A Deep Generative Model for Graphs: Supervised Subset Selection to Create Diverse Realistic Graphs with Applications to Power Networks Synthesis

Mahdi Khodayar, Student Member, IEEE, Jianhui Wang, Senior Member, IEEE, and Zhaoyu Wang, Member, IEEE

Abstract—Creating and modeling real-world graphs is a crucial problem in various applications of engineering, biology, and social sciences; however, learning the distributions of nodes/edges and sampling from them to generate realistic graphs is still challenging. Moreover, generating a diverse set of synthetic graphs that all imitate a real network is not addressed. In this paper, the novel problem of creating diverse synthetic graphs is solved. First, we devise the deep supervised subset selection (DeepS3) algorithm; Given a ground-truth set of data points, DeepS3 selects a diverse subset of all items (i.e. data points) that best represent the items in the ground-truth set. Furthermore, we propose the deep graph representation recurrent network (GRRN) as a novel generative model that learns a probabilistic representation of a real weighted graph. Training the GRRN, we generate a large set of synthetic graphs that are likely to follow the same features and adjacency patterns as the original one. Incorporating GRRN with DeepS3, we select a diverse subset of GRRN’s generated graphs that best represent the behaviors of the real graph (i.e. our ground-truth). We apply our model to the novel problem of power grid synthesis, where a synthetic power network is created with the same physical/geometric properties as a real power system without revealing the real locations of the substations (nodes) and the lines (edges), since such data is confidential. Experiments on the Synthetic Power Grid Data Set show accurate synthetic networks such data is confidential. Experiments on the Synthetic Power Grid Data Set show accurate synthetic networks that follow similar structural and spatial properties as the real power grid.

Index Terms—Graph Modeling, Generative Learning, Deep Neural Networks, Subset Selection, Graphs

I. INTRODUCTION

Graphs are natural data representations in various domains including social networks, physical interactions, chemical and molecular structures, and power grids. Learning the distribution of a graph helps data science to find new configurations that share similar local and global properties. As a result, various graph representation learning models have been recently employed for a wide range of applications such as drug discovery [1], finding graph-based semantic representations of natural sentences [2], and Bayesian structure learning in graphical models [3].

The literature of probabilistic graph generating algorithms consists of two general categories; the first class of such approaches generate graphs based on strong a priori assumptions on the graph structures. In this line of research, [4] and [5] assign probabilities to large classes of graphs using robust techniques that work based on random graph modeling. A significant drawback of these approaches is the strong independence assumption on graph variables. Moreover, they can merely capture specific properties such as the degree distributions and diameter; hence, they are not able to discriminate graphs with small structural differences. As a result, such approaches cannot be employed for richly structured applications such as learning molecular structures or natural language representation. Also, in this category, [6] shows the merit of Kronecker graphs that generally follow common real-world network properties. Motivated by this fact, a fast algorithm is devised to model networks and mimic their behaviors. In addition, [7] provides a comprehensive review on random, small-world and scale-free networks and discuss their applications in graph modeling. Furthermore, [8] introduces mixed membership stochastic blockmodels that combine global network parameters showing dense patches of connectivity (blockmodel) with local parameters representing nodal variability in the connections (mixed membership). In the same class of methods, [9] introduced exponential random graphs in which the possible ties among the nodes of a social network are regarded as random variables, and assumptions on the variable dependencies decide the general form of the graph.

The second group of probabilistic algorithms that are more expressive is based on graph grammar which is a generalization of formal language theory for modeling non-sequential structures [10]. Using graph grammars, a graph can be derived sequentially through a chain of transformations on an intermediate graph. Such models can have a stochastic nature; however, their learnability is not justified. In fact, reasoning about the structure building operations potentially applied to create graphs is algorithmically hard [11]; hence, inducing graph grammars while observing a set of unannotated graph structured data is non-trivial [12]. Recent advances in machine learning
introduced generative architectures for various complex domains including computer vision and natural language processing. These approaches are basically based on two deep learning models, variational auto-encoders (VAEs) \cite{13} and generative adversarial networks (GANs) \cite{14}. In \cite{15}, a graph generating VAE is proposed; however, this model is merely capable of generating small graphs due to the large memory requirements of VAE. Also, \cite{12} introduced sequential graph generation processes that show better performance compared to long short-term memory networks (LSTMs). Moreover, \cite{16} presents a deep auto-regression algorithm to create graphs in a sequential order. At each iteration of the LSTM, the maximum likelihood adjacency is found to update an intermediate graph; hence learning the structure of the underlying data.

