_{ij}\in E} \log p^{(1)}(v_i, v_j) \tag{14}$$

Similarly, *second-order proximity* of v_i and v_j is the conditional probability of v_j generated by v_i using y_i and y_j:

$$p^{(2)}(v_j|v_i) = \frac{\exp(y_j^T y_i)}{\sum_{k=1}^{|V|}\exp(y_k^T y_i)} \tag{15}$$

It can be interpreted as the probability of a random walk in a graph which starts from v_i and ends with v_j. Hence the graph embedding objective function is:

$$\mathcal{O}^{(2)}_{max} = \max \sum_{\{v_i,v_j\}\in\mathcal{P}} logp^{(2)}(v_j|v_i) \tag{16}$$

where $\mathcal{P}$ is a set of $\{start\_node, end\_node\}$ in the paths sampled from the graph, i.e., the two end nodes from each sampled path. This simulates the second-order proximity as the probability of a random walk starting from the $start\_node$ and ending with the end_node.

4.3.2 Minimizing Distance-based Loss

Insight: *The node proximity calculated based on node embedding should be as close to the node proximity calculated based on the observed edges as possible.*

Specifically, node proximity can be calculated based on node embedding or empirically calculated based on observed edges. Minimizing the differences between the two types of proximities preserves the corresponding proximity.

For the first-order proximity, it can be computed using node embedding as defined in Eq. 13. The empirical probability is $\hat{p}^{(1)}(v_i, v_j) = A_{ij}/\sum_{e_{ij}\in E} A_{ij}$, where A_{ij} is the weight of edge e_{ij}. The smaller the distance between $p^{(1)}$ and $\hat{p}^{(1)}$ is, the better first-order proximity is preserved. Adopting KL-divergence as the distance function to calculate the differences between $p^{(1)}$ and $\hat{p}^{(1)}$ and omitting some constants, the objective function to preserve the first-order proximity in graph embedding is:

$$\mathcal{O}^{(1)}_{min} = \min -\sum_{e_{ij}\in E} A_{ij} \log p^{(1)}(v_i, v_j) \tag{17}$$

Similarly, the second-order proximity of v_i and v_j is the conditional probability of v_j generated by node v_i (Eq. 15). The empirical probability of $\hat{p}^{(2)}(v_i|v_j)$ is calculated as $\hat{p}^{(2)}(v_j|v_i) = A_{ij}/d_i$, where $d_i = \sum_{e_{ik}\in E} A_{ik}$ is the out-degree (or degree in the case of undirected graph) of node v_i. Similar to Eq. 10, it is very expensive to calculate Eq. 15 and negative sampling is again adopted for approximate computation to improve the efficiency. By minimizing the KL divergence between $\hat{p}(2)(v_j|v_i)$ and $\hat{p}^{(2)}(v_j|v_i)$, the objective function to preserve second-order proximity is:

$$\mathcal{O}^{(2)}_{min} = \min -\sum_{e_{ij}\in E} A_{ij} \log p^{(2)}(v_j|v_i) \tag{18}$$

4.3.3 Minimizing Margin-based Ranking Loss

In the margin-based ranking loss optimization, the edges of the input graph indicate the relevance between a node pair. Some nodes in the graph are usually associated with a set of relevant nodes. E.g., in a cQA site, a set of answers are marked as relevant to a given question. The insight of the loss is straightforward.

Insight: *A node's embedding is more similar to the embedding of relevant nodes than that of any other irrelevant node.*

Denote $s(v_i, v_j)$ as the similarity score for node v_i and v_j, V_i^+ as the set of nodes relevant to v_i and V_i^- as the irrelevant nodes set. The margin-based ranking loss is defined as:

$$\mathcal{O}_{rank} = \min \sum_{v_i^+\in V_i^+} \sum_{v_i^-\in V_i^-} \max\{0, \gamma - s(v_i, v_i^+) + s(v_i, v_i^-)\} \tag{19}$$

where γ is the margin. Minimizing the loss rank encourages a large margin between $s(v_i, v_i^+)$ and $s(v_i, v_i^-)$, and thus enforces v_i to be embedded closer to its relevant nodes than to any other irrelevant nodes.

In Table 6, we summarize existing edge reconstruction based graph embedding methods, based on their objective

TABLE 6
Edge reconstruction based graph embedding. $\mathcal{O}_*^*(\mathcal{T}_{v_i}^v, \mathcal{T}_{v_j}^v)$ refers to one of Eq. 14, Eq. 16 $\sim$ Eq. 19; *e.g.*, $\mathcal{O}_{min}^{(2)}$(word-label) refers to Eq. 18 with a word node and a label node. $\mathcal{T}_{v_i}^v$ denote the type of node v_i.

GE Algorithm	Objectives	Order of Proximity
PALE [17]	$\mathcal{O}_{max}^{(1)}$(node, node)	1^{st}
NRCL [4]	$\mathcal{O}_{rank}$(node, neighbour node) + $\mathcal{O}$(attribute loss)	
PTE [79]	$\mathcal{O}_{min}^{(2)}$(word, word) + $\mathcal{O}_{min}^{(2)}$(word, document) + $\mathcal{O}_{min}^{(2)}$(word, label)	2^{nd}
APP [3]	$\mathcal{O}_{max}^{(2)}$(node, node))	
GraphEmbed [43]	$\mathcal{O}_{min}^{(2)}$(word, word) + $\mathcal{O}_{min}^{(2)}$(word, time) + $\mathcal{O}_{min}^{(2)}$(word, location) + $\mathcal{O}_{min}^{(2)}$(time, location) + $\mathcal{O}_{min}^{(2)}$(location, location) + $\mathcal{O}_{min}^{(2)}$(time, time)	
[40], [41]	$\mathcal{O}_{min}^{(2)}$(station, company), $\mathcal{O}_{min}^{(2)}$(station, role), $\mathcal{O}_{min}^{(2)}$(destination, boarding)	
PLE [80]	$\mathcal{O}_{rank}$(mention-type) + $\mathcal{O}_{min}^{(2)}$(mention, feature) + $\mathcal{O}_{min}^{(2)}$(type, type)	
IONE [25]	$\mathcal{O}_{min}^{(2)}$(node, node) + $\mathcal{O}$(anchor align)	
HEBE [45]	$\mathcal{O}_{min}^{(2)}$(node, other nodes in one hyperedge)	
GAKE [37]	$\mathcal{O}_{max}^{(2)}$(node, neighbour context)+$\mathcal{O}_{max}^{(2)}$(node, path context)+$\mathcal{O}_{max}^{(2)}$(node, edge context)	
CSIF [64]	$\mathcal{O}_{max}^{(2)}$(user pair, diffused content)	
ESR [68]	$\mathcal{O}_{max}^{(2)}$(entity, author) + $\mathcal{O}_{max}^{(2)}$(entity, entity) + $\mathcal{O}_{max}^{(2)}$(entity, words) + $\mathcal{O}_{max}^{(2)}$(entity, venue)	
line [26]	$\mathcal{O}_{min}^{(1)}$(node, node) + $\mathcal{O}_{min}^{(2)}$(node, node))	$1^{st} + 2^{nd}$
EBPR [70]	$\mathcal{O}$(AUC ranking) + $\mathcal{O}_{max}^{(2)}$(node, node) + $\mathcal{O}_{max}^{(2)}$(node, node context)	
[89]	$\mathcal{O}_{rank}$(question, answer)	$1^{st} + 2^{nd}$ + higher

functions and preserved node proximity. In general, most methods use one of the above objective functions (Eq. 14, Eq. 16 $\sim$ Eq. 19) [70] optimizes a AUC ranking loss, which is a substitution loss for margin based ranking loss (Eq. 19). Note that when another task is simultaneously optimized during graph embedding, that task-specific objective will be incorporated in the overall objectives. For instance, [25] aims to align two graphs. Hence an network alignment objective function is optimized together with $\mathcal{O}_{min}^{(2)}$ (Eq. 18).

