

DeepNC: Deep Generative Network Completion

Cong Tran, *Student Member, IEEE*, Won-Yong Shin, *Senior Member, IEEE*, Andreas Spitz,
and Michael Gertz

Abstract—Most network data are collected from partially observable networks with both missing nodes and missing edges, for example, due to limited resources and privacy settings specified by users on social media. Thus, it stands to reason that inferring the missing parts of the networks by performing *network completion* should precede downstream applications. However, despite this need, the recovery of missing nodes and edges in such incomplete networks is an insufficiently explored problem due to the modeling difficulty, which is much more challenging than link prediction that only infers missing edges. In this paper, we present DeepNC, a novel method for inferring the missing parts of a network based on a *deep generative* model of graphs. Specifically, our method first learns a likelihood over edges via an *autoregressive generative* model, and then identifies the graph that maximizes the learned likelihood conditioned on the observable graph topology. Moreover, we propose a computationally efficient DeepNC algorithm that *consecutively* finds individual nodes that maximize the probability in each node generation step, as well as an enhanced version using the expectation-maximization algorithm. The runtime complexities of both algorithms are shown to be *almost linear* in the number of nodes in the network. We empirically demonstrate the superiority of DeepNC over state-of-the-art network completion approaches.

Index Terms—Autoregressive generative model; deep generative model of graphs; inference; network completion; partially observable network

1 INTRODUCTION

1.1 Background and Motivation

Real-world networks extracted from various biological, social, technological, and information systems tend to be only partially observable with missing both nodes and edges [1]. For example, users and organizations may have limited access to data due to insufficient resources or a lack of authority. In social networks, a source of incompleteness stems from privacy settings specified by users who partially or completely hide their identities and/or friendships [2]. As an example, consider a demographic analysis of Facebook users in New York City in June 2011 that showed 52.6% of the users to be hiding their Facebook friends [3]. Using such incomplete network data may severely degrade the performance of downstream analyses such as community detection, link prediction, and node classification due to significantly altered estimates of structural properties (see, e.g., [1], [4], [5], [6] and references therein).

This motivates us to conduct *network completion* to infer the missing part (i.e., a set of both missing nodes and associated edges), prior to performing downstream appli-

cations. While intuitively similar, network completion is a much more challenging task than the well-studied *link prediction* and *low-rank matrix completion*, since it *jointly infers both missing nodes and edges*, while link prediction and matrix completion only infer missing edges. Although one prior contribution has attempted to address the recovery of missing nodes and edges with an algorithm, dubbed KronEM, that infers the missing parts of a graph based on the Kronecker graph model [5], this current state-of-the-art model suffers from three major problems: 1) setting the size of a Kronecker generative parameter is not trivial; 2) the Kronecker graph model is inherently designed under the assumption of a pure power-law degree distribution that not all real-world networks necessarily follow; and 3) its inference accuracy is not satisfactory.

As a way of further enhancing the performance of network completion, our study is intuitively motivated by the existence of *structurally similar* graphs with respect to graph distance, whose topologies are almost entirely observable.¹ Such similar graphs can be retrieved from the same domain as that of the target graph (see [7], [8], [9] for more information). As an example, suppose that many citizens residing in country A strongly protect the privacy of their social relationships, while citizens of country B tend to provide their friendship relations on social media. Intuitively, as long as the graph structures between two countries are similar to each other, latent information within the (almost) complete data collected from country B can be uncovered and leveraged to infer the missing part of the collected data from country A. Additionally, the use of deep learning on graphs has been actively studied by exploiting this structural similarity of graphs (see, e.g., [10], [11] and references therein), which enables us to model complex

- C. Tran is with the Department of Computer Science and Engineering, Dankook University, Yongin 16890, Republic of Korea, and also with the Machine Intelligence & Data Science Laboratory, Yonsei University, Seoul 03722, Republic of Korea.
E-mail: congtran@ieee.org.
- W.-Y. Shin is with the School of Mathematics and Computing (Computational Science and Engineering), Yonsei University, Seoul 03722, Republic of Korea.
E-mail: wy.shin@yonsei.ac.kr.
- A. Spitz is with the School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne, Lausanne 1015, Switzerland.
E-mail: andreas.spitz@epfl.ch.
- M. Gertz is with the Institute of Computer Science, Heidelberg University, Heidelberg 69120, Germany.
E-mail: gertz@informatik.uni-heidelberg.de.
(Corresponding author: Won-Yong Shin.)

1. In the following, we use the terms “network” and “graph” interchangeably.

structures over graphs with high accuracy. For example, the framework of recurrent neural networks (RNN) and generative adversarial networks (GAN) were recently introduced to construct deep generative models of graphs [10], [11]. Thus, a natural question is how such *structural similarity* can be incorporated into the problem of network completion by taking advantage of effective deep learning-based approaches.

1.2 Main Contributions

In this paper, we introduce DeepNC, a novel method for completing the missing part of an observed incomplete network G_O based on a *deep generative* model of graphs. Specifically, we first learn a likelihood over edges (i.e., a latent representation) via an *autoregressive generative* model of graphs, e.g., GraphRNN [10] built upon RNN, by using a set of structurally similar graphs as training data, and then infer the missing part of the network. Unlike GraphRNN, which is only applicable to fully observable graphs, our method is capable of accommodating both observable and missing parts by imputing a number of missing nodes and edges with *sampled* values from a multivariate Bernoulli distribution. To this end, we formulate a new optimization problem with the aim of finding the graph that maximizes the learned likelihood conditioned on the observable graph topology. To efficiently solve the problem, we first propose a low-complexity DeepNC algorithm, termed DeepNC-L, that *consecutively* finds a single node maximizing the probability in each node generation step in a greedy fashion under the assumption that there are no missing edges between two nodes in a partially observable network G_O . We then present judicious approximation and computational reduction techniques to DeepNC-L by exploiting the *sparseness* of real-world networks. Second, by relaxing this assumption to deal with a more realistic scenario in which there are missing edges in G_O , we propose an enhanced version of DeepNC using the expectation-maximization (EM) algorithm, termed DeepNC-EM, which enables us to jointly find both missing edges between nodes in G_O and edges associated with missing nodes by executing DeepNC-L iteratively. That is, the DeepNC-EM algorithm jointly solves network completion and link prediction in a single module. We show that the computational complexity of both DeepNC algorithms is *almost linear* in the number of nodes in the network. By adopting the graph edit distance (GED) [12] as a performance metric, we empirically evaluate the performance of both DeepNC algorithms for various environments. Experimental results show that our algorithms consistently outperform state-of-the-art network completion approaches by up to 68.25% in terms of GED. The results also demonstrate the robustness of our method not only on various real-world networks that do not necessarily follow a power-law degree distribution, but also in three more difficult and challenging situations where 1) a large portion of nodes are missing, 2) training graphs are only partially observed, and 3) a large portion of edges between nodes in G_O are missing. Additionally, we analyze and empirically validate the computational complexity of DeepNC algorithms. Our main contributions are five-fold and summarized as follows:

TABLE 1: Summary of notations

Notation	Description
G_T	true graph
G_O	partially observable graph
V_O	set of nodes in G_O
E_O	set of edges in G_O
V_M	set of missing nodes
E_M	set of missing edges
G_I	training graph
p_{model}	probability distribution over edges of a graph
Θ	learned parameters of p_{model}
\hat{G}	recovered graph
π	node ordering
S^π	a sequence of nodes and edges under π

- We introduce DeepNC, a deep learning-based network completion method for partially observable networks;
- We formalize our problem as the imputation of missing data in an optimization problem that maximizes the conditional probability of a generated node sequence;
- We design two computationally efficient DeepNC algorithms to solve the problem by exploiting the sparsity of networks;²
- We validate DeepNC through extensive experiments using real-world datasets across various domains, as well as synthetic datasets;
- We analyze and empirically validate the computational complexity of DeepNC.

To the best of our knowledge, this study is the first work that applies deep learning to network completion.

1.3 Organization and Notations

The remainder of this paper is organized as follows. In Section 2, we summarize significant studies that are related to our work. In Section 3, we explain the methodology of our work, including the problem definition and an overview of our DeepNC method. Section 4 describes implementation details of the two DeepNC algorithms and analyzes their computational complexities. Experimental results are discussed in Section 5. Finally, we provide a summary and concluding remarks in Section 6.

Table 1 summarizes the notation that is used in this paper. This notation will be formally defined in the following sections when we introduce our methodology and the technical details.

2 RELATED WORK

The method that we propose in this paper is related to four broader areas of research, namely generative models of graphs, link prediction, low-rank matrix completion, and network completion.

²The source code used in this paper is available online (<https://github.com/congasix/DeepNC>).

TABLE 2: Summary of deep generative models of graphs

Deep generative models of graphs	Scalable	Flexible	Attributed
Autoregressive [10], [16]	✓	✓	
GAN [11], [20]	✓		
VAE [17], [18]			✓
Reinforcement learning [19]		✓	✓
General neural network [21]		✓	✓

Generative models of graphs. The study of generative models of graphs has a long history, beginning with the first random model of graphs that robustly assigns probabilities to large classes of graphs, and was introduced by Erdős and Rényi [13]. Another well-known model generates new nodes based on preferential attachment [14]. More recently, a generative model based on Kronecker graphs, the so-called KronFit [15], was introduced, which generates synthetic networks that match many of the structural properties of real-world networks such as constant and shrinking diameters. Recent advances in *deep learning*-based approaches have made further progress towards generative models for complex networks [10], [11], [16], [17], [18], [19], [20], [21]. GraphRNN [10] and graph recurrent attention networks (GRAN) [16] were presented to learn a distribution over edges by decomposing the graph generation process into sequences of node and edge formations via autoregressive generative models; an approach using the Wasserstein GAN objective in the training process was applied to generate discrete output samples [11]; variational autoencoders (VAEs) were employed to design another deep learning-based generative model of graphs [17], [18]; a graph convolutional policy network was presented for goal-directed graph generation (e.g., drug molecules) using reinforcement learning [19]; a multi-scale graph generative model, named Misc-GAN, was introduced by modeling the underlying distribution of graph structures at different levels of granularity to aim at generating graphs having similar community structures [20]; and a more general deep generative model was presented to learn distributions over any arbitrary graph via graph neural networks [21]. Among the aforementioned methods, autoregressive generative models such as GraphRNN and GRAN are the most scalable and flexible approaches in terms of graph size, while others are beneficial in generating non-topological information such as node attributes. Table 2 summarizes the literature overview of the aforementioned deep generative models of graphs.

Link prediction. Inferring the presence of links in a given network according to the neighborhood similarity of existing connections is a longstanding task in network science. Although numerous algorithms have been developed based on traditional statistical measures [22] and deep learning such as graph neural networks [18], [23], existing link prediction methods are not inherently designed to solve the network completion problem that jointly recovers missing nodes and edges in partially observable networks. Specifically, when a node is completely missing from the underlying network, link prediction models can no longer exploit structural neighborhood information.

Low-rank matrix completion. Missing entries in a low-rank matrix due to partial observations can be inferred by

solving the rank minimization problem using approximation methods such as singular value decomposition [24], matrix factorization [25], neural networks [26], and adaptive clustering of bandit strategies [27], [28]. Since many graphs tend to exhibit low-rank connectivity structures [29], several techniques used in matrix completion can also be applied to perform link prediction [30]. Similarly as in the setting of link prediction, low-rank matrix completion requires at least one entry in each row and column to be known in order to infer missing entries.

Network completion. Observing a partial sample of a network and inferring the remainder of the network is referred to as *network completion*. As the most influential prior work, KronEM, an approach based on Kronecker graphs to solving the network completion problem by applying the EM algorithm, was suggested by Kim and Leskovec [5]. MISC was developed to tackle the *missing node identification* problem when the information of connections between missing nodes and observable nodes is assumed to be available [31]. A follow-up study of MISC [32] incorporated metadata such as demographic information and the nodes' historical behavior into the inference process. Furthermore, a graph upscaling method, termed EvoGraph [33], can be regarded as a network completion method using a preferential attachment mechanism.