A. Proposed Algorithm

In this paper, a novel deep generative algorithm is proposed for the problem of diverse realistic graph generation. Our objective is to learn the nodal/edge distributions of an observed real-world network and generate a variety of synthetic graphs with the same structural properties. There are two major challenges addressed simultaneously: 1- Generating synthetic graphs that follow the same spatial/physical behavior as the original graph/network. 2- Increasing the diversity of solutions by selecting a diverse subset of synthetic graphs that are most informative about the original graph/network.

First, we define the supervised dissimilarity-based sparse subset selection (supervised DS3) as a novel problem in machine learning, where given a set of data points (items) inside any arbitrary high dimensional space, and a set of ground-truth items, we select a sparse subset of those items as close as possible to the ground-truth set. We propose the deep supervised DS3 (DeepS3) algorithm in order to solve this problem. The idea is to learn a deep embedding space for the set of data points such that the items with similar characteristics would lie in the same linear subspace while dissimilar items fall into distinct subspaces. Moreover, we provide theoretical guarantees to mathematically justify that under certain conditions the embedding space would cause our subset selection model to pick diverse elements that are as close as possible to the ground-truth set. Defining a set of novel loss functions, we ensure that our DeepS3 algorithm complies with such conditions; hence, learning to effectively solve the supervised DS3.

Then, we present our novel graph representation neural network (GRNN). This deep recurrent model creates the synthetic nodes and edges in a sequential order. At each time instance, a new node is generated by a “node-level” Gated Recurrent Unit network (GRU) that maintains the state of the nodes in the intermediate graph created so far. Subsequently, a new edge is added to our intermediate graph, based on the current state of the nodes and edges stored in an “edge-level” GRU. Alternating between the node-level and edge-level GRUs iteratively, a realistic graph is synthesized through multiple iterations based on the learned probability densities of nodal features and edge features captured by the GRUs.

Finally, we cast the problem of diverse graph synthesis as a supervised DS3 problem. Given a set of graphs generated by our GRNN, we select a sparse subset of them that are as close as possible to the real graph (i.e., our ground-truth); hence, learning to generate graphs that follow the same spatial and structural patterns as the realistic graph/network in the dataset while finding a diverse collection of synthetic graphs that are theoretically guaranteed to follow the same behavior as the original ground-truth graph.

B. Contributions

Current research on graph generative modeling and subset selection show significant drawbacks discussed in the following that motivate our presented work:

1) Scalability and Flexibility: In order to create an n-node graph, a generative algorithm requires to determine the values of n^2 random variables each corresponding to an entry of the corresponding adjacency matrix. Also, notice that the variability of number of nodes and edges adds further burden to the time and memory complexity of the generative algorithms. Therefore, even the deep learning based approaches in this area can merely work with small graphs sizes and cannot accommodate such complexities.

2) Representation Complexity: Previous works seek to learn a representation for each graph observed by the model; however, in practice, there exist many different configurations of nodes and adjacency matrices that can lead to a realistic graph observed in the training set. Therefore, in contrast to previous methods, we seek to learn the probability distribution densities of nodal and edge features rather than learning unique representations for a fixed set of nodes. Notice that in our general case, an n-node graph can have n! different adjacency configurations each corresponding to a different node ordering. Also, in contrast to previous works, here the edges are associated with features/properties as in the real-world cases (e.g. in power networks each edge represents the reactance between the corresponding nodes (i.e., buses)). This leads to highly complex representations; hence, the computational burden of such representations is a challenging factor that avoids previous generative algorithms to be applicable in practice for real-world large scale networks including power grids and social networks.