It is worth noting that most of the existing knowledge graph embedding methods choose to optimize margin-based ranking loss. Recall that a knowledge graph $\mathcal{G}$ consists of a set of triplets (h, r, t) denoting the head entity h is linked to a tail entity t by a relation r. Embedding $\mathcal{G}$ can be interpreted as preserving the ranking of a true triplet (h, r, t) over a false triplet (h', r, t') that does not exist in $\mathcal{G}$. Particularly, in knowledge graph embedding, similar to $s(v_i, v_i^+)$ in Eq. 19, an energy function is designed for a triplet (h, r, t) as $f_r(h, t)$. There is a slight difference between these two functions. $s(v_i, v_i^+)$ denotes the **similarity** score between the embedding of node v_i and v_i^+, while $f_r(h, t)$ is the **distance** score of the embedding of h and t in terms of relation r. One example of $f_r(h, t)$ is $\|h + r - t\|_{l1}$, where relationships are represented as *translations in the embedded space* [86]. Other options of $f_r(h, t)$ are summarized in Table 7. Consequently, for knowledge graph embedding, Eq. 19 becomes:

$$\mathcal{O}_{rank}^{kg} = \min \sum_{(h,r,t)\in\mathcal{S}} \sum_{(h',r,t')\notin\mathcal{S}} \max\{0, \gamma + f_r(h, t) - f_r(h', t')\} \tag{20}$$

where $\mathcal{S}$ is the triples in the input knowledge graph. Existing knowledge graph embedding methods mainly optimize Eq. 20 in their work. The difference of them is how they define $f_r(h, t)$ as summarized in Table 7.

TABLE 7
Knowledge graph embedding using margin-based ranking loss.

GE Algorithm	Energy Function $\mathbf{f_r(h, t)}$
TransE [86]	$\|h + r - t\|_{l1}$
TKRL [53]	$\|M_{rh}h + r - M_{rt}t\|$
TransR [14]	$\|hM_r + r - tM_r\|_2^2$
CTransR [14]	$\|hM_r + r_c - tM_r\|_2^2 + \alpha\|r_c - r\|_2^2$
TransH [13]	$\|(h - w_r^T hw_r) + d_r - (t - w_r^T tw_r)\|_2^2$
SePLi [38]	$\frac{1}{2}\|W_i e_{ih} + b_i - e_{it}\|^2$
TransD [120]	$\|M_{rh}h + r - M_{rt}t\|_2^2$
TranSparse [121]	$\|M_r^h(\theta_r^h)h + r - M_r^t(\theta_r^t)t\|_{l1/2}^2$
m-TransH [122]	$\|\sum_{\rho\in\mathcal{M}(R_r)} a_r(\rho)\mathbb{P}_{n_r}(t(\rho)) + b_r\|^2, t \in \mathcal{N}^{\mathcal{M}(R_r)}$
DKRL [123]	$\|h_d + r - t_d\| + \|h_d + r - t_s\| + \|h_s + r - t_d\|$
ManifoldE [124]	Sphere: $\|\varphi(h) + \varphi(r) - \varphi(t)\|^2$ Hyperplane: $(\varphi(h) + \varphi(r_{head}))^T(\varphi(t) + \varphi(r_{tail}))$ φ is the mapping function to Hilbert space
TransA [125]	$\|h + r - t\|$
puTransE [42]	$\|h + r - t\|$
KGE-LDA [60]	$\|h + r - t\|_{l1}$
SE [85]	$\|R_u h - R_u t\|_{l1}$
SME [87] linear	$(W_{u1}r + W_{u2}h + b_u)^T(W_{v1}r + W_{v2}t + b_v)$
SME [87] bilinear	$(W_{u1}r + W_{u2}h + b_u)^T(W_{v1}r + W_{v2}t + b_v)$
SSP [59]	$-\lambda\|e - s^T es\|_2^2 + \|e\|_2^2$, $S(s_h, s_t) = \frac{s_h + s_t}{\|s_h + s_t\|_2^2}$
NTN [126]	$u_r^T \tanh(h^T W_r t + W_{rh}h + W_{rt}t + b_r)$
HOLE [127]	$r^T(h \star t)$, where $\star$ is circular correlation
MTransE [128]	$\|h + r - t\|_{l1}$

Note that some studies jointly optimize the ranking loss (Eq. 20) and other objectives to preserve more information. E.g., SSP [59] optimizes a topic model loss together with Eq. 20 to utilize node's textual descriptions for embedding. [128] categorizes monolingual relations and uses linear transformation to learn cross-lingual alignment for entities and relations. There also exists some work which defines a matching degree score rather than an energy function for a triplet (h, r, t). E.g., [129] defines a bilinear score function, $v_h^T W_r v_t$ and adds a normality constraint and a commutativity constraint to impose analogical structures among the embedding. ComplEx [130] extends embedding to complex domain and defines the real part of $v_h^T W_r v_t$ as the score.

Summary: Edge reconstruction based optimization is applicable for most graph embedding settings. As far as can be observed, only graph constructed from non-relational data (Sec. 3.1.4) and whole-graph embedding (Sec. 3.2.4) have not been tried. The reason is that reconstructing manually constructed edges is not as intuitive as in other graphs. Moreover, as this technique focuses on the directly observed local edges, it is not suitable for whole-graph embedding.

4.4 Graph Kernel

Insight: *The whole graph structure can be represented as a vector containing the counts of elementary substructures that are decomposed from it.*

Graph kernel is an instance of R-convolution kernels [131], which is a generic way of defining kernels on discrete compound objects by recursively decomposing structured objects into "atomic" substructures and comparing all pairs of them [88]. The graph kernel represents each graph as a vector, and two graphs are compared using an inner product of the two vectors. There are generally three types of "atomic" substructures defined in graph kernel.

Graphlet. A graphlet is an induced and non-isomorphic subgraph of size-k [88]. Suppose graph $\mathcal{G}$ is decomposed into a set of graphlets $\{G_1, G_2, \cdots, G_d\}$. Then $\mathcal{G}$ is embedded as a d-dimensional vector (denoted as $y_{\mathcal{G}}$) of normalized counts. The i-th dimension of $y_{\mathcal{G}}$ is the frequency of the graphlet G_i occurring in $\mathcal{G}$.