Discussion. Despite these contributions, no prior work in the literature exploits the power of deep generative models in the context of network completion. Although generative models of graphs such as GraphRNN can be used as a network completion method, nontrivial extra tasks are required, including computationally expensive *graph matching* to find the correspondence between generated graphs and the partially observable network. Furthermore, MISC and other follow-up studies do not truly address network completion, since they solve the *node identification* problem under the assumption that the connections between missing nodes and observable nodes are known beforehand, which is not feasible in a setting where only partial observation of nodes is possible as we address with DeepNC in the following.

3 METHODOLOGY

As a basis for the proposed DeepNC algorithm in Section 4, we first describe our network model with basic assumptions and formalize the problem. Then, we explain a deep generative graph model and our research methodology adopting the deep generative graph model to solve the problem of network completion.

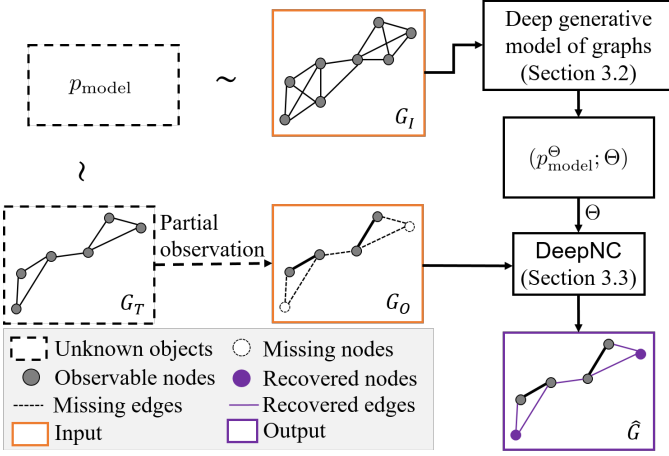


Fig. 1: The schematic overview of our DeepNC method.

3.1 Problem Definition

3.1.1 Network Model and Basic Assumptions

Let us denote a partially observable network as $G_O = (V_O, E_O)$, where V_O and E_O are the set of vertices and the set of edges, respectively. The network G_O with $|V_O|$ observable nodes can be interpreted as a subgraph taken from an underlying true network $G_T = (V_O \cup V_M, E_O \cup E_M)$, where V_M is the set of unobservable (missing) nodes and E_M is the set of three types of unobservable (missing) edges, including i) the edges connecting two nodes in V_M , ii) the edges connecting one node in V_O and another node in V_M , and iii) the missing edges connecting two nodes in V_O . More specifically, the set of observable edges, E_O , is regarded as a subset of all true edges connecting nodes in V_O . In contrast to the conventional setting that assumes no missing edges between two nodes in V_O [5], we relax this assumption by not requiring that G_O is a complete subgraph. In the following, we assume both G_O and G_T to be *undirected unweighted* networks without self-loops or repeated edges.

Let us denote p_{model} as a family of probability distributions over the edges of a graph, which can be parameterized by a set of model parameters Θ , i.e., $(p_{\text{model}}^\Theta; \Theta)$.³ In this paper, we assume that G_T is a sample drawn from the distribution p_{model} . Furthermore, we assume that the number of missing nodes, $|V_M|$, is available or can be estimated. In practice, $|V_M|$ can be readily estimated by standard statistical methods; for example, a latent non-random mixing model in [34] is capable of estimating a network size $|V_O \cup V_M|$ by asking respondents how many people they know in specific subpopulations. For an overview of network-relevant notations, see Fig. 1.

3.1.2 Problem Formulation

In the following, we formally define the network completion problem, the idea behind our approach, and the problem formulation.

Definition 1. Network completion problem. Given a partially observable network G_O , network completion aims to recover all missing edges connecting nodes in the true network G_T so

3. To simplify notations, p_{model}^Θ will be written as p_{model} if omitting Θ does not cause any confusion.

that the inferred network, denoted by \hat{G} , is equivalent to G_T (up to isomorphism).

As illustrated in Fig. 1, a network \hat{G} is inferred using the partially observable network G_O as input of DeepNC. We tackle this problem by minimizing a distance metric $\delta(G_T, \hat{G})$ that measures the difference between G_T and \hat{G} . Due to the fact that the true network G_T is not available, our intuition is to analyze the connectivity patterns of one (or multiple) fully observed network(s) G_I whose structure is similar to that of G_T (i.e., $\delta(G_T, G_I)$ is sufficiently small) and then to make use of this information for recovering the network G_O , where G_I is a sample drawn from the distribution p_{model} .⁴ To this end, we first learn $(p_{\text{model}}; \Theta)$ by using G_I as the *training* data under a deep generative model of graphs described in Section 3.2. Afterwards, we generate graphs with similar structures via the set of learned model parameters Θ . Among all generated graphs $G \in \mathcal{G}$ having $|V_O| + |V_M|$ nodes and containing a subgraph isomorphic to G_O , we find the most likely graph configuration \hat{G} from the distribution over graphs in the set \mathcal{G} parametrized by Θ . In this context, our optimization problem can be formulated as follows:

$$\begin{aligned} \hat{G} &= \arg \max_{G \in \mathcal{G}} p(G|G_O; \Theta) \\ \text{s.t. } |V_G| &= |V_O| + |V_M|, \end{aligned} \quad (1)$$

where $|V_G|$ denotes the number of nodes in G . The overall procedure of our approach is visualized in Fig. 1.

3.2 Deep Generative Model of Graphs

Deep generative models of graphs have the ability to approximate any distribution of graphs with minimal assumptions about their structures [10], [21]. Among recently introduced deep generative models, we adopt GraphRNN [10] in our study due to its state-of-the-art performance in generating diverse graphs that match the structural characteristics of a target set as well as the scalability to much larger graphs than those from other deep generative models (refer to Section 4 and Corollary 1 in [10] for more details). In this subsection, we briefly describe a variant of GraphRNN, termed simplified GraphRNN (GraphRNN-S), where the probability of edge connections for a node is assumed to be independent of each other. This method effectively learns $(p_{\text{model}}; \Theta)$ from the set of *structurally similar* network(s) G_I .

We first describe how to vectorize a graph. Given a graph G sampled from the distribution p_{model} with a number of nodes equal to $|V_O| + |V_M|$, we define a node ordering π that maps nodes to rows or columns of a given adjacency matrix of G as a permutation function over the set of nodes. Thus, $\{\pi(v_1), \dots, \pi(v_{|V_O|+|V_M|})\}$ is a permutation of $\{v_1, \dots, v_{|V_O|+|V_M|}\}$, yielding $(|V_O| + |V_M|)!$ possible node permutations. Given a node ordering π , a sequence \mathbf{S} is then defined as:

$$\mathbf{S}^\pi \triangleq (\mathbf{S}_1^\pi, \dots, \mathbf{S}_{|V_O|+|V_M|}^\pi), \quad (2)$$

where each element $\mathbf{S}_i^\pi \in \{0, 1\}^{i-1}$ for $i \in \{2, \dots, |V_O| + |V_M|\}$ is a binary adjacency vector representing the edges

4. The number of nodes in G_I should be greater than or equal to that in G_T so that the information (i.e., the distribution p_{model}) encoded by learned parameters Θ is sufficient to infer G_T .

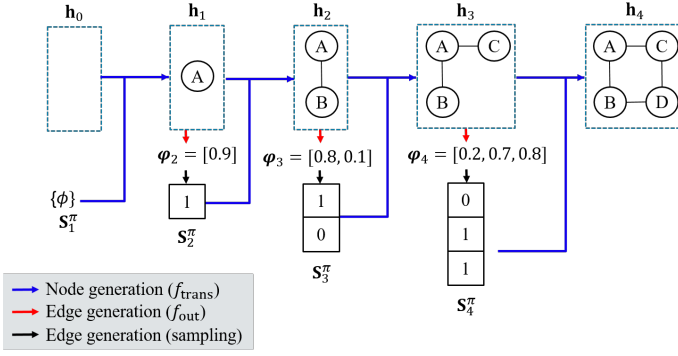


Fig. 2: An example illustrating the inference process of GraphRNN-S. Here, the blue arrows denote the graph-level RNN that encodes the “graph state” vector \mathbf{h}_i in its hidden state, and the red and black arrows represent the edge generation process whose input is given by the graph-level RNN.

between node $\pi(v_i)$ and the previous nodes $\pi(v_j)$ for $j \in \{1, \dots, i-1\}$ that already exist in the graph, and $\mathbf{S}_1^\pi = \emptyset$. Here, \mathbf{S}_i^π can be expressed as

$$\mathbf{S}_i^\pi = (a_{1,i}^\pi, \dots, a_{i-1,i}^\pi), \quad \forall i \in \{2, \dots, |V_O| + |V_M|\}, \quad (3)$$

where $a_{u,v}^\pi$ denotes the (u, v) -th element of the adjacency matrix $\mathbf{A}^\pi \in \{0, 1\}^{(|V_O|+|V_M|) \times (|V_O|+|V_M|)}$ for $u, v \in \{1, \dots, |V_O| + |V_M|\}$ (refer to Fig. 2 for an illustration of the sequence). Due to the fact that the graphs are discrete objects, the graph generation process involves discrete decisions that are not differentiable and therefore problematic for backpropagation. Thus, instead of directly learning the distribution p_{model} , we sample π from the set of $(|V_O| + |V_M|)!$ node permutations to generate the sequences \mathbf{S}^π and learn the distribution $p(\mathbf{S}^\pi)$ over sequences.

Next, we explain how to characterize the distribution $p(\mathbf{S}^\pi)$. Due to the sequential nature of \mathbf{S}^π , the distribution $p(\mathbf{S}^\pi)$ for a given node ordering π can be decomposed into the product of conditional probability distributions over the elements as follows:

$$p(\mathbf{S}^\pi) = \prod_{i=2}^{|V_O|+|V_M|} p(\mathbf{S}_i^\pi | \mathbf{S}_1^\pi, \dots, \mathbf{S}_{i-1}^\pi). \quad (4)$$

For ease of notation, we simplify $p(\mathbf{S}_i^\pi | \mathbf{S}_1^\pi, \dots, \mathbf{S}_{i-1}^\pi)$ as $p(\mathbf{S}_i^\pi | \mathbf{S}_{<i}^\pi)$ for the remainder of the paper.

Now, we turn to describing the use of RNN in generating a sequence \mathbf{S}^π from the training data G_I . The core idea is to learn two functions f_{trans} and f_{out} that are used in each generation step according to the following procedure (refer to Fig. 2). We denote $\mathbf{h}_i \in \mathbb{R}^d$ as the graph state vector representing the hidden state of the model in the i -th step, where $d \in \mathbb{N}$ is a user-defined parameter that is typically set to a value smaller than $|V_O| + |V_M|$. A state-transition function f_{trans} is used to compute the graph state vector \mathbf{h}_i based on both the previous hidden state \mathbf{h}_{i-1} and the input \mathbf{S}_i^π , and is given by

$$\mathbf{h}_i = f_{\text{trans}}(\mathbf{h}_{i-1}, \mathbf{S}_i^\pi). \quad (5)$$

Intuitively, \mathbf{h}_i encodes the topological information of i generated nodes in a low-dimensional vector. For the first

generation step, we randomly initialize \mathbf{h}_0 and set $\mathbf{S}_1^\pi = \emptyset$ to produce \mathbf{h}_1 . Then, as the output of the i -th step of GraphRNN-S, an output function f_{out} is invoked to obtain a vector $\boldsymbol{\varphi}_{i+1} \in (0, 1)^i$ specifying the distribution of the next node’s adjacency vector as

$$\boldsymbol{\varphi}_{i+1} = f_{\text{out}}(\mathbf{h}_i). \quad (6)$$

In GraphRNN-S, $p(\mathbf{S}_i^\pi | \mathbf{S}_{<i}^\pi)$ is modeled as a multivariate Bernoulli distribution parametrized by $\boldsymbol{\varphi}_i$. Thus, every entry of $\boldsymbol{\varphi}_i$ in (6) can be interpreted as a probability representing whether there exists an edge between nodes i and j for $j \in \{1, \dots, i-1\}$. The function f_{trans} is found via general neural networks such as gated recurrent units (GRUs) [35] or long short-term memory (LSTM) units [36] in RNN, and the function f_{out} is a multilayer perceptron. The weights of f_{trans} and f_{out} are optimized using training sequences sampled from G_I (refer to [10] for further details on the training process). It is worth noting that, rather than learning to generate graphs under any possible node permutations, GraphRNN-S learns from samples generated via breadth-first search (BFS) to allow the training process to be tractable.