3) Structural Dependencies: In most realistic networks, the adjacency formations convey important information about the structural dependencies of the nodes. As an example, in a distribution power network, a pair of nodes (buses) with a common neighbor are likely to have no edges. Moreover, in a transmission power network, the density of the nodes in a region has positive correlation with the number of edges. While previous generative models neglect such complex dependencies by considering edges as a sequence of independent variables, we
devise a recurrent generative algorithm that builds new nodes/edges based on the current nodes/edges created so far in the intermediate steps of graph synthesis.

4) Diversity of Outputs: While generative models create unique synthesized graphs, in practice, various scientific and experimental applications require a diverse set of synthetic graphs that all follow similar structural and spatial behaviors as the graphs/networks in the dataset. For instance, in the case of power network synthesis, such diversity in the created power networks would help researchers view different feasible scenarios and configurations of real power systems. However, current algorithms are incapable of providing various distinct synthetic outputs due to their lack of graph representation complexities and lack of a diverse subset selection method for graphs.

C. Application: Power Network Synthesis

In recent years, improving power system performance and resilience has been a crucial challenge associated with various aspects of power grids including the incorporation of renewable energy [17], robust power network monitoring [18], transmission expansion planning [19], and vulnerability analysis and control [20]. Addressing such challenges require the realistic power transmission network topologies with real geographical coordinates and real physical data corresponding to the substations (nodes) and lines (edges) of such networks. However, the data of real power grids is generally confidential in order to avoid exposing the vulnerabilities of power systems. There have been several studies on the structural properties of power networks [21-22]. These works focus on a small set of graph properties including average degree, average path length, as well as the degree distributions. Moreover, some works suggested certain classes of graphs that best represent a power grid with respect to several topological properties. For instance, [23] makes a very strict assumption that the generators supply power to all distribution substations, and that the electricity always follows the most efficient path between the substations. Also, [24] considers similar assumption for an ideal network where each distribution substation can receive power from any generator in the system. Moreover, it is assumed that the cumulative degree distribution follows certain distribution function.

In very recent literature, several works have tackled the problem of power grid synthesis. [25] applied a naive linear regression model to estimate the total number of loads with respect to the total number of ZIP codes at each region. Moreover, the transmission lines where created manually and the generators and loads where placed based on a nearest neighborhood assumption rather than the geographical properties of the real network. Also, [26] follows a similar approach using the location of cities and power plants to generate the synthetic power grid of Texas. [27] and [28] did not provide any comparisons in terms of performance or topological behaviors with the real power network. In fact, these works did not present a generalized data-driven approach to address network synthesis for arbitrary power networks. In this line of research, [27] has applied Gaussian mixture model (GMM) to learn the coordination of the substations (i.e. nodes) and generate similar nodes. However, the lines (i.e. edges) are synthesized using very strict (and not necessarily true) assumptions on the node degrees, rather than being learned by a data-driven model. As the edge patterns are not learned and the edges are created manually, the presented work in [27] does not closely follow the real power network.

In this paper, we employ our novel deep learning based graph generative model to learn highly complex spatial and structural relationships between the substations (nodes) and lines (edges) in the real power network. Unlike previous works, here the model automatically learns the joint distribution of nodes and edges, hence, can generate network structures that imitate the same physical and spatial characteristics as a large-scale real power grid. Moreover, in contrast to existing works, our model is capable of generating a diverse set of power grids with similar physical behaviors rather than a single network.

This paper is organized as the following: Section II discusses the classic subset selection problem. In Section III, we introduce the novel problem of supervised S3, and devise DeepS3 as a deep learning solution; Moreover, we theoretically guarantee that the proposed algorithm solves supervised S3. In Section IV, we introduce the supervised S3 problem for the graph-structured datasets and devise a solution using DeepS3 and GRRN. Section V explains our power network synthesis algorithm based on DeepS3, GRRN, and statistical inference. Section VI shows the experiments on a publicly available large-scale power network dataset. Finally, the conclusions and future works are presented in Section VII.