Subtree Patterns. In this kernel, a graph is decomposed as its subtree patterns. One example is Weisfeiler-Lehman subtree [49]. In particular, a relabelling iteration process is conducted on a labelled graph (i.e., a graph with discrete node labels). At each iteration, a multiset label is generated based on the label of a node and its neighbours. The new generated multiset label is a compressed label which denotes the subtree patterns, which is then used for the next iteration. Based on Weisfeiler-Lehman test of graph isomorphism, counting the occurrence of labels in a graph is equivalent to counting the corresponding subtree structures. Suppose h iterations of relabelling are performed on graph $\mathcal{G}$. Its embedding $y_{\mathcal{G}}$ contains h blocks. The i-th dimension in the j-th block of $y_{\mathcal{G}}$ is the frequency of the i-th label being assigned to a node in the j-th iteration.

Random Walks. In the third type of graph kernels, a graph is decomposed into random walks or paths and represented as the counts of occurrence of random walks [132] or paths [133] in it. Take paths as an example, suppose graph $\mathcal{G}$ is decomposed into d shortest paths. Denote the i-th path as a triplet (l_i^s, l_i^e, n_i), where l_i^s and l_i^e are the labels of the starting and ending nodes, n_i is the length of the path. Then $\mathcal{G}$ is represented as a d-dimensional vector $y_{\mathcal{G}}$ where the i-th dimension is the frequency of the i-th triplet occurring in $\mathcal{G}$.

Summary: Graph kernel is designed for whole-graph embedding (Sec. 3.2.4) only as it captures the global property of a whole graph. The type of input graph is usually a homogeneous graph (Sec. 3.1.1) [88] or a graph with auxiliary information (Sec. 3.1.3) [49].

4.5 Generative Model

A generative model can be defined by specifying the joint distribution of the input features and the class labels, conditioned on a set of parameters [134]. An example is Latent Dirichlet Allocation (LDA), in which a document is interpreted as a distribution over topics, and a topic is a distribution over words [135]. There are the following two ways to adopt a generative model for graph embedding.

4.5.1 Embed Graph Into The Latent Semantic Space

Insight: *Nodes are embedded into the latent semantic space where the distances among node embeddings can explain the observed graph structure.*

The first type of generative model based graph embedding methods directly embeds a graph in the latent space. Each node is represented as a vector of the latent variables. In other words, it views the observed graph as generated by a model. E.g., in LDA, documents are "embedded" in a "topic" space where documents with similar words have similar topic vector representations. [69] designs a LDA-like model to embed a location-based social network (LBSN) graph. Specifically, the input is locations (documents), each of which contains a set of users (words) who visited that location. Users visit the same location (words appearing in the same document) due to some activities (topics). Then a model is designed to represent a location as a distribution over activities, where each activity has an attractiveness distribution over users. Consequently, both user and location are represented as a vector in the "activity" space.

4.5.2 Incorporate Latent Semantics for Graph Embedding

Insight: *Nodes who are not only close in the graph but also having similar semantics should be embedded closer. The node semantics can be detected from node descriptions via a generative model.*

In this line of methods, latent semantics are used to leverage auxiliary node information for graph embedding. The embedding is decided not only by the graph structure information but also by the latent semantics discovered from other sources of node information. For example, [58] proposes a unified framework which jointly integrates topic modelling and graph embedding. Its principal is that if two nodes are close in the embedded space, they will also share similar topic distribution. A mapping function from the embedded space to the topic semantic space is designed so as to correlate the two spaces. [136] proposes a generative model (Bayesian non-parametric infinite mixture embedding model) to address the issue of multiple relation semantics in knowledge graph embedding. It discovers the latent semantics of a relation and leverages a mixture of relation components for embedding. [59] embeds a knowledge graph from both the knowledge graph triplets and the textual descriptions of entities and relations. It learns the semantic representation of text using topic modelling and restricts the triplet embedding in the semantic subspace.

The difference between the above two directions of methods is that the embedded space is the latent space in the first way. In contrast, in the second way, the latent space is used to integrate information from different sources, and help to embed a graph to another space.

Summary: Generative model can be used for both node embedding (Sec. 3.2.1) [69] and edge embedding (Sec. 3.2.2) [136]. As it considers node semantics, the input graph is usually a heterogeneous graph (Sec. 3.1.2) [69] or a graph with auxiliary information (Sec. 3.1.3) [59].

4.6 Hybrid Techniques and Others

Sometimes multiple techniques are combined in one study. For example, [4] learns edge-based embedding via minimizing the margin-based ranking loss (Sec. 4.3), and learns attribute-based embedding by matrix factorization (Sect. 4.1). [51] optimizes a margin-based ranking loss (Sec. 4.3) with matrix factorization based loss (Sec. 4.1) as regularization terms. [31] uses LSTM (Sec. 4.2) to learn sentences embedding in cQAs and a margin-based ranking loss (Sec. 4.3) to incorporate friendship relations. [137] adopts CBOW/SkipGram (Sec. 4.2) for knowledge graph entity embedding, and then fine-tune the embedding by minimize a margin-based ranking loss (Sec. 4.3). [61] use word2vec (Sec. 4.2) to embed the textual context and TransH (Sec. 4.3) to embed the entity/relations so that the rich context information is utilized in knowledge graph embedding. [138] leverages the heterogeneous information in a knowledge base to improve recommendation performance. It uses

TABLE 8
Comparison of graph embedding techniques.

Category	Subcategory	Advantages	Disadvantages
matrix factorization	graph Laplacian eigenaps	consider global node proximity	large time and space consumption
	node proximity matrix factorization		
deep learning	with random walk	effective and robust	a) only consider local context within a path b) hard to find "optimal" sampling strategy
	without random walk		high computation cost
edge reconstruction	maximize edge reconstruct probability	relatively efficient training	optimization using only observed local information, i.e., edges (1-hop neighbour) or ranked node pairs
	minimize distance-based loss		
	minimize margin-based ranking loss		
graph kernel	based on graphlet	efficient, only counting the desired atom substructure	a) substructures are not independent b) embedding dimensionality grows exponentially
	based on subtree patterns		
	based on random walks		
generative model	embed graph in the latent space	interpretable embedding	a) hard to justify the choice of distribution b) require a large amount of training data
	incorporate latent semantics for graph embedding	naturally leverage multiple information sources	

TransR (Sec. 4.3) for network embedding, and uses auto-encoders for textual and visual embedding (Sec. 4.2). Finally, a generative framework (Sec. 4.5) is proposed to integrate collaborative filtering with items semantic representations.