A set of model parameters Θ is referred to as learned weights of both f_{trans} and f_{out} after the training process. Fig. 2 illustrates the inference process of GraphRNN-S, where a graph consisting of four nodes is generated as depicted from left to right. In more detail, after obtaining $\boldsymbol{\varphi}_2$ via (5) and (6), $\mathbf{S}_2^\pi = [1]$ is acquired by sampling from the multivariate Bernoulli distribution parameterized by $\boldsymbol{\varphi}_2$, which means that the next generated node (i.e., node B) is linked to node A. Following a similar procedure, we obtain $\mathbf{S}_3^\pi = [1, 0]$ and $\mathbf{S}_4^\pi = [0, 1, 1]$ representing the connections of nodes C and D with previously generated nodes, respectively.

3.3 Network Completion

In this subsection, we present our DeepNC method that recovers the missing part of the true network G_T based on the deep generative model. We first describe the approach that seamlessly accommodates both observable and missing parts of G_T into the graph generation process using the trained functions f_{trans} and f_{out} in Section 3.2. Then, we present the problem reformulation built upon (1).

We start by modeling the graphs that we want to recover as sequences and incorporating the information from the observed graph G_O into the generation process. To this end, we reuse the notation \mathbf{S}^π in (2) so that the sequence accounts for both observable and missing parts, where indices of missing nodes correspond to placeholders (e.g., M_1 and M_2 in Fig. 3), if such inclusion of unknown entries in \mathbf{S}^π does not cause any confusion. Then, we solve (1) through *data imputation* of the unknown entries (i.e., the entries associated with missing nodes), which also include non-existent edges between nodes in G_O . Let

$$\tilde{\mathbf{S}}^\pi = (\tilde{\mathbf{S}}_1^\pi, \dots, \tilde{\mathbf{S}}_{|V_O|+|V_M|}^\pi) \quad (7)$$

denote the sequence that we obtain from data imputation under a node ordering π , which contains both observable edges taken directly from \mathbf{S}^π , corresponding to the set E_O , and possible instances of all missing entries. Then, we impute each missing entry in \mathbf{S}^π with either 0 or 1, thereby

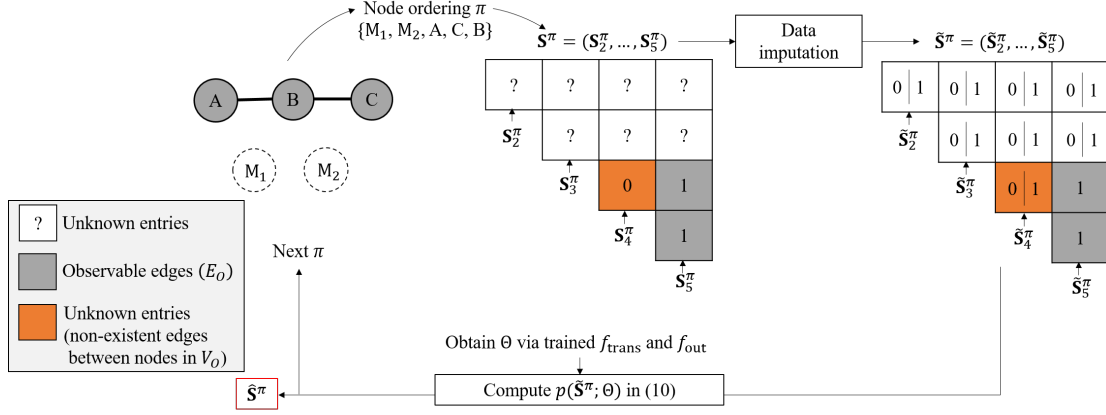


Fig. 3: An example illustrating the schematic overview of our DeepNC method, where three nodes (i.e., A, B, and C) and two edges with solid lines are observable instead of the true graph G_T consisting of five nodes and all associated edges. Both white and orange entries in \tilde{S}^π are imputed with either 0 or 1 while grey entries in \tilde{S}^π remain unchanged.

yielding $2^{\frac{(|V_O|+|V_M|)(|V_O|+|V_M|-1)}{2}-|E_O|}$ possible outcomes of \tilde{S}^π , where data imputation for non-existent edges between nodes in G_O (i.e., orange entries in \tilde{S}^π of Fig. 3) can be thought of as *link prediction* since structural neighborhood information regarding observable nodes is available. For each outcome, we use trained functions f_{trans} and f_{out} to obtain the corresponding φ_i for $i \in \{2, \dots, |V_O| + |V_M|\}$.

Next, since the constraint in (1) is incorporated into \tilde{S}^π in (7), we reformulate our optimization problem in (1) as finding a sequence \hat{S}^π that maximizes the probability $p(\tilde{S}^\pi; \theta)$ under a node ordering π from a distribution of sequences parametrized by θ as follows:

$$\hat{S}^\pi = \arg \max_{\tilde{S}^\pi} p(\tilde{S}^\pi; \theta), \quad (8)$$

which can be computed as

$$\begin{aligned} p(\tilde{S}^\pi; \theta) &= p(\tilde{S}_2^\pi | \tilde{S}_{<2}^\pi; \theta) p(\tilde{S}_3^\pi | \tilde{S}_{<3}^\pi; \theta) \cdots \\ &= p(\tilde{S}_{|V_O|+|V_M|}^\pi | \tilde{S}_{<|V_O|+|V_M|}^\pi; \theta) \\ &= p(\tilde{S}_2^\pi; \{\mathbf{h}_1, \varphi_2\}) p(\tilde{S}_3^\pi; \{\mathbf{h}_2, \varphi_3\}) \cdots \\ &= p(\tilde{S}_{|V_O|+|V_M|}^\pi; \{\mathbf{h}_{|V_O|+|V_M|-1}, \varphi_{|V_O|+|V_M|}\}) \\ &= p(\tilde{S}_2^\pi; \varphi_2) p(\tilde{S}_3^\pi; \varphi_3) \cdots p(\tilde{S}_{|V_O|+|V_M|}^\pi; \varphi_{|V_O|+|V_M|}) \\ &= \prod_{i=2}^{|V_O|+|V_M|} p(\tilde{S}_i^\pi; \varphi_i), \end{aligned} \quad (9)$$

where the first equality follows due to (4); the second equality holds since θ is the set of learned model parameters of both f_{trans} and f_{out} in (5) and (6), respectively; and the third equality stems from the fact that \tilde{S}_i^π is determined only by φ_i . Since φ_i is the set of variables of a multivariate Bernoulli distribution in which each entry represents the likelihood of edge existence, we have:

$$p(\tilde{S}^\pi; \theta) = \prod_{i=2}^{|V_O|+|V_M|} \left(\prod_{\tilde{s}_{i,j}^\pi=1} \varphi_{i,j} \prod_{\tilde{s}_{i,j}^\pi=0} (1 - \varphi_{i,j}) \right), \quad (10)$$

where $\tilde{s}_{i,j}^\pi$ denotes the j -th element of the binary vector \tilde{S}_i^π for $i \in \{2, \dots, |V_O| + |V_M|\}$ and $j \in \{1, \dots, i-1\}$;

and $\varphi_{i,j} \in (0, 1)$ is the j -th element of φ_i . An example visualizing our DeepNC method is presented in Fig. 3, where we observe a network G_O consisting of three nodes (i.e., A, B, and C) and two edges, instead of the true network G_T with 5 nodes (i.e., A, B, C, M_1 , and M_2).

To solve (8), we need to compute $p(\tilde{S}^\pi; \theta)$ via exhaustive search over $(|V_O| + |V_M|)!$ node permutations. Since computing $p(\tilde{S}^\pi; \theta)$ in (9) requires $\frac{(|V_O|+|V_M|)^2}{2}$ multiplication operations and data imputation yields $2^{\frac{(|V_O|+|V_M|)(|V_O|+|V_M|-1)}{2}-|E_O|}$ possible outcomes of \tilde{S}^π , its computational complexity is bounded by $\mathcal{O}((|V_O| + |V_M|)^2 2^{\frac{(|V_O|+|V_M|)(|V_O|+|V_M|-1)}{2}-|E_O|} (|V_O| + |V_M|)!)$. This motivates us to introduce a low-complexity algorithm in the next section for efficiently solving this problem.

4 DEEPCNC ALGORITHMS

In this section, we introduce two algorithms that we design to efficiently solve the network completion problem in (8). In designing such algorithms, we focus on how to compute the likelihood of edge existence in the form of a tuple $(\hat{\pi}, \Phi)$, where $\hat{\pi}$ represents a node ordering to be inferred and $\Phi = \{\varphi_2, \dots, \varphi_{|V_O|+|V_M|}\}$. Then, \hat{S}^π in (8) can be acquired by sampling from $(\hat{\pi}, \Phi)$. First, we present DeepNC-L, a low-complexity deep network completion algorithm, working based on the assumption that a partially observable graph G_O is a complete subgraph with no missing edges. Second, we present an enhanced version of DeepNC-L using the EM algorithm [37], dubbed DeepNC-EM, to deal with the case where edges are missing in G_O . The overall architecture of both DeepNC algorithms is illustrated in Fig. 4. We also analyze their computational complexities.

4.1 DeepNC-L Algorithm

4.1.1 Overall Procedure

We propose DeepNC-L that approximates the optimal solution to (8) under the assumption that there are no missing edges in G_O , which implies that the non-existent edges between nodes in G_O are regarded as observable entries in \tilde{S}^π . Since Φ indicates the set of edge existence probabilities and

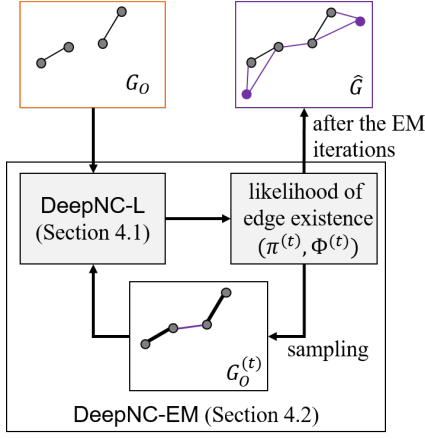


Fig. 4: The overall architecture of DeepNC algorithms.

is thus obtained from the set of learned model parameters Θ for each π , (8) can be simplified to the problem of finding a node ordering $\hat{\pi}$ such that

$$\hat{\pi} = \arg \max_{\pi} p(\tilde{\mathbf{S}}^{\pi}; \Theta), \quad (11)$$

where $\tilde{\mathbf{S}}^{\pi}$ is the sequence after data imputation under a given π .