Apart from the introduced five categories of techniques, there exist other approaches. [91] presents embedding of a graph by its distances to prototype graphs. [15] first embeds a few landmark nodes using their pairwise shortest path distances. Then other nodes are embedded so that their distances to a subset of landmarks are as close as possible to the real shortest paths. [4] jointly optimizes a link-based loss (maximize the likelihood of the observing a node's neighbours than non-neighbours) and an attribute-based loss (learn a linear projection based on link-based representation). KR-EAR [140] distinguishes the relations in knowledge graph as attribute-based and relation-based. It constructs a relational triple encoder (TransE, TransR) to embed the correlations between entities and relations, and an attributional triple encoder to embed the correlations between entities and attributes. [141] proposes struc2vec, a flexible framework to learn representations that capture the structural identity of nodes. [139] proposes a unified model which improves the performance of many existing graph embedding methods (e.g., DeepWalk [16], GraRep [20], etc) by implicitly approximating higher order proximity matrix.

4.7 Summary

We now summarize and compare all the five categories of introduced graph embedding techniques in Table 8 in terms of their advantages and disadvantages.

Matrix factorization based graph embedding learns the representations based on the statistics of global pairwise similarities. Hence it can outperform deep learning based graph embedding (random walk involved) in certain tasks as the latter relies on separate local context windows [142], [143]. However, either the proximity matrix construction or the eigendecomposition of the matrix is time and space consuming [144], making matrix factorization inefficient and unscalable for large graphs.

Deep learning based graph embedding has shown its outstanding and robust performance due to its multi-layered learning architecture. However, both the approaches with and without random walk sampling have some disadvantages. On the one hand, for DL based graph embedding with random walks, it only considers a node's local neighbours within the same path and thus overlooks the global structure information. Moreover, it is difficult to find an "optimal" sampling strategy as the embedding and path sampling are not jointly optimized in a unified framework. On the other hand, for DL based graph embedding without random walks, the computational cost is usually high. The traditional deep learning architectures assume the input data on 1D or 2D grid to take advantage of GPU [112]. However, graphs do not have such a grid structure, and thus require different solutions to improve the efficiency [112].

Edge reconstruction based graph embedding optimizes an objective function based on the observed edges or ranking triplets. It is more efficient compared to the previous two categories of graph embedding. However, this line of methods are trained using the directly observed local information, and thus the obtained embedding lacks the awareness of the global graph structure.

Graph kernel based graph embedding converts a graph into one single vector to facilitate the graph level analytic tasks such as graph classification. It is more efficient than other categories of techniques as it only needs to enumerate the desired atomic substructures in a graph. However, such "bag-of-structure" based methods have two limitations [88]. Firstly, the substructures are not independent. For example, the graphlet of size $k+1$ can be derived from size k graphlet by adding a new node and some edges. This means there exist redundant information in the graph representation. Secondly, the embedding dimensionality usually grows exponentially when the size of the substructure grows, leading to a sparse problem in the embedding.

Generative model based graph embedding can naturally leverage information from different sources (e.g., graph structure, node attribute) in a unified model. Directly embedding graphs into the latent semantic space generates the embedding that can be interpreted using the semantics. But the assumption of modelling the observation using certain distributions is hard to justify. Moreover, the generative method needs a large amount of training data to estimate a proper model which fits the data. Hence it may not work well for small graphs or a few graphs.

5 Applications

Graph embedding benefits a wide variety of graph analytics application as the vector representations can be processed

efficiently in both time and space. In this section, we categorize the graph embedding enabled applications as node related, edge related and graph related.

5.1 Node Related Applications

5.1.1 Node Classification

Node classification is to assign a class label to each node in a graph based on the rules learnt from the labelled nodes. Intuitively, "similar" nodes have the same labels. It is one of the most common application discussed in graph embedding literatures. In general, each node is embedded as a low-dimensional vector. Node classification is conducted by applying a classifier on the set of labelled node embedding for training. The example classifiers include SVM ([1], [19], [32], [33], [40], [41], [45], [56], [57], [60], [72], [74], [83], [97]), logistic regression ([1], [16], [18], [19], [20], [24], [26], [27], [45], [59], [79]) and k-nearest neighbour classification ([58], [145]). Then given the embedding of an unlabelled node, the trained classifier can predict its class label. In contrast to the above sequential processing of first node embedding then node classification, some other work ([47], [48], [62], [71], [78]) designs a unified framework to jointly optimize graph embedding and node classification, which learns a classification-specific representation for each node.

5.1.2 Node Clustering

Node clustering aims to group similar nodes together, so that nodes in the same group are more similar to each other than those in other groups. As an unsupervised algorithm, it is applicable when the node labels are unavailable. After representing nodes as vectors, the traditional clustering algorithms can then be applied on the node embedding. Most of existing work [1], [2], [20], [21], [22], [32], [97] adopts k-means as the clustering algorithm. In contrast, [4] and [76] jointly optimize clustering and graph embedding in one objective, and learn a clustering-specific node representation.

5.1.3 Node Recommendation/Retrieval/Ranking

The task of node recommendation is to recommend top K nodes of interest to a given node based on certain criteria such as similarity [3], [15], [42], [45], [47], [101]. In real-world scenarios, there are various types of recommended node, such as research interests for researchers [65], items for customers [3], [70], images for curation network users [34], friends for social network users [3], and documents for a query [68]. It is also popular in community-based question answering. Given a question, they predict the relative rank of users ([29], [30]) or answers ([28], [31]). In proximity search [38], [44], they rank the nodes of a particular type (e.g., "user") for a given query node (e.g., "Bob") and a proximity class (e.g., "schoolmate"), e.g., ranking users whom are the schoolmates of Bob. And there is some work focusing on cross-modal retrieval [32], [33], [35], [94], e.g., keyword-based image/video search.

A specific application which is popularly discussed in knowledge graph embedding area is entity ranking [51], [52], [53], [59], [61] . Recall that a knowledge graph consists of a set of triplets $< h, r, t >$. Entity ranking aims to rank the correct missing entities given the other two components in a triplet higher than the false entities. E.g., it returns the true h's among all the candidate entities given r and t, or returns the true t's given r and h.

5.2 Edge Related Applications

Next we introduce edge related applications in which an edge or a node pair is involved.

5.2.1 Link Prediction and Graph Reconstruction

Link prediction is one of the most popular edge related applications [3], [4], [15], [18], [23], [27], [58], [69]. It aims to predict the existence of a link between two nodes. Intuitively, the more similar two nodes are, the more likely there is an edge between them. Node similarity can be easily calculated based on two node embedding vectors.

Similarly, graph reconstruction [19], [32], [101] aims to reconstruct a graph by predicting links among nodes. The differences between link prediction and graph reconstruction are the training data and the evaluation metrics. For link prediction, a portion of the edges are removed from the graph for testing and the rest of the graph is used for training. During the test, a set of negative edges are sampled. The performance of link prediction is evaluated by AUC, which checks whether the true edges are ranked higher than the negative ones given two end nodes. For graph reconstruction, the whole graph is used for training. Then in the test phase, all the edges are removed from the graph, and the edges are predicted for all the node pairs in the graph. The original graph is served as the ground-truth. The predicted links are ranked by their existence probability. Graph construction is evaluated by precision@K and MAP, which checks the ratio of real links in the top K ranked links.

5.2.2 Triple Classification

Triplet classification [13], [14], [37], [51], [52], [53], [61], [137] is a specific application for knowledge graph. It aims to classify whether an unseen triplet $< h, r, t >$ is correct or not, i.e., whether the relation between h and t is r.

5.3 Graph Related Applications

5.3.1 Graph Classification

Graph classification assigns a class label to a whole graph. This is important when the graph is the unit of data. For example, in [50], each graph is a chemical compound, an organic molecule or a protein structure. In most cases, whole-graph embedding is applied to calculate graph level similarity [49], [54], [55], [88], [91]. Recently, some work starts to match node embedding for graph similarity [50], [146]. Each graph is represented as a set of node embedding vectors. Graphs are compared based on two sets of node embedding. [88] decomposes a graph into a set of substructures and then embed each substructure as a vector and compare graphs via the substructure similarities.

5.3.2 Visualization

Graph visualization generates visualizations of a graph on a low dimensional space [19], [22], [48], [55], [58], [72]. Usually, for visualization purpose, all nodes are embedded as 2D vectors and then plotted in a 2D space with different colours indicating nodes' categories. It provides a vivid demonstration of whether nodes belonging to the same category are embedded closer to each other.

5.4 Other Applications

Above are some general applications that are commonly discussed in existing work. Depending on the information carried in the input graph, more specific applications may exist. Below are some example scenarios.

Knowledge graph related: [14] and [13] extract relational fact from large-scale plain text. [62] extracts medical entities from text. [68] links natural language text with entities in a knowledge graph. In [87], the authors focus on deduplicating entities that are equivalent in a knowledge graph. [80] jointly embeds entity mentions, text features and entity types and then use them to estimate the true type-path for each mentions from its noisy candidate type set. For example, the candidate type set for "Trump" is {person, politician, businessman, artist, actor}. For the mention "Trump" in sentence "Republican residential candidate Donald Trump spoke during a campaign event in Rock Hill.", only {person, politician} are correct types.

Multimedia network related: [43] embeds the geotagged social media (GTSM) records <time, location, message> which enables them to recover the missing component from a GTSM triplet given the other two. It can also classify the GTSM records, e.g., whether a check-in record is related to "Food" or "Shop". [81] uses graph embedding to reduce data dimensionality for face recognition. [84] maps images into a semantic manifold that faithfully grasps users' preferences to facilitate content-based image retrieval.

Information propagation related: [63] predicts the increment of a cascade size after a given time interval. [64] predicts the propagation user and identifies the domain expert by embedding the social interaction graph.

Social networks alignment: Both [25] and [17] learn node embedding to align users across different social networks, i.e., to predict whether two user accounts in two different social networks are owned by the same user.

6 Future Directions

In this section, we summarize four future directions for the field of graph embedding, including computation efficiency, problem settings, techniques and application scenarios.

Computation. The deep architecture, which takes the geometric input (e.g., graph), suffers the low efficiency problem. Traditional deep learning models (designed for Euclidean domains) utilize the modern GPU to optimize their efficiency by assuming the input data on a 1D or 2D grid. However, graphs do not have such a kind of grid structure and thus the deep architecture designed for graph embedding needs to seek alternative solutions to improve the model efficiency. [112] suggested that the computational paradigms developed for large-scale graph processing can be adopted to facilitate the efficiency improvement in deep learning models for graph embedding.

Problem settings. The dynamic graph is one promising setting for graph embedding. Graphs are not always static, especially in real life scenarios, e.g., social graphs in Twitter, citation graphs in DBLP. Graphs can be dynamic in terms of graph structure or node/edge information. On the one hand, the graph structure may evolve with time, i.e., new nodes/edges are coming while some old nodes/edges disappear. On the other hand, the nodes/edges may be described by some time-varying information. Existing graph embedding mainly focuses on embedding the static graph and the settings of dynamic graph embedding are overlooked. Unlike static graph embedding, the techniques for dynamic graphs need to be scalable and better to be incremental so as to deal with the dynamic changes efficiently. This makes most of the existing graph embedding methods, which suffer from the low efficiency problem, not suitable anymore. How to design effective graph embedding methods in dynamic domains remains an open question.

Techniques. Structure awareness is important for edge reconstruction based graph embedding. Current edge reconstruction based graph embedding methods are mainly based on the edges only, e.g., 1-hope neighbours in a general graph, a ranked triplet ((h, r, t) in knowledge graph, and (v_i, v_i^+, v_i^-) in cQA graph. Single edges only provide local neighbourhood information to calculate the first- and second-order proximity. The global structure of a graph (e.g., paths, tree, subgraph patterns) is omitted. Intuitively, a substructure contains richer information than one single edge. A few work has attempted to explore the path information in knowledge graph embedding ([37], [38], [39], [137]). However, most of them use deep learning models ([37], [39], [137]) which suffer the low efficiency issue as discussed earlier. How to design the non-deep learning based methods that can take advantage of the expressive power of graph structure is a question. [38] provides one example solution. It minimizes both pairwise and long-range loss to capture both pairwise relations and long-range interactions between entities. Note that in addition to the list/path structure, there are various kinds of substructures which carry different structure information. An efficient structure-aware graph embedding optimization solution, together with the substructure sampling strategy, is in need.

Applications. Graph embedding has been adopted in many different applications. It is an effective way to learn the representations of the data with the consideration of their relations. Moreover, it can convert data instances from different sources/platforms/views into one common space so that they are directly comparable. For example, [15], [33], [35] use graph embedding for cross-modal retrieval, such as content-based image retrieval, keyword-based image/video search. The advantages of using graph embedding for representation learning is that the graph manifold of the training data instances are preserved in the representations and can further benefit the follow-up applications. Consequently, graph embedding can benefit the tasks which assume the input data instances are correlated with certain relations (i.e., connected by certain links). It is of great importance to exploring the application scenarios which benefit from graph embedding, as it provides effective solutions to the conventional problems from a different perspective.

7 Conclusion

In this survey, we conduct a comprehensive review of the literature in graph embedding. We provide a formal definition to the problem of graph embedding and introduce some basic concepts. More importantly, we propose two taxonomies

of graph embedding, categorizing existing work based on problem settings and embedding techniques respectively. In terms of problem setting taxonomy, we introduce four types of embedding input and four types of embedding output and summarize the challenges faced in each setting. For embedding technique taxonomy, we introduce the work in each category and compare them in terms of their advantages and disadvantages. After that, we summarize the applications that graph embedding enables. And finally, we suggest four promising future research directions in the field of graph embedding in terms of computation efficiency, problem settings, techniques and application scenarios.

REFERENCES

[1] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *AAAI*, 2017, pp. 203–209.

[2] F. Nie, W. Zhu, and X. Li, "Unsupervised large graph embedding," in *AAAI*, 2017, pp. 2422–2428.

[3] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, "Scalable graph embedding for asymmetric proximity," in *AAAI*, 2017, pp. 2942–2948.