To efficiently solve (11), we present two judicious approximation methods in the following. First, we design a *greedy* strategy that selects a single node at each inference (generation) step. More precisely, instead of exhaustively searching for the node ordering that maximizes $p(\tilde{\mathbf{S}}^{\pi}; \Theta)$ among $(|V_O| + |V_M|)!$ possible permutations, we aim to *consecutively* find a single node $\hat{v} \in V^{(i)}$ such that

$$\begin{aligned} \hat{v} &= \arg \max_{v \in V^{(i)}} p(\tilde{\mathbf{S}}_i^{\pi}; \varphi_i) \\ &\text{subject to } \pi(v) = i \end{aligned} \quad (12)$$

for each step $i \in \{2, \dots, |V_O| + |V_M|\}$, where $V^{(i)}$ is a set of nodes that have not been generated until the i -th inference step and \hat{v} is removed from $V^{(i)}$ after each inference step. That is, $V^{(i+1)} \leftarrow V^{(i)} \setminus \{\hat{v}\}$ (refer to Fig. 5 for the node removal). We note that the first node can be arbitrarily chosen in the generation process. Second, we further approximate the solution to (12) by treating all unknown entries (i.e., missing data) in $\tilde{\mathbf{S}}_i^{\pi}$ *equally* during the computation while retrieving \hat{v} from the set $V^{(i)}$, rather than computing the likelihoods in (12) along with all entries in $\tilde{\mathbf{S}}_i^{\pi}$. Let us define *two types of nodes* as observable nodes and missing nodes. Then, we select a node of either type at random in *proportion to the number of nodes* belonging to *each type* in $V^{(i)}$ to ensure that there is no bias in the node selection. When the selected node type is “missing”, we choose \hat{v} at random from all missing nodes in $V^{(i)}$ without any computation since all missing nodes are treated equally. In contrast, when the selected node type is “observable”, we choose an observable node based solely on the computation for the observable entries in \mathbf{S}_i^{π} by reformulating our problem as follows:

$$\begin{aligned} \hat{v} &= \arg \max_{v \in V_O \cap V^{(i)}} p(\mathbf{O}_i^{\pi}; \varphi_i) \\ &\text{subject to } \pi(v) = i, \end{aligned} \quad (13)$$

for each step $i \in \{2, \dots, |V_O| + |V_M|\}$, where \mathbf{O}_i^{π} denotes the set of observable entries in \mathbf{S}_i^{π} ; $p(\mathbf{O}_i^{\pi}; \varphi_i) = \prod_{s_{i,j}^{\pi}=1} \varphi_{i,j} \prod_{s_{i,j}^{\pi}=0} (1 - \varphi_{i,j})$ since the observable entries are only taken into account and $s_{i,j}^{\pi}$ denotes the j -th element of \mathbf{S}_i^{π} ; and $V_O \cap V^{(i)}$ indicates the set of remaining observable nodes after $i - 1$ inference steps. Note that $p(\mathbf{O}_i^{\pi}; \varphi_i)$ is non-computable if there is no observable entry in \mathbf{S}_i^{π} .

Now, we are ready to show a stepwise description of the DeepNC-L algorithm.

1. Initialization: For $i = 1$, we set $V^{(1)}$ to $V_O \cup V_M$ and randomly choose a node in $V^{(1)}$ to be \hat{v} .

2. Node selection: For $i \in \{2, \dots, |V_O| + |V_M|\}$, we find \hat{v} by either randomly selecting a missing node in $V^{(i)}$ or solving (13), depending on which node type is selected.

3. Data imputation: After finding \hat{v} , we apply a *data imputation* strategy of the missing part (i.e., unknown entries) in \mathbf{S}_i^{π} through the inference process of GraphRNN-S. Specifically, suppose that $\pi(u) = i$ and $\pi(v) = j$, which means that the i -th and j -th nodes in a given node ordering π are u and v , respectively. Then, we have

$$\tilde{s}_{i,j}^{\pi} = \begin{cases} \text{Bernoulli}(\varphi_{i,j}), & \text{if } u \notin V_O \text{ or } v \notin V_O \\ s_{i,j}^{\pi}, & \text{otherwise,} \end{cases} \quad (14)$$

where the Bernoulli trial with the probability $\varphi_{i,j}$ maps the value of the unknown entry to 1 if the outcome “success” occurs and to 0 otherwise.

4. Repetition: We iterate the second and third steps $|V_O| + |V_M| - 1$ times until the recovered graph is fully generated.

For a more intuitive understanding, consider the following example.

Example 1: As illustrated in Fig. 5, let us describe three steps to select the first three nodes of a given graph according to the aforementioned procedure. We start by randomly assigning the first node of the inference process to node M_1 (i.e., $\pi(M_1) = 1$ and $V^{(2)} \leftarrow V^{(1)} \setminus \{M_1\}$). Since we do not have any information about the connections for the unseen node M_1 , $s_{2,1}^{\pi}$ is unknown for all nodes $v \in V^{(2)}$. Suppose that we generate an observable node at this step by random selection. Since there is no observable entry in \mathbf{S}_i^{π} , we randomly choose node A among the three nodes in $V_O \cap V^{(2)}$ as the second node and set $\pi(A) = 2$, resulting in $V^{(3)} \leftarrow V^{(2)} \setminus \{A\}$. Assuming that $\varphi_2 = [0.9]$ and a Bernoulli trial with the probability φ_2 returns 1, we impute $\tilde{s}_{2,1}^{\pi}$ with 1 according to (14). Let us turn to the next step in order to select the third node. In this case, since nodes B and C belong to the type of observable nodes, $\tilde{s}_{3,2}^{\pi}$ takes the value of either 1 or 0, depending on the connections to node A. Suppose that we again generate an observable node at this step and $\varphi_3 = [0.75, 0.2]$. When either $\pi(B) = 3$ or $\pi(C) = 3$, the likelihood $p(\mathbf{O}_3^{\pi}; \varphi_3)$ can be computed as:

- If $\pi(B) = 3$, then it follows that $p(\mathbf{O}_3^{\pi}; \varphi_3) = \varphi_{3,2} = 0.2$ using (9).
- If $\pi(C) = 3$, then it follows that $p(\mathbf{O}_3^{\pi}; \varphi_3) = 1 - \varphi_{3,2} = 1 - 0.2 = 0.8$ in a similar manner.

Based on the above results, setting $\pi(C)$ to 3 leads to the maximum value of $p(\mathbf{O}_3^{\pi}; \varphi_3)$, which is thus the solution to the problem in (13) for $i = 3$. As depicted in Fig. 5, node C is chosen in this step. By assuming that a Bernoulli trial

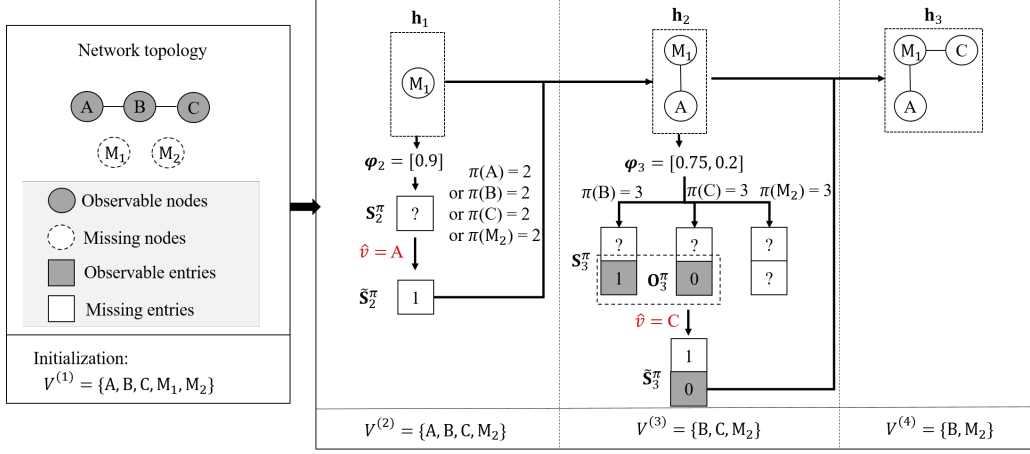


Fig. 5: An illustration of the mechanism of DeepNC-L. The first three steps are shown as an example.

with the probability $\varphi_{3,1} = 0.75$ returns 1, we finally have $\hat{S}_3^\pi = [1, 0]$.

4.1.2 Computational Efficiency

In the following, we examine how to efficiently compute the likelihoods in (13) through a complexity reduction technique. We start by making a helpful observation as illustrated in Fig. 6. Suppose that nodes M_1, A, B , and E from the original graph with 8 observable nodes and 3 missing nodes have already been generated sequentially after four inference steps. Then, one can see that $O_5^\pi = 0$ when node D, G , or H is selected in the fifth step (i.e., $\pi(D) = 5, \pi(G) = 5$, or $\pi(H) = 5$) since each of the three nodes has no connection to the nodes A, B , and E that have already been generated. Consequently, the likelihood $p(O_5^\pi; \varphi_5)$ is identical for these three cases. We generalize this observation in the following lemma.

Lemma 1. Let $L^{(i)}$ denote the set of not yet selected direct neighbors of observable nodes generated for $i - 1$ inference steps, expressed as

$$L^{(i)} = \begin{cases} (L^{(i-1)} \cup \mathcal{N}(\hat{v})) \cap V^{(i)}, & \text{if } \hat{v} \in V_O \\ L^{(i-1)} \cap V^{(i)}, & \text{otherwise,} \end{cases} \quad (15)$$

where $i \in \{2, \dots, |V_O| + |V_M|\}$, $L^{(1)} = \emptyset$, \hat{v} is the selected node in the $(i - 1)$ -th step, and $\mathcal{N}(\hat{v})$ is the set of (direct) neighbors of \hat{v} . Then, the likelihood $p(O_i^\pi; \varphi_i)$ in (13) is the same for all $u \notin L^{(i)}$, where $u \in V_O$ and $\pi(u) = i$.

Proof. For the observable node u that does not belong to the set $L^{(i)}$ and is not generated for $i - 1$ inference steps, all observable entries in S_i^π (i.e., entries in O_i^π) take the value of 0's since there is no associated edge. Thus, it follows that $p(O_i^\pi; \varphi_i) = \prod_{s_{i,j}^\pi=0} (1 - \varphi_{i,j})$, which is identical for all $u \notin L^{(i)}$, where $u \in V_O$ and $\pi(u) = i$. This completes the proof of this lemma. \square

Lemma 1 allows us to compute the likelihood $p(O_i^\pi; \varphi_i)$ only once for all nonselected observable nodes $u \notin L^{(i)}$ when solving (13), which corresponds to the case where node D, G , or H is selected in the fifth step in Fig. 6 while $L^{(5)} = \{C, F\}$, indicating the set of nonselected neighbors of nodes A, B , and E .

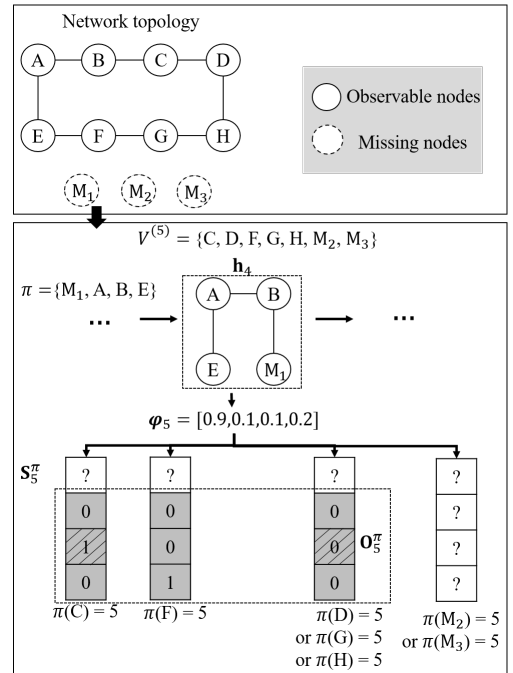


Fig. 6: An example illustrating the fifth inference step of DeepNC-L, where nodes M_1, A, B , and E have been generated sequentially.