[4] X. Wei, L. Xu, B. Cao, and P. S. Yu, "Cross view link prediction by learning noise-resilient representation consensus," in *WWW*, 2017, pp. 1611–1619.

[5] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," in *OSDI*, 2014, pp. 599–613.

[6] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, 2012.

[7] P. Kumar and H. H. Huang, "G-store: High-performance graph store for trillion-edge processing," in *SC*, 2016, pp. 71:1–71:12.

[8] N. Satish, N. Sundaram, M. M. A. Patwary, J. Seo, J. Park, M. A. Hassaan, S. Sengupta, Z. Yin, and P. Dubey, "Navigating the maze of graph analytics frameworks using massive graph datasets," in *SIGMOD*, 2014, pp. 979–990.

[9] Y. Bengio, A. C. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *PAMI*, vol. 35, no. 8, pp. 1798–1828, 2013.

[10] A. Mahmood, M. Small, S. A. Al-Máadeed, and N. M. Rajpoot, "Using geodesic space density gradients for network community detection," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 4, pp. 921–935, 2017.

[11] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *CoRR*, vol. abs/1705.02801, 2017.

[12] N. S. S and S. Surendran, "Graph embedding and dimensionality reduction-a survey," *IJCSET*, vol. 4, no. 1, pp. 29–34, 2013.

[13] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *AAAI*, 2014, pp. 1112–1119.

[14] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *AAAI*, 2015, pp. 2181–2187.

[15] X. Zhao, A. Chang, A. D. Sarma, H. Zheng, and B. Y. Zhao, "On the embeddability of random walk distances," *PVLDB*, vol. 6, no. 14, pp. 1690–1701, 2013.

[16] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.

[17] T. Man, H. Shen, S. Liu, X. Jin, and X. Cheng, "Predict anchor links across social networks via an embedding approach," in *IJCAI*, 2016, pp. 1823–1829.

[18] T. Pimentel, A. Veloso, and N. Ziviani, "Unsupervised and scalable algorithm for learning node representations," in *ICLR*, 2017.

[19] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, 2016, pp. 1225–1234.

[20] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *CIKM*, 2015, pp. 891–900.

[21] F. Tian, B. Gao, Q. Cui, E. Chen, and T. Liu, "Learning deep representations for graph clustering," in *AAAI*, 2014, pp. 1293–1299.

[22] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *AAAI*, 2016, pp. 1145–1152.

[23] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *WWW*, 2013, pp. 37–48.

[24] Z. Jin, R. Liu, Q. Li, D. D. Zeng, Y. Zhan, and L. Wang, "Predicting user's multi-interests with network embedding in health-related topics," in *IJCNN*, 2016, pp. 2568–2575.

[25] L. Liu, W. K. Cheung, X. Li, and L. Liao, "Aligning users across social networks using network embedding," in *IJCAI*, 2016, pp. 1774–1780.

[26] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[27] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[28] H. Fang, F. Wu, Z. Zhao, X. Duan, Y. Zhuang, and M. Ester, "Community-based question answering via heterogeneous social network learning," in *AAAI*, 2016, pp. 122–128.

[29] Z. Zhao, Q. Yang, D. Cai, X. He, and Y. Zhuang, "Expert finding for community-based question answering via ranking metric network learning," in *IJCAI*, 2016, pp. 3000–3006.

[30] H. Lu and M. Kong, "Community-based question answering via contextual ranking metric network learning," in *AAAI*, 2017, pp. 4963–4964.

[31] Z. Zhao, H. Lu, V. W. Zheng, D. Cai, X. He, and Y. Zhuang, "Community-based question answering via asymmetric multi-faceted ranking network learning," in *AAAI*, 2017, pp. 3532–3539.

[32] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *KDD*, 2015, pp. 119–128.

[33] H. Zhang, X. Shang, H. Luan, M. Wang, and T. Chua, "Learning from collective intelligence: Feature learning using social images and tags," *TOMCCAP*, vol. 13, no. 1, pp. 1:1–1:23, 2016.

[34] X. Geng, H. Zhang, J. Bian, and T. Chua, "Learning image and user features for recommendation in social networks," in *ICCV*, 2015, pp. 4274–4282.

[35] F. Wu, X. Lu, J. Song, S. Yan, Z. M. Zhang, Y. Rui, and Y. Zhuang, "Learning of multimodal representations with random walks on the click graph," *IEEE Trans. Image Processing*, vol. 25, no. 2, pp. 630–642, 2016.

[36] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *SIGMOD*, 2008, pp. 1247–1250.

[37] J. Feng, M. Huang, Y. Yang, and X. Zhu, "GAKE: graph aware knowledge embedding," in *COLING*, 2016, pp. 641–651.

[38] F. Wu, J. Song, Y. Yang, X. Li, Z. M. Zhang, and Y. Zhuang, "Structured embedding via pairwise relations and long-range interactions in knowledge base," in *AAAI*, 2015, pp. 1663–1670.

[39] B. Shi and T. Weninger, "Proje: Embedding projection for knowledge graph completion," in *AAAI*, 2017, pp. 1236–1242.

[40] M. Ochi, Y. Nakashio, Y. Yamashita, I. Sakata, K. Asatani, M. Ruttley, and J. Mori, "Representation learning for geospatial areas using large-scale mobility data from smart card," in *UbiComp*, 2016, pp. 1381–1389.

[41] M. Ochi, Y. Nakashio, M. Ruttley, J. Mori, and I. Sakata, "Geospatial area embedding based on the movement purpose hypothesis using large-scale mobility data from smart card," *IJCNS*, vol. 9, pp. 519–534, 2016.

[42] Y. Zhao, Z. Liu, and M. Sun, "Representation learning for measuring entity relatedness with rich information," in *IJCAI*, 2015, pp. 1412–1418.

[43] C. Zhang, K. Zhang, Q. Yuan, H. Peng, Y. Zheng, T. Hanratty, S. Wang, and J. Han, "Regions, periods, activities: Uncovering urban dynamics via cross-modal representation learning," in *WWW*, 2017, pp. 361–370.

[44] Z. Liu, V. W. Zheng, Z. Zhao, F. Zhu, K. C. Chang, M. Wu, and J. Ying, "Semantic proximity search on heterogeneous graph by proximity embedding," in *AAAI*, 2017, pp. 154–160.

[45] H. Gui, J. Liu, F. Tao, M. Jiang, B. Norick, and J. Han, "Large-scale embedding learning in heterogeneous event data," in *ICDM*, 2016, pp. 907–912.

[46] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *KDD*, 2017, pp. 135–144.

[47] J. Li, J. Zhu, and B. Zhang, "Discriminative deep random walk for network classification," in *ACL*, 2016.

[48] C. Tu, W. Zhang, Z. Liu, and M. Sun, "Max-margin deepwalk: Discriminative learning of network representation," in *IJCAI*, 2016, pp. 3889–3895.

[49] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.

[50] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," in *AAAI*, 2017, pp. 2429–2435.

[51] S. Guo, Q. Wang, B. Wang, L. Wang, and L. Guo, "Semantically smooth knowledge graph embedding," in *ACL*, 2015, pp. 84–94.

[52] ——, "SSE: semantically smooth embedding for knowledge graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 4, pp. 884–897, 2017.

[53] R. Xie, Z. Liu, and M. Sun, "Representation learning of knowledge graphs with hierarchical types," in *IJCAI*, 2016, pp. 2965–2971.

[54] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *ICML*, 2016, pp. 2702–2711.

[55] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *ICML*, vol. 48, 2016, pp. 2014–2023.

[56] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," in *IJCAI*, 2015, pp. 2111–2117.

[57] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Homophily, structure, and content augmented network representation learning," in *ICDM*, 2016, pp. 609–618.

[58] T. M. V. Le and H. W. Lauw, "Probabilistic latent document network embedding," in *ICDM*, 2014, pp. 270–279.

[59] H. Xiao, M. Huang, L. Meng, and X. Zhu, "SSP: semantic space projection for knowledge graph embedding with text descriptions," in *AAAI*, 2017, pp. 3104–3110.

[60] L. Yao, Y. Zhang, B. Wei, Z. Jin, R. Zhang, Y. Zhang, and Q. Chen, "Incorporating knowledge graph embeddings into topic modeling," in *AAAI*, 2017, pp. 3119–3126.

[61] Z. Wang and J. Li, "Text-enhanced representation learning for knowledge graph," in *IJCAI*, 2016, pp. 1293–1299.

[62] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *ICML*, 2016, pp. 40–48.

[63] C. Li, J. Ma, X. Guo, and Q. Mei, "Deepcas: An end-to-end predictor of information cascades," in *WWW*, 2017, pp. 577–586.

[64] N. Zhao, H. Zhang, M. Wang, R. Hong, and T. Chua, "Learning content-social influential features for influence analysis," *IJMIR*, vol. 5, no. 3, pp. 137–149, 2016.

[65] Z. Yang, J. Tang, and W. Cohen, "Multi-modal bayesian embeddings for learning social knowledge graphs," in *IJCAI*, 2016, pp. 2287–2293.

[66] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *WWW*, 2007, pp. 697–706.

[67] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "Dbpedia - A crystallization point for the web of data," *J. Web Sem.*, vol. 7, no. 3, pp. 154–165, 2009.

[68] C. Xiong, R. Power, and J. Callan, "Explicit semantic ranking for academic search via knowledge graph embedding," in *WWW*, 2017, pp. 1271–1279.

[69] B. Alharbi and X. Zhang, "Learning from your network of friends: A trajectory representation learning model based on online social ties," in *ICDM*, 2016, pp. 781–786.

[70] Q. Zhang and H. Wang, "Not all links are created equal: An adaptive embedding approach for social personalized ranking," in *SIGIR*, 2016, pp. 917–920.

[71] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[72] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *IJCAI*, 2016, pp. 1895–1901.

[73] T. Hofmann and J. M. Buhmann, "Multidimensional scaling and data clustering," in *NIPS*, 1994, pp. 459–466.

[74] Y. Han and Y. Shen, "Partially supervised graph embedding for positive unlabelled feature selection," in *IJCAI*, 2016, pp. 1548–1554.

[75] M. Yin, J. Gao, and Z. Lin, "Laplacian regularized low-rank representation and its applications," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 3, pp. 504–517, 2016.

[76] M. Tang, F. Nie, and R. Jain, "Capped lp-norm graph embedding for photo clustering," in *MM*, 2016, pp. 431–435.

[77] M. Balasubramanian and E. L. Schwartz, "The isomap algorithm and topological stability," *Science*, vol. 295, no. 5552, pp. 7–7, 2002.

[78] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *CVPR*, 2017.

[79] J. Tang, M. Qu, and Q. Mei, "Pte: Predictive text embedding through large-scale heterogeneous text networks," in *KDD*, 2015, pp. 1165–1174.

[80] X. Ren, W. He, M. Qu, C. R. Voss, H. Ji, and J. Han, "Label noise reduction in entity typing by heterogeneous partial-label embedding," in *KDD*, 2016, pp. 1825–1834.

[81] S. Yan, D. Xu, B. Zhang, H. Zhang, Q. Yang, and S. Lin, "Graph embedding and extensions: A general framework for dimensionality reduction," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 1, pp. 40–51, 2007.

[82] C. Gong, D. Tao, J. Yang, and K. Fu, "Signed laplacian embedding for supervised dimension reduction," in *AAAI*, 2014, pp. 1847–1853.

[83] L. Sun, S. Ji, and J. Ye, "Hypergraph spectral learning for multi-label classification," in *KDD*, 2008, pp. 668–676.

[84] Y.-Y. Lin, T.-L. Liu, and H.-T. Chen, "Semantic manifold learning for image retrieval," in *MM*, 2005, pp. 249–258.

[85] A. Bordes, J. Weston, R. Collobert, and Y. Bengio, "Learning structured embeddings of knowledge bases," in *AAAI*, 2011.

[86] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NIPS*, 2013, pp. 2787–2795.

[87] A. Bordes, X. Glorot, J. Weston, and Y. Bengio, "A semantic matching energy function for learning with multi-relational data - application to word-sense disambiguation," *Machine Learning*, vol. 94, no. 2, pp. 233–259, 2014.

[88] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *KDD*, 2015, pp. 1365–1374.

[89] A. Bordes, S. Chopra, and J. Weston, "Question answering with subgraph embeddings," in *EMNLP*, 2014, pp. 615–620.

[90] V. W. Zheng, S. Cavallari, H. Cai, K. C. Chang, and E. Cambria, "From node embedding to community embedding," *CoRR*, vol. abs/1610.09950, 2016.

[91] S. F. Mousavi, M. Safayani, A. Mirzaei, and H. Bahonar, "Hierarchical graph embedding in vector space by graph pyramid," *Pattern Recognition*, vol. 61, pp. 245–254, 2017.

[92] W. N. A. Jr. and T. D. Morley, "Eigenvalues of the laplacian of a graph," *Linear and Multilinear Algebra*, vol. 18, no. 2, pp. 141–145, 1985.

[93] X. He and P. Niyogi, "Locality preserving projections," in *NIPS*, 2003, pp. 153–160.

[94] D. Cai, X. He, and J. Han, "Spectral regression: a unified subspace learning framework for content-based image retrieval," in *MM*, 2007, pp. 403–412.

[95] K. Q. Weinberger, F. Sha, and L. K. Saul, "Learning a kernel matrix for nonlinear dimensionality reduction," in *ICML*, 2004.

[96] K. Allab, L. Labiod, and M. Nadif, "A semi-nmf-pca unified framework for data clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 1, pp. 2–16, 2017.

[97] M. Chen, I. W. Tsang, M. Tan, and C. T. Jen, "A unified feature selection framework for graph embedding on high dimensional data," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 6, pp. 1465–1477, 2015.