Next, we explain how to efficiently solve the problem in (13) without computing likelihoods $p(O_i^\pi; \varphi_i)$ for observable nodes. From Fig. 6, one can see that $s_{5,3}^\pi$ (corresponding to entries with diagonal lines in O_5^π) is the only term that makes the difference between two sets O_5^π for the cases when node C is selected and when either node D, G , or H is selected, which implies that it may not be necessary to compute the likelihoods of $s_{5,2}^\pi$ and $s_{5,4}^\pi$ for node selection. Thus, from the fact that most of the entries in O_i^π tend to be 0's in many real-world networks that are usually sparse, the computational complexity can be greatly reduced if we make the comparison of likelihoods in (13) based only on the entries in O_i^π that have a value of 1. To this end, we eliminate all terms $(1 - \varphi_{i,j})$ corresponding to $s_{i,j}^\pi = 0$

from $p(\mathbf{O}_i^\pi; \varphi_i)$ when a node $v \in V_O \cap V^{(i)}$ is selected. For computational convenience, we define

$$D_v = \frac{\prod_{s_{i,j}^\pi=1} \varphi_{i,j} \prod_{s_{i,j}^\pi=0} (1 - \varphi_{i,j})}{\prod_{s_{i,j}^\pi \in \mathbf{O}_i^\pi} (1 - \varphi_{i,j})} \quad (16)$$

$$= \prod_{s_{i,j}^\pi=1} \frac{\varphi_{i,j}}{(1 - \varphi_{i,j})}$$

for $v \in V_O \cap V^{(i)}$. Since the denominator in (16) is the same for all $v \in V_O \cap V^{(i)}$, it is obvious that $\hat{v} = \arg \max_v D_v$ is the solution to (13). We note that computing D_v is less computationally expensive than computing $p(\mathbf{O}_i^\pi; \varphi_i)$ when the number of entries with the value of 1's in \mathbf{O}_i^π is low. As a special case in which all observable entries in \mathbf{S}_i^π take the value of 0's, the denominator in (16) is equivalent to $p(\mathbf{O}_i^\pi; \varphi_i)$, from which it follows that $D_u = 1$ due to the fact that a node $u \notin L^{(i)}$ is selected. Thus, if $D_v < 1$ for all $v \in L^{(i)}$, then the likelihood in (13) for selecting a node $u \notin L^{(i)}$ is higher than that for selecting a node $v \in L^{(i)}$. In this case, we randomly choose a node $\hat{v} \notin L^{(i)}$ without further computation based on Lemma 1. In consequence, we compute D_v only for nodes in the set $L^{(i)}$, rather than computing D_v for all nodes in $V_O \cap V^{(i)}$. The following example describes how the computational complexity can be reduced according to the aforementioned technique by revisiting Fig. 6.

Example 2: Suppose that we generate an observable node at the fifth inference step and $\varphi_5 = [0.9, 0.1, 0.1, 0.2]$. In this step, one can see that $L^{(5)} = \{C, F\}$; thus, instead of computing the likelihood $p(\mathbf{O}_5^\pi; \varphi_5)$ in (13) five times for all nonselected observable nodes C, D, F, G, and H in $V^{(5)}$, we only compute $D_C = \frac{\varphi_{5,3}}{1 - \varphi_{5,3}} = \frac{0.1}{1 - 0.1}$ and $D_F = \frac{\varphi_{5,4}}{1 - \varphi_{5,4}} = \frac{0.2}{1 - 0.2}$ from (16). Since both D_C and D_F are smaller than 1, we randomly choose one of the three observable nodes D, G, and H that are not in $L^{(5)}$ as \hat{v} .

4.1.3 Stepwise Summary of DeepNC-L

We summarize the overall procedure of our DeepNC-L algorithm in Algorithm 1. We initially select the first node at random, and then start the inference process by identifying connections for the next node according to the following four stages:

1. Using the two functions f_{trans} and f_{out} in (5) and (6), respectively, we obtain φ_i (refer to lines 4–5).
2. Let m denote the cardinality of the set of missing nodes that can be potentially generated in the i -th step. We then randomly select a node type so that the selected node is missing with probability of $\frac{m}{|V_O| + |V_M| - i + 1}$ (refer to line 6).
- 3a. If the type of observable nodes is selected, then we compute D_v , which is a function of φ_i , according to (16) for all $v \in L^{(i)}$. When $D_v < 1$ for all $v \in L^{(i)}$ or $L^{(i)} = \emptyset$, we randomly select an observable node $\hat{v} \notin L^{(i)}$ provided that $L^{(i)} \neq V_O \cap V^{(i)}$. Otherwise, we select the node \hat{v} that maximizes D_v . Afterwards, we update $L^{(i)}$ by including neighbors of the selected node \hat{v} (refer to lines 7–14).
- 3b. If the type of missing nodes is selected, then we select one node \hat{v} randomly among all missing nodes that have not been generated until the i -th step. (refer to lines 15–16).
4. The data imputation process takes place before the next iteration of node generation. Finally, we update the

Algorithm 1: DeepNC-L

Input: $G_O, |V_M|, f_{\text{out}}, f_{\text{trans}}$
Output: $(\hat{\pi}, \Phi)$

- 1 **Initialization:** $i \leftarrow 2; \mathbf{h}_0 \leftarrow$ random initialization;
 $\tilde{\mathbf{S}}_1^\pi \leftarrow \emptyset; \hat{v} \leftarrow v \in V_O \cup V_M; \pi(\hat{v}) \leftarrow 1;$
 $L^{(1)} \leftarrow \emptyset$; Update $L^{(i)}$ according to (15);
- 2 **function** DeepNC-L
- 3 **while** $i \geq |V_O| + |V_M|$ **do**
- 4 $\mathbf{h}_{i-1} \leftarrow f_{\text{trans}}(\mathbf{h}_{i-2}, \tilde{\mathbf{S}}_{i-1}^\pi)$
- 5 $\varphi_i \leftarrow f_{\text{out}}(\mathbf{h}_{i-1})$
- 6 Select a node type
- 7 **if** the selected node type is “observable” **then**
- 8 **for** $v \in L^{(i)}$ **do**
- 9 Compute D_v according to (16)
- 10 **if** ($D_v < 1$ for all v or $L^{(i)} = \emptyset$) and $L^{(i)} \neq V_O \cap V^{(i)}$ **then**
- 11 Randomly select an observable node
 $\hat{v} \notin L^{(i)}$
- 12 **else**
- 13 $\hat{v} \leftarrow \arg \max_v D_v$
- 14 Update $L^{(i)}$ according to (15)
- 15 **else**
- 16 Randomly select an unobservable node \hat{v}
- 17 $\tilde{\mathbf{S}}_i^\pi \leftarrow$ Impute \mathbf{S}_i^π according to (14)
- 18 $\pi(\hat{v}) \leftarrow i + 1$
- 19 $i \leftarrow i + 1$
- 20 **return** $(\hat{\pi}, \Phi)$

node ordering π by including the selected node \hat{v} for the i -th step. The algorithm continues by repeating stages 1–4 and terminates when a fully inferred sequence \mathbf{S}^π is generated (refer to lines 17–20).

We remark that a node ordering $\hat{\pi}$ is found given a set of edge existence probabilities Φ , which is inferred by our model parameters Θ , while assuming that G_O is a complete subgraph; thus, the resulting tuple $(\hat{\pi}, \Phi)$ may not be accurate when there are missing edges in G_O . This motivates us to develop the DeepNC-EM algorithm in the following subsection.

4.2 DeepNC-EM Algorithm

In this subsection, we introduce DeepNC-EM to further improve the performance of DeepNC-L by relaxing the assumption that there are no missing edges between two nodes in G_O . A naïve recovery of G_O even with state-of-the-art link prediction methods before conducting network completion may lead to suboptimal performance since the network structures of G_O are potentially distorted due to the effect of missing nodes and missing incident edges. Thus, we aim to find the most likely configuration of three types of missing edges in the set E_M specified in Section 3.1.1 by jointly estimating a tuple (π, Φ) . To this end, we solve (8) by designing an improved DeepNC method using the EM algorithm.

We now describe the proposed DeepNC-EM, which is built upon the DeepNC-L algorithm in Section 4.1. Let

Algorithm 2: DeepNC-EM

Input: $\pi^{(0)}, \Phi^{(0)}, G_O, |V_M|, f_{\text{out}}, f_{\text{trans}}, \Delta_s$
Output: $(\hat{\pi}, \hat{\Phi})$

1 **Initialization:** $t \leftarrow 0; \Phi_Z^{(0)} \leftarrow \text{Filter}(\pi^{(0)}, \Phi^{(0)});$
 function DeepNC-EM

2 **do**

3 E-step:

4 **for** $i \in \{1, \dots, \Delta_s\}$ **do**

5 $Z^{(t)}[i] \sim p(Z|\Phi_Z^{(t)})$

6 $G_O^{(t)}[i] \leftarrow$ add edges sampled from $Z^{(t)}[i]$

7 M-step:

8 **for** $i \in \{1, \dots, \Delta_s\}$ **do**

9 $(\pi^{(t+1)}[i], \Phi^{(t+1)}[i]) \leftarrow$
 DeepNC-L($G_O^{(t)}[i], |V_M|, f_{\text{out}}, f_{\text{trans}}$)

10 $\Phi_Z^{(t+1)}[i] \leftarrow \text{Filter}(\pi^{(t+1)}[i], \Phi^{(t+1)}[i])$

11 $\Phi_Z^{(t+1)} \leftarrow \frac{1}{\Delta_s} \sum_i \Phi_Z^{(t+1)}[i]$

12 $t \leftarrow t + 1$

13 **while** $\|\Phi_Z^{(t)} - \Phi_Z^{(t-1)}\|_2 < \eta$

14 $\hat{Z} \sim p(Z|\Phi_Z^{(t+1)})$

15 $\hat{G}_O \leftarrow$ add edges from \hat{Z}

16 $(\hat{\pi}, \hat{\Phi}) \leftarrow \text{DeepNC-L}(\hat{G}_O, |V_M|, f_{\text{out}}, f_{\text{trans}})$

17 **return** $(\hat{\pi}, \hat{\Phi})$

$(\pi^{(0)}, \Phi^{(0)})$ and Z denote the initial output of DeepNC-L and the set of non-existent edges between nodes in G_O , respectively. First, we estimate the *potential existence* likelihoods of edges in Z , denoted by Φ_Z , by extracting $|V_O|^2 - E_O$ elements corresponding to Z from the likelihoods $\Phi^{(0)}$ of all edges under the node ordering $\pi^{(0)}$. Then, the E-step samples $Z^{(t)}$ from $p(Z^{(t)}|\Phi_Z^{(t)})$ via Bernoulli trials to create multiple instances of $G_O^{(t)}$, where the superscript (t) denotes the EM iteration index. In the M-step, we adopt DeepNC-L to subsequently optimize the parameters Φ_Z given the samples obtained in the E-step. The EM iteration alternates between performing the E-step and M-step according to the following expressions, respectively:

$$\text{E-step: } Z^{(t)} \sim p(Z|\Phi_Z^{(t)}),$$

$$\text{M-step: } \Phi_Z^{(t+1)} = \arg \max_{\Phi_Z} \mathbb{E}[p(Z^{(t)}|\Phi_Z)].$$

The overall procedure of DeepNC-EM is summarized in Algorithm 2. Here, $\text{Filter}(\pi^{(t)}[i], \Phi^{(t)}[i])$ in lines 1 and 10 is invoked to retrieve $\Phi_Z^{(t)}$ from $\Phi^{(t)}$; $\eta > 0$ is an arbitrarily small threshold indicating a stopping criterion for the algorithm; Δ_s denotes the number of samples in each E-step; and $[i]$ indicates the sample index.