[98] S. T. Roweis and L. K. Saul, "Nonlinear Dimensionality Reduction by Locally Linear Embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[99] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM Rev.*, vol. 38, no. 1, pp. 49–95, 1996.

[100] B. Shaw and T. Jebara, "Structure preserving embedding," in *ICML*, 2009, pp. 937–944.

[101] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *KDD*, 2016, pp. 1105–1114.

[102] B. Shaw and T. Jebara, "Minimum volume embedding," in *AISTATS*, 2007, pp. 460–467.

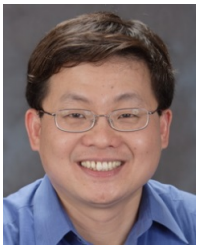
[103] M. Nickel, V. Tresp, and H. peter Kriegel, "A three-way model for collective learning on multi-relational data," in *ICML*. ACM, 2011, pp. 809–816.

[104] J. H. Tianji Pang, Feiping Nie, "Flexible orthogonal neighborhood preserving embedding," in *IJCAI-17*, 2017, pp. 2592–2598.

[105] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numer. Math.*, vol. 14, no. 5, pp. 403–420, 1970.
[106] X. Z. C. Z. Daokun Zhang, Jie Yin, "User profile preserving social network embedding," in *IJCAI*, 2017, pp. 3378–3384.
[107] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *AISTATS*, 2009, pp. 488–495.
[108] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.
[109] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.
[110] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[111] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014.
[112] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *CoRR*, vol. abs/1611.08097, 2016.
[113] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *ICLR*, 2013.
[114] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *CoRR*, vol. abs/1506.05163, 2015.
[115] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016, pp. 3837–3845.
[116] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.
[117] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *NIPS*, 2015, pp. 2224–2232.
[118] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *ICLR*, 2016.
[119] R. Khasanova and P. Frossard, "Graph-based isometry invariant representation learning," in *ICML*, 2017, pp. 1847–1856.
[120] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, "Knowledge graph embedding via dynamic mapping matrix," in *ACL*, 2015, pp. 687–696.
[121] G. Ji, K. Liu, S. He, and J. Zhao, "Knowledge graph completion with adaptive sparse transfer matrix," in *AAAI*, 2016, pp. 985–991.
[122] J. Wen, J. Li, Y. Mao, S. Chen, and R. Zhang, "On the representation and embedding of knowledge bases beyond binary relations," in *IJCAI*, 2016, pp. 1300–1307.
[123] R. Xie, Z. Liu, J. Jia, H. Luan, and M. Sun, "Representation learning of knowledge graphs with entity descriptions," in *AAAI*, 2016, pp. 2659–2665.
[124] H. Xiao, M. Huang, and X. Zhu, "From one point to a manifold: Knowledge graph embedding for precise link prediction," in *IJCAI*, 2016, pp. 1315–1321.
[125] Y. Jia, Y. Wang, H. Lin, X. Jin, and X. Cheng, "Locally adaptive translation for knowledge graph embedding," in *AAAI*, 2016, pp. 992–998.
[126] R. Socher, D. Chen, C. D. Manning, and A. Y. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *NIPS*, 2013, pp. 926–934.
[127] M. Nickel, L. Rosasco, and T. Poggio, "Holographic embeddings of knowledge graphs," in *AAAI*, 2016, pp. 1955–1961.
[128] M. Y. C. Z. Muhao Chen, Yingtao Tian, "Multilingual knowledge graph embeddings for cross-lingual knowledge alignment," in *IJCAI*, 2017, pp. 1511–1517.
[129] H. Liu, Y. Wu, and Y. Yang, "Analogical inference for multi-relational embeddings," in *ICML*, 2017, pp. 2168–2178.
[130] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *ICML*, 2016, pp. 2071–2080.
[131] D. Haussler, "Convolution kernels on discrete structures," University of California at Santa Cruz, Technical Report UCS-CRL-99-10, 1999.
[132] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.
[133] K. M. Borgwardt and H. Kriegel, "Shortest-path kernels on graphs," in *ICDM*, 2005, pp. 74–81.
[134] C. M. Bishop and J. Lasserre, "Generative or Discriminative? Getting the Best of Both Worlds," in *Bayesian Statistics 8*, 2007, pp. 3–24.
[135] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," in *NIPS*, 2001, pp. 601–608.
[136] H. Xiao, M. Huang, and X. Zhu, "Transg : A generative model for knowledge graph embedding," in *ACL*, 2016.
[137] Y. Luo, Q. Wang, B. Wang, and L. Guo, "Context-dependent knowledge graph embedding," in *EMNLP*, 2015, pp. 1656–1661.
[138] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W. Ma, "Collaborative knowledge base embedding for recommender systems," in *KDD*, 2016, pp. 353–362.
[139] Z. L. C. T. Cheng Yang, Maosong Sun, "Fast network embedding enhancement via high order proximity approximation," in *IJCAI*, 2017, pp. 3894–3900.
[140] Y. Lin, Z. Liu, and M. Sun, "Knowledge representation learning with entities, attributes and relations," in *IJCAI*, 2016, pp. 2866–2872.
[141] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "Struc2vec: Learning node representations from structural identity," in *KDD*, 2017, pp. 385–394.
[142] J. A. Bullinaria and J. P. Levy, "Extracting semantic representations from word co-occurrence statistics: stop-lists, stemming, and svd," *Behavior Research Methods*, vol. 44, no. 3, pp. 890–907, 2012.
[143] O. Levy, Y. Goldberg, and I. Dagan, "Improving distributional similarity with lessons learned from word embeddings," *TACL*, vol. 3, pp. 211–225, 2015.
[144] J. Demmel, I. Dumitriu, and O. Holtz, "Fast linear algebra is stable," *Numerische Mathematik*, vol. 108, no. 1, pp. 59–91, 2007.
[145] R. C. Wilson, E. R. Hancock, E. Pekalska, and R. P. W. Duin, "Spherical and hyperbolic embeddings of data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2255–2269, 2014.
[146] F. D. Johansson and D. P. Dubhashi, "Learning with similarity functions on graphs using matchings of geometric embeddings," in *KDD*, 2015, pp. 467–476.

Hongyun Cai received the Ph.D. degree in Computer Science from the University of Queensland in 2016. She is currently a post-doctoral researcher at the Advanced Digital Sciences Center, Singapore. Her research focuses on graph mining, social data management and analysis.

Vincent W. Zheng received the Ph.D. degree in Computer Science from the Hong Kong University of Science and Technology in 2011. He is a research scientist at the Advanced Digital Sciences Center, Singapore, and a research affiliate at the University of Illinois at Urbana-Champaign. He research interests include graph mining, information extraction, ubiquitous computing and machine learning.



Kevin Chen-Chuan Chang is a Professor in the Department of Computer Science, University of Illinois at Urbana-Champaign. His research addresses large-scale information access, for search, mining, and integration across structured and unstructured big data including Web data and social media. He also co-founded Cazoodle for deepening vertical data-aware search over the Web.