4.3 Complexity Analysis

In this subsection, we analyze the computational complexities of the DeepNC-L and DeepNC-EM algorithms.

4.3.1 Complexity of DeepNC-L

We start by examining the complexity of each inference step $i \in \{2, \dots, |V_O| + |V_M|\}$. It is not difficult to show that the complexity is dominated by the case in which an observable node is selected in the inference process. Note

that it is possible to compute D_v in constant time as the average degree over a network is typically regarded as a constant [38]. Thus, the complexity of this step is bounded by $\mathcal{O}(|L^{(i)}|)$ since we exhaustively compute D_v over the nodes $v \in L^{(i)}$. The data imputation process is computable in constant time when parallelization can be applied since the Bernoulli trials are independent of each other. As our algorithm is composed of $|V_O| + |V_M| - 1$ inference steps, the total complexity is finally given by $\mathcal{O}((|V_O| + |V_M|)|L^{(i)}|)$, which can be rewritten as $\mathcal{O}(|V_O| \cdot |L^{(i)}|)$ from the fact that $|V_M| \ll |V_O|$. The following theorem states a comprehensive analysis of this computational complexity.

Theorem 1. *Lower and upper bounds on the computational complexity of the proposed DeepNC-L algorithm are given by $\Omega(|V_O|)$ and $\mathcal{O}(|V_O|^2)$, respectively.*

Proof. The parameter $L^{(i)}$ is the set of neighboring nodes to the observable nodes that have already been generated in the i -th step, while its cardinality depends on the network topology. For the best case where all nodes are isolated with no neighbors, we always have $|L^{(i)}| = 0$ for each generation step; thus, each step is computable in constant time, yielding the total complexity of $\Omega(|V_O|)$. For the worst case, corresponding to a fully-connected graph, it follows that $|L^{(i)}| = |V_O| + |V_M| - i$ for each generation step, thus yielding the total complexity of $\mathcal{O}(|V_O|^2)$. This completes the proof of this theorem. \square

From Theorem 1, it is possible to establish the following corollary.

Corollary 1. *The computational complexity of the DeepNC-L algorithm scales as $\Theta(|V_O|^{1+\epsilon})$, where $0 \leq \epsilon \leq 1$ depends on a given network topology, i.e., the sparsity of networks.*

We shall validate this assertion in Corollary 1 via empirical evaluation for various datasets in the next section by identifying that ϵ is indeed small, which implies that the complexity of DeepNC-L is almost linear in $|V_O|$.

4.3.2 Complexity of DeepNC-EM

We turn to examining the computational complexity of each EM step to finally analyze the overall complexity. In the E-step, we can parallelize both the Bernoulli trials for edge sampling and the operation that adds sampled edges to $G_O^{(t)}[i]$ in lines 5 and 6, respectively. Consequently, the computational complexity of each E-step is given by $\mathcal{O}(\Delta_s)$, where Δ_s is the number of samples in each E-step. The M-step is dominated by DeepNC-L as the function $\text{Filter}(\cdot, \cdot)$ can also be executed in parallel since all operations therein are performed independently of each other. Thus, the computational complexity of each M-step is given by $\mathcal{O}(\Delta_s |V_O|^{1+\epsilon})$. When the number of EM iterations is given by k_{EM} , determined by the threshold η , and there are a total of Δ_s samples, the complexity of DeepNC-EM is finally given as $\Theta(k_{\text{EM}} \Delta_s |V_O|^{1+\epsilon})$ based on Corollary 1. Since both k_{EM} and Δ_s are regarded as constants as in [5], the total computational complexity scales as $\Theta(|V_O|^{1+\epsilon})$.

5 EXPERIMENTAL EVALUATION

In this section, we first describe both synthetic and real-world datasets that we use in the evaluation. We also

TABLE 3: Statistics of 5 datasets, where NG and NN denote the number of similar graphs and the range of the number of nodes in each dataset, respectively, including training graphs G_I and a test graph G_T . Here, k denotes 10^3 .

Name	NG	NN
LFR	500	1.6k–2k
B-A	500	1.6k–2k
Protein	918	100–500
Ego-CiteSeer	737	50–399
Ego-Facebook	10	52–1,034

present three state-of-the-art methods for network completion as a comparison. After presenting a performance metric and our experimental settings, we intensively evaluate the performance of our DeepNC algorithms.

5.1 Datasets

Two synthetic and three real-world datasets across various domains (e.g., social, citations, and biological networks) are used as a series of homogeneous networks (graphs), denoted by G_I , and described in sequence. For all experiments, we treat graphs as undirected and only consider the largest connected component without isolated nodes. The statistics of each dataset, including the number of similar graphs and the range of the number of nodes, is described in Table 3. In the following, we summarize important characteristics of the datasets.

Lancichinetti-Fortunato-Radicchi (LFR) [39]. We generate synthetic graphs with the LFR model in which the degree exponent of a power-law distribution, the average degree, the minimum community size, the community size exponent, and the mixing parameter are set to 3, 5, 20, 1.5, and 0.1, respectively. Refer to the original paper [39] for a detailed description of these parameters.

Barabasi-Albert (B-A) [14]. We generate further synthetic graphs using the B-A model. The attachment parameter of the model is set in such a way that each newly added node is connected to four existing nodes, unless otherwise stated.

Protein [8]. The protein structure network is a biological network. Each protein is represented by a graph, in which nodes represent amino acids. Two nodes are connected if they are less than 6 Angstroms apart.

Ego-CiteSeer [7]. This CiteSeer dataset is an online citation network and is a frequently used benchmark. Nodes and edges represent publications and citations, respectively.

Ego-Facebook [9]. This Facebook dataset is a social friendship network extracted from Facebook. Nodes and edges represent people and friendship ties, respectively.

5.2 State-of-the-art Approaches

In this subsection, we present three state-of-the-art network completion approaches for comparison.

KronEM [5]. This approach aims to infer the missing part of a true network based solely on the connectivity patterns in the observed part via a generative graph model based on Kronecker graphs, where the parameters are estimated via an EM algorithm.

EvoGraph [33]. To solve the network completion problem, EvoGraph infers the missing nodes and edges in such a way that the topological properties of the observable network are preserved via an efficient preferential attachment mechanism.

A variant of GraphRNN-S. As a naïve approach for network completion using deep generative models of graphs, we modify the inference process of the original GraphRNN-S [10] so that it can be used as a network completion method as follows. Under a random ordering of observable nodes, we first obtain the sequence $\{\mathbf{S}_2^\pi, \dots, \mathbf{S}_{|V_O|}^\pi\}$ along with the observable entries from G_O . Then, by invoking the inference process of GraphRNN-S, we generate $|V_M|$ missing nodes using trained functions f_{trans} and f_{out} based on $\{\mathbf{S}_2^\pi, \dots, \mathbf{S}_{|V_O|}^\pi\}$. This variant of GraphRNN-S for network completion is termed vGraphRNN in our evaluation.

5.3 Performance Metric

To assess the performance of our proposed method and the above state-of-the-art approaches, we need to quantify the degree of agreement between the recovered graph and the original graph. To this end, we adopt the GED as a well-known performance metric.

Definition 2. Graph edit distance (GED) [12]. *Given a set of graph edit operations, the GED between a recovered graph \hat{G} and the true graph G is defined as*

$$\text{GED}(\hat{G}, G) = \min_{(e_1, \dots, e_k) \in \mathcal{P}(\hat{G}, G)} \sum_{i=1}^k c(e_i), \quad (17)$$

where $\mathcal{P}(\hat{G}, G)$ denotes the set of edit paths transforming \hat{G} into a graph isomorphic to G , and $c(e) \geq 0$ is the cost of each graph edit operation e .

Note that only four operations are allowed in our setup, including vertex substitution, edge insertion, edge deletion, and edge substitution, and $c(e)$ is identically set to 1 for all operations. Since the problem of computing the GED is NP-complete [40], we adopt an efficient approximation algorithm proposed in [41]. In our experiments, GED is normalized by the average size of the two graphs.

5.4 Experimental Setup

We first describe the settings of the neural networks. In our experiments, the function f_{trans} is implemented by using 4 layers of GRU cells with a 128-dimensional hidden state; and the function f_{out} is implemented by using a two-layer perceptron with a 64-dimensional hidden state and a sigmoid activation function. The Adam optimizer [42] is used for minibatch training with a learning rate of 0.001, where each minibatch contains 32 graph sequences. We train the model for 32,000 batches in all experiments.

To test the performance of our method, we randomly select one graph from each dataset to act as the underlying true network G_T . From each dataset, we select all remaining similar graphs as training data G_I unless otherwise stated.

To create a partially observable network from the true network G_T , we adopt the following two graph sampling strategies from [43]. The first strategy, called *random node* (RN) sampling, selects nodes uniformly at random to create

a sample graph. The second strategy, *forest fire* (FF) sampling, starts by picking a seed node uniformly at random and adding it to a sample graph (referred to as burning). Then, FF sampling burns a fraction of the outgoing links with nodes attached to them. This process is repeated recursively for each neighbor that is burned until no new node is selected to be burned. Afterwards, we sample uniformly at random a portion of edges from the complete subgraph sampled from G_T to finally acquire G_O . In our experiments, the partially observable network G_O is constructed by 90% of edges in a complete subgraph consisting of 70% of nodes sampled from G_T unless otherwise specified. Each experimental result is averaged over 10 executions.

5.5 Experimental Results

Our empirical study in this subsection is designed to answer the following five key research questions.

- Q1. How much does the performance of DeepNC-EM improve with respect to the number of EM iterations?
- Q2. How much do the DeepNC algorithms improve the performance of network completion over the state-of-the-art approaches?
- Q3. How beneficial are the DeepNC algorithms in more difficult situations where either a large number of nodes and edges are missing or the training data are also incomplete?
- Q4. How robust is DeepNC-EM to the portion of missing edges in G_O in comparison with the other state-of-the-art approaches?
- Q5. How scalable are DeepNC algorithms with the size of the graph?

To answer these questions, we carry out six comprehensive experiments as follows.

5.5.1 Comparative Study Between DeepNC-L and DeepNC-EM (Q1)

In Fig. 7, we show the performance of the DeepNC-EM algorithm proposed in Section 4.2 with respect to GED according to the number of EM iterations using two synthetic datasets, i.e., the LFR and B-A models. As shown in Fig. 7, our findings are as follows:

- For both RN and FF sampling strategies, the GED of DeepNC-EM decreases as the number of EM iterations increases.
- The number of EM iterations required to achieve a sufficiently low GED value is relatively small compared to the network size. This can be seen from the LFR dataset, where the performance improvement is marginal after four iterations.
- We observe that DeepNC-EM exhibits less fluctuations over EM iterations on the LFR dataset. This might be caused by the fact that graphs generated using the LFR model are denser than those using the B-A model under our setting, which enables the algorithm to be more likely to correctly recover the edges connecting two nodes in the set V_O .

In the subsequent experiments, the number of EM iterations is set to 6.

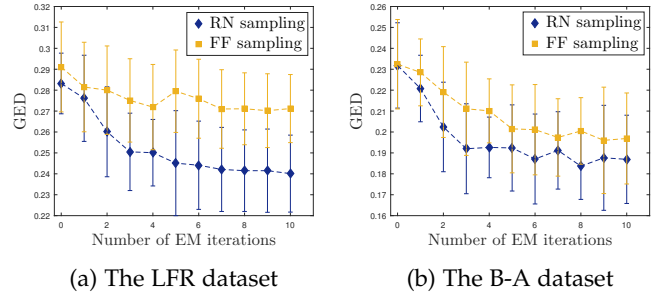


Fig. 7: GED of DeepNC-EM over the number of EM iterations. Here, the performance of DeepNC-L corresponds to the case where the number of EM iterations is zero.

5.5.2 Comparison With State-of-the-Art Approaches (Q2)

The performance comparison between two DeepNC algorithms and three state-of-the-art network completion methods, including vGraphRNN, KronEM [5], and EvoGraph [33], with respect to GED is presented in Table 4 for all five datasets. We note that DeepNC-EM, DeepNC-L, and vGraphRNN use structurally similar graphs as training data G_I ; meanwhile, both KronEM and EvoGraph operate based solely on the partially observable graph G_O without any training phase. We observe the following:

- The improvement rates of DeepNC-EM over vGraphRNN, KronEM, and EvoGraph are up to 40.16%, 54.55%, and 68.25%, respectively. These maximum gains are achieved for the Ego-CiteSeer and B-A datasets.
- The DeepNC-L and DeepNC-EM algorithms are insensitive to sampling strategies for creating a partially observable network, whereas the performance of EvoGraph depends on the sampling strategy. Specifically, sampling via FF results in better performance than that via RN sampling when EvoGraph is used due to the fact that the FF sampling strategy tends to preserve the network properties such as the degree distribution [43]. In reality, if the sampling strategy is unknown and one only acquires randomly sampled data, then graph upscaling methods such as EvoGraph would certainly perform poorly. This result displays the robustness of our DeepNC algorithms to graph samplings.
- Even with deletions of only 10% of edges, the additional gain of DeepNC-EM over DeepNC-L is still significant for all datasets. The maximum improvement rate of 13.58% is achieved on the Protein dataset.
- In a comparison of the performance differences between KronEM and EvoGraph, KronEM performs better in most cases. However, KronEM is inferior to EvoGraph in the case where the degree distribution of a network does not strictly follow the pure power-law degree distribution. EvoGraph consistently outperforms KronEM in the Protein dataset.
- The standard deviation of GED is relatively high when vGraphRNN is employed (e.g., 0.2514 for the Ego-CiteSeer dataset), which demonstrates that a random node ordering of observable nodes for net-

TABLE 4: Performance comparison in terms of graph edit distance (average \pm standard deviation). Here, the best method for each dataset is highlighted using bold fonts.

Method Dataset	DeepNC-EM (X)	DeepNC-L	vGraphRNN (Y ₁)	KronEM (Y ₂)	EvoGraph (Y ₃)	Gain (%)		
						$\frac{Y_1-X}{Y_1} \times 100$	$\frac{Y_2-X}{Y_2} \times 100$	$\frac{Y_3-X}{Y_3} \times 100$
LFR (RN)	0.2793 \pm 0.0145	0.2864 \pm 0.0206	0.3099 \pm 0.0241	0.3713 \pm 0.0428	0.5126 \pm 0.0124	9.87	24.78	45.51
LFR (FF)	0.2612 \pm 0.0205	0.2801 \pm 0.0214	0.3155 \pm 0.0197	0.3671 \pm 0.0278	0.4512 \pm 0.0075	17.21	28.85	42.11
B-A (RN)	0.1782 \pm 0.0120	0.1888 \pm 0.0104	0.2015 \pm 0.0210	0.3921 \pm 0.0304	0.5612 \pm 0.0084	11.56	54.55	68.25
B-A (FF)	0.1811 \pm 0.0106	0.2024 \pm 0.0134	0.2041 \pm 0.0202	0.3706 \pm 0.0418	0.5455 \pm 0.0087	11.27	51.13	66.80
Protein (RN)	0.2616 \pm 0.0521	0.3015 \pm 0.0520	0.3861 \pm 0.2101	0.4565 \pm 0.1077	0.4422 \pm 0.0014	32.25	42.69	40.84
Protein (FF)	0.2603 \pm 0.0571	0.3012 \pm 0.0481	0.3761 \pm 0.1121	0.4455 \pm 0.1240	0.4111 \pm 0.0025	30.79	41.57	36.68
Ego-CiteSeer (RN)	0.3012 \pm 0.0414	0.3236 \pm 0.0414	0.4915 \pm 0.2514	0.5811 \pm 0.0438	0.9166 \pm 0.0109	39.16	48.17	67.14
Ego-CiteSeer (FF)	0.3241 \pm 0.0571	0.3458 \pm 0.0511	0.5416 \pm 0.1918	0.5571 \pm 0.0518	0.9013 \pm 0.0041	40.16	41.82	64.04
Ego-Facebook (RN)	0.4213 \pm 0.0502	0.4535 \pm 0.0508	0.5928 \pm 0.2015	0.6167 \pm 0.0268	0.8161 \pm 0.0121	28.93	31.68	48.38
Ego-Facebook (FF)	0.4711 \pm 0.0471	0.5021 \pm 0.0604	0.6182 \pm 0.1897	0.6160 \pm 0.0447	0.7222 \pm 0.0104	23.79	23.52	34.77

TABLE 5: Performance comparison in terms of graph edit distance when 70% of nodes are missing (average \pm standard deviation). Here, the best method for each dataset is highlighted using bold fonts.

Method Dataset	DeepNC-EM (X)	DeepNC-L	vGraphRNN (Y ₁)	KronEM (Y ₂)	EvoGraph (Y ₃)	Gain (%)		
						$\frac{Y_1-X}{Y_1} \times 100$	$\frac{Y_2-X}{Y_2} \times 100$	$\frac{Y_3-X}{Y_3} \times 100$
LFR	0.2902 \pm 0.1204	0.3251 \pm 0.1245	0.3516 \pm 0.1284	0.6167 \pm 0.0802	0.7177 \pm 0.0212	17.46	52.94	59.57
B-A	0.2611 \pm 0.1021	0.2635 \pm 0.1018	0.2644 \pm 0.1487	0.6547 \pm 0.0728	0.8273 \pm 0.0140	1.25	60.12	68.44
Protein	0.3244 \pm 0.1014	0.3648 \pm 0.1189	0.4678 \pm 0.2428	0.9674 \pm 0.0437	0.7272 \pm 0.0161	30.65	66.47	55.39
Ego-CiteSeer	0.3414 \pm 0.1144	0.3988 \pm 0.1171	0.6031 \pm 0.3125	0.7727 \pm 0.0578	0.9161 \pm 0.0116	43.39	55.82	62.73
Ego-Facebook	0.5685 \pm 0.1412	0.5875 \pm 0.1280	0.6448 \pm 0.2985	0.8027 \pm 0.0689	0.9505 \pm 0.1057	11.83	29.18	40.19

work completion does not guarantee a stable solution.

Consequently, DeepNC-EM consistently outperforms all state-of-the-art methods for all synthetic and real-world datasets, which reveals the robustness of our method toward diverse network topologies.

5.5.3 Applicability to Fringe Scenarios (Q3)

We now compare our DeepNC algorithms to the three state-of-the-art network completion methods in more difficult settings that often occur in real environments: 1) the case in which a large portion of nodes are missing and 2) the case in which training graphs are also only partially observed. In these experiments, we only show the results for the RN sampling strategy since the results from FF sampling follow similar trends.

First, we create a partially observable network G_O consisting of only 30% of nodes from the underlying true graph G_T via sampling. The performance comparison between the DeepNC algorithms and the three state-of-the-art methods with respect to GED is presented in Table 5 for all five datasets. As shown in Tables 4 and 5, a large number of missing nodes and edges result in significant performance degradation for KronEM and EvoGraph, while DeepNC-EM, DeepNC-L, and vGraphRNN are more robust as the latter three methods take advantage of the topological information from similar graphs (i.e., training data) to infer the missing part.

Next, we perform RN sampling so that only a part of nodes in the training graphs is observable. In Fig. 8, we compare the GED of the two DeepNC algorithms and the three state-of-the-art methods, where the degree of observability in training graphs is set to $\{95, 90\}\%$. We find that the DeepNC algorithms still outperform the state-of-the-art methods on all datasets with the exception of the Ego-

Facebook dataset, where the performance of DeepNC-L is slightly inferior to that of KronEM when 90% of nodes in training graphs are observable.

5.5.4 Robustness to the Degree of Edge Observability in G_O (Q4)

We evaluate the GED performance in the second fringe scenario, in which a partially observable network G_O is created by deleting a large portion of edges uniformly at random from a complete subgraph that consists of 70% of nodes sampled from G_T . In Fig. 9, the performance of the DeepNC algorithms is compared to the state-of-the-art network completion methods using two synthetic datasets, where the fraction of missing edges is set to $\{10, 15, 20\}\%$. Our main findings are: 1) DeepNC-L outperforms the three state-of-the-art methods in all cases; 2) the gain of DeepNC-EM over DeepNC-L is more substantial when the LFR dataset is used since missing edges are inferred more accurately; and 3) both DeepNC algorithms exhibit less performance degradation as the number of missing edges increases, which demonstrates the robustness of our method for various degrees of edge observability.

From Tables 4–5 and Figs. 8–9, it is worth noting that the proposed DeepNC-EM algorithm outperforms all state-of-the-art methods for all types of datasets under various fringe scenarios and experimental settings.

5.5.5 Scalability (Q5)

Finally, we empirically show the average runtime complexity via experiments using the three sets of B-A synthetic graphs, which can conveniently be scaled up while preserving the same structural properties, where the number of connections from each new node to existing nodes is set to $c \in \{2, 4, 8\}$. In these experiments, we focus on evaluating the complexity of DeepNC-EM since EM iterations take

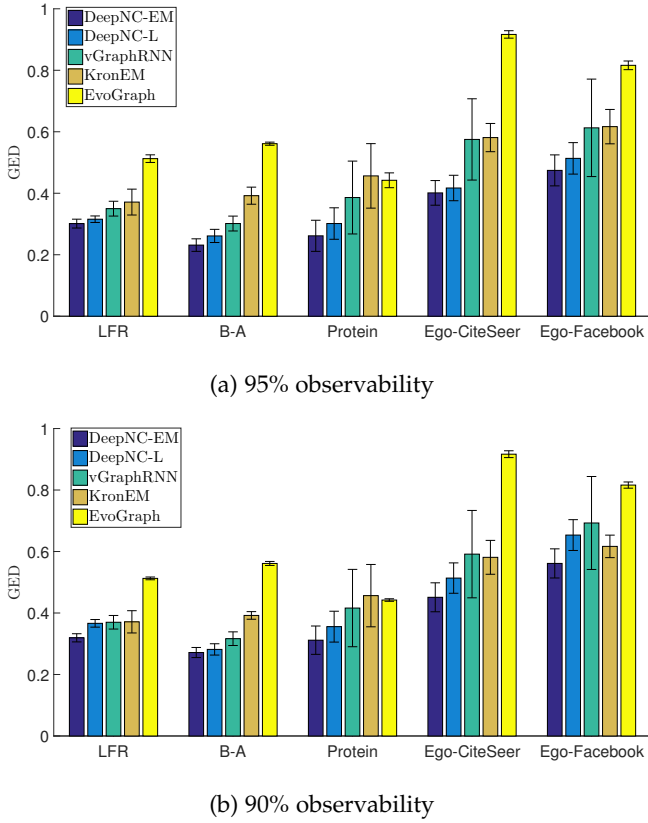


Fig. 8: Performance comparison in terms of GED (the lower the better), where the degree of observability in training graphs is set to $\{95, 90\}\%$.

constant time by executing DeepNC-L for each iteration. In each set of graphs, the number of nodes, $|V_O| + |V_M|$, varies from 200 to 2,000 in increments of 200; and 30% of nodes and their associated edges are deleted by RN sampling to create partially observable networks. Other parameter settings follow those in Section 5.4. In Fig. 10, we illustrate the log-log plot of the execution time in seconds versus $|V_O|$, where each point represents the average complexity over 10 executions of DeepNC-EM. In the figure, dotted lines are also shown from the analytical result with a proper bias, showing a tendency that slopes of the lines for $c \in \{2, 4, 8\}$ are approximately given by 1.16, 1.26, and 1.41, respectively. This indicates that the computational complexity of DeepNC-EM is dependent on the average degree in a given graph. Moreover, we find that an almost linear complexity in $|V_O|$, i.e., $\Theta(|V_O|^{1+\epsilon})$ for a small $\epsilon > 0$, is attainable since the slopes are at most 1.41 even for the relatively dense graph corresponding to $c = 8$.

6 CONCLUDING REMARKS

In this paper, we explored the open problem of recovering not only missing edges between observable nodes but also entirely hidden nodes and associated edges of an underlying true network. To tackle this new challenge, we introduced a novel method, termed DeepNC, that infers such missing nodes and edges via deep learning. Specifically, we presented an approach to first learning a likelihood

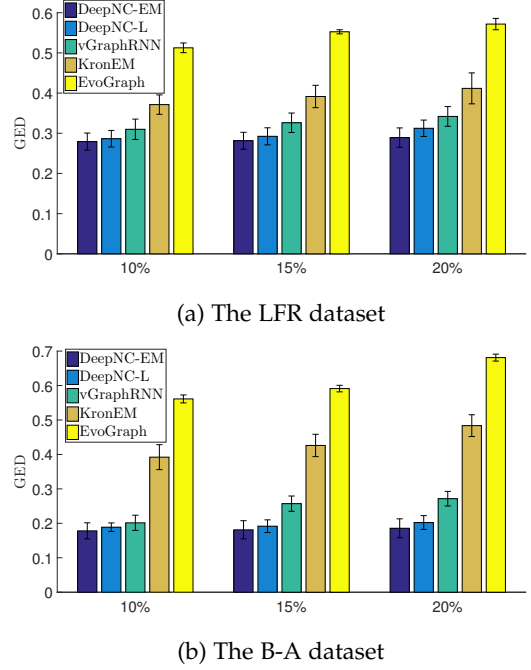


Fig. 9: Performance comparison in terms of GED (the lower the better), where the degree of missingness in edges between nodes in G_O is set to $\{10, 15, 20\}\%$.

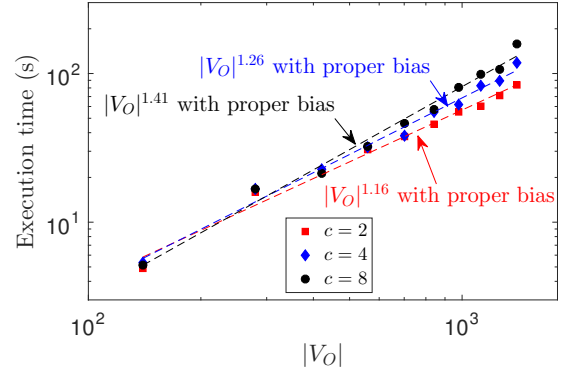


Fig. 10: The computational complexity of DeepNC-EM, where the log-log plot of the execution time versus $|V_O|$ is shown.

over edges via an RNN-based generative graph model by using structurally similar graphs as training data and then inferring the missing parts of the network by applying an imputation strategy that restores the missing data. Furthermore, we proposed two DeepNC algorithms whose run-time complexities are almost linear in $|V_O|$. Using various synthetic and real-world datasets, we demonstrated that our DeepNC algorithms not only remarkably outperform vGraphRNN, KronEM, and EvoGraph methods, but are also robust to many difficult and challenging situations that often occur in real environments such as 1) a significant portion of unobservable nodes, 2) training graphs that are only partially observable, or 3) a large fraction of missing edges between nodes in the observed network. Additionally, we analytically and empirically showed the scalability of

our DeepNC algorithms.

Potential avenues of future research include the design of a unified framework for improving the performance of various downstream mining and learning tasks such as multi-label node classification, community detection, and influence maximization when DeepNC is adopted in partially observable networks. Here, the challenges lie in task-specific preprocessing that should be accompanied by network completion to guarantee satisfactory performance in each individual task.

ACKNOWLEDGMENTS

This research was supported by the Republic of Korea's MSIT (Ministry of Science and ICT), under the High-Potential Individuals Global Training Program (No. 2020-0-01463) supervised by the IITP (Institute of Information and Communications Technology Planning Evaluation), by a grant of the Korea Health Technology R&D Project through the Korea Health Industry Development Institute (KHIDI), funded by the Ministry of Health & Welfare, Republic of Korea (HI20C0127), and by the Yonsei University Research Fund of 2020 (2020-22-0101).

REFERENCES

- [1] G. Kossinets, "Effects of missing data in social networks," *Soc. Netw.*, vol. 28, no. 3, pp. 247–268, Jul. 2006.
- [2] A. Acquisti, L. Brandimarte, and G. Loewenstein, "Privacy and human behavior in the age of information," *Science*, vol. 347, no. 6221, pp. 509–514, Jan. 2015.
- [3] R. Dey, Z. Jelveh, and K. Ross, "Facebook users have become much more private: A large-scale study," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Worksh.*, Lugano, Switzerland, Mar. 2012, pp. 346–352.
- [4] J. H. Koskinen, G. L. Robins, P. Wang, and P. E. Pattison, "Bayesian analysis for partially observed network data, missing ties, attributes and actors," *Soc. Netw.*, vol. 35, no. 4, pp. 514–527, Oct. 2013.
- [5] M. Kim and J. Leskovec, "The network completion problem: Inferring missing nodes and edges in networks," in *Proc. 2011 SIAM Int. Conf. Data Mining (SDM '11)*, Mesa, AZ, USA, Apr. 2011, pp. 47–58.
- [6] C. Tran, W.-Y. Shin, and A. Spitz, "Community detection in partially observable social networks," *arXiv preprint arXiv:1801.00132*, 2017.
- [7] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.
- [8] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *J. Molecular Bio.*, vol. 330, no. 4, pp. 771–783, Jul. 2003.
- [9] A. L. Traud, E. D. Kelsic, P. J. Mucha, and M. A. Porter, "Comparing community structure to characteristics in online collegiate social networks," *SIAM Rev.*, vol. 53, no. 3, pp. 526–543, Aug. 2011.
- [10] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *Proc. Int. Conf. Machine Learning (ICML '18)*, Stockholm, Sweden, Jul. 2018, pp. 5694–5703.
- [11] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *Proc. Int. Conf. Machine Learning (ICML '18)*, Stockholm, Sweden, Jul. 2018, pp. 609–618.
- [12] A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE Trans. Syst. Man Cybernetics*, vol. SMC-13, no. 3, pp. 353–362, Jun. 1983.
- [13] P. Erdos and A. Rényi, "On random graphs I," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.
- [14] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.
- [15] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *J. Mach. Learning Res.*, vol. 11, pp. 985–1042, Feb. 2010.
- [16] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel, "Efficient graph generation with graph recurrent attention networks," in *Proc. Advances Neural Inf. Processing Syst. (NIPS '19)*, Vancouver, Canada, Dec. 2019, pp. 4257–4267.
- [17] M. Simonovsky and N. Komodakis, "GraphVAE: Towards generation of small graphs using variational autoencoders," in *Proc. Int. Conf. Artificial Neural Netw. Machine Learning (ICANN '18)*, Rhodes, Greece, Oct. 2018, pp. 412–422.
- [18] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *NIPS Worksh. Bayesian Deep Learning*, Montréal, Canada, Dec. 2018.
- [19] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Proc. Advances Neural Inf. Processing Syst. (NIPS '18)*, Montréal, Canada, Dec. 2018, pp. 6410–6421.
- [20] D. Zhou, L. Zheng, J. Xu, and J. He, "Misc-GAN: A multi-scale generative model for graphs," *Front. Big Data*, vol. 2, pp. 3:1–3:10, Apr. 2019.
- [21] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.
- [22] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Phys. A: Stat. Mech. Appl.*, vol. 390, no. 6, pp. 1150–1170, Mar. 2011.
- [23] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. Advances Neural Inf. Processing Syst. (NIPS '18)*, Montreal, Canada, Dec. 2018, pp. 5165–5175.
- [24] P. Jain, R. Meka, and I. S. Dhillon, "Guaranteed rank minimization via singular value projection," in *Proc. Advances Neural Inf. Processing Syst. (NIPS '10)*, Vancouver, Canada, Dec. 2010, pp. 937–945.
- [25] J. P. Haldar and D. Hernando, "Rank-constrained solutions to linear matrix equations using powerfactorization," *IEEE Signal Process. Lett.*, vol. 16, no. 7, pp. 584–587, Jul. 2009.
- [26] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Proc. Advances Neural Inf. Processing Syst. (NIPS '17)*, Long Beach, CA, Dec. 2017, pp. 3697–3707.
- [27] C. Gentile, S. Li, and G. Zappella, "Online clustering of bandits," in *Proc. Int. Conf. Machine Learning (ICML '14)*, Beijing, China, Jun. 2014, pp. 757–765.
- [28] K. Mahadik, Q. Wu, S. Li, and A. Sabne, "Fast distributed bandits for online recommendation systems," in *Proc. 34th Int. Conf. Supercomput. (ICS '20)*, Las Vegas, NV, Jun.-Jul. 2020, pp. 1–13.
- [29] N. Linial, E. London, and Y. Rabinovich, "The geometry of graphs and some of its algorithmic applications," *Combinatorica*, vol. 15, no. 2, pp. 215–245, 1995.
- [30] A. K. Menon and C. Elkan, "Link prediction via matrix factorization," in *Proc. European Conf. Machine Learning Knowl. Disc. Databases (ECML PKDD '11)*, Athens, Greece, Sep. 2011, pp. 437–452.
- [31] R. Eyal, A. Rosenfeld, S. Sina, and S. Kraus, "Predicting and identifying missing node information in social networks," *ACM Trans. Knowl. Disc. Data*, vol. 8, no. 3, pp. 14:1–14:35, Jun. 2014.
- [32] S. Sina, A. Rosenfeld, and S. Kraus, "Solving the missing node problem using structure and attribute information," in *Proc. 2013 IEEE/ACM Int. Conf. Advances Social Netw. Analysis Mining (ASONAM '13)*, Niagara Falls, Canada, Aug. 2013, pp. 744–751.
- [33] H. Park and M.-S. Kim, "EvoGraph: An effective and efficient graph upscaling method for preserving graph properties," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining (KDD '18)*, London, United Kingdom, Aug. 2018, pp. 2051–2059.
- [34] T. H. McCormick, M. J. Salganik, and T. Zheng, "How many people you know?: Efficiently estimating personal network size," *J. Am. Stat. Assoc.*, vol. 105, no. 489, pp. 59–70, Sep. 2010.
- [35] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *Proc. Deep Learning and Representation Learning Worksh.*, Montreal, Canada, Dec. 2014.
- [36] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
- [37] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *J. Royal Stat. Soc. Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.
- [38] M. E. Newman, "Random graphs as models of networks," *Proc. National Acad. Sci.*, vol. 99, no. 1, pp. 2566–2572, 2002.

- [39] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Phys. Rev. E*, vol. 80, no. 1, pp. 016118:1–016118:8, Apr. 2009.
- [40] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: On approximating graph edit distance," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 25–36, Aug. 2009.
- [41] A. Fischer, K. Riesen, and H. Bunke, "Improved quadratic time approximation of graph edit distance by combining Hausdorff matching and greedy assignment," *Pattern Recogn. Lett.*, vol. 87, pp. 55–62, Feb. 2017.
- [42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learning Rep. (ICLR '15)*, San Diego, CA, May 2015.
- [43] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining (KDD '06)*, Philadelphia, PA, USA, Aug. 2006, pp. 631–636.