II. Subset Selection

Our proposed DeepS3 framework builds on the dissimilarity based Subset Selection [28-29] reviewed in this section. Given a ground set $D = \{x_i\} i = 1, 2, ..., N$, the objective of the Subset Selection task is to choose a small set of items (data points) $S \subseteq D$ as the representatives, that can capture the major mathematical characteristics of the entire set $D$. Let $d_{ij}$ show some dissimilarity measurements between two items $x_i$ and $x_j$, that is, $d_{ij}$ shows how well an item $x_j$ is represented by $x_i$. Notice that, if an item $x_k$ exists such that $d_{kj} < d_{ij}$, then $x_k$ would better capture the characteristics of $x_j$ compared to $x_i$. Assuming that the dissimilarity metric $d$ is non-negative, a partial ordering exists for the data points in $D$. Moreover, we assume $\forall x_i, x_j d_{jj} < d_{ij}$, that is, each item $x_j$ is the most optimal representative for its own data content compared to all other items. The objective is to find a diverse set of items $S$ that well represents our ground set $D$ given the pairwise dissimilarities $\{d_{ij}\} i, j = 1, 2, ..., N$. 
A. Facility Location Objective

In order to solve the Subset Selection problem, the facility location objective chooses a subset of items $S \subseteq D$ such that the data points $x_i \notin S$ are assigned to one and only one item in $S$ whose distance is the smallest among the data points in the selected subset. Now, the objective would be to find the optimal encoding and sufficient cardinality (of course, the cardinality cannot be too large or too small depending on the problem domain).

The unsupervised optimization corresponding to the objective of facility location is shown as the following:

$$\min_{S \subseteq D} \sum_{j: x_j \in D} \min_{i: x_i \in S} d_{ij} + \lambda |S|$$

Notice that, to achieve a good encoding, $d_{ij}$ is minimized for encoding the data point $x_j$ by $x_i$ inside $S$. Moreover, the regularization term $\lambda |S|$ with parameter $\lambda > 0$ is added to the encoding objective, in order to obtain a trade-off between the quality of encoding and the number of representative data points. Notice that, when no regularization exists, i.e. $\lambda = 0$, a trivial solution would be $S = D$ where each item is represented by itself.

B. Dissimilarity-based Sparse Subset Selection

The problem of finding the optimal $S$ in (1) is NP-hard; therefore, some attempts are made to obtain an approximation for $S$. DS3 is a recent algorithm that equivalently solves the facility location optimization as an integer binary optimization using a set of binary assignment variables. In this approach, a set of binary assignments are defined as $z_{ij}$, $i, j$ s.t. $x_i, x_j \in D$. Each variable $z_{ij} \in \{0, 1\}$ is associated with the distance $d_{ij}$ and is equal to 1 when $x_j$ is represented by $x_i$ and zero otherwise. Hence, the binary representation vector $r_{v_i} = [z_{i1} z_{i2} \ldots z_{iN}]$ is non-zero if the $i$-th data point in $D$ is a representative for some item, and is zero if this data point is represented by some item in $D - \{x_i\}$. Therefore, one can rewrite (1) using:

$$\min_{\{z_{ij}\}} \sum_{j: x_j \in D} \sum_{i: x_i \in D} d_{ij} z_{ij} + \lambda \sum_{i: x_i \in D} I(||r_{v_i}||_p)$$

$$s.t. \ z_{ij} \in \{0, 1\}, \ \sum_{i=1}^{N} z_{ij} = 1, \ \forall i, j: x_i, x_j \in D$$

Here, $I(.)$ is the indicator function, with output value 1 if the argument is non-zero, and zero otherwise. Notice that the regularization term of DS3 in (2) counts the number of representative items, while the first term minimizes the encoding cost. Furthermore, the condition ensures that each item is represented by only a single representative data point.

The optimization in (2) is still an NP hard problem equivalent to (1); hence, in order to efficiently solve the facility location problem, the DS3 computes the convex relaxation of the problem by dropping $I(.)$, and relaxing the binary constraints on $z_{ij}$ variables to $z_{ij} \in [0, 1]$. Therefore, the following simultaneous sparse recovery problem is solved:

$$\min_{\{z_{ij}\}} \sum_{j: x_j \in D} \sum_{i: x_i \in D} d_{ij} z_{ij} + \lambda \sum_{i: x_i \in D} I(||r_{v_i}||_p)$$

$$s.t. \ z_{ij} \in [0, 1], \ \sum_{i=1}^{N} z_{ij} = 1, \ \forall i, j: x_i, x_j \in D$$

Here, the optimization is convex for $p \geq 1$. Notice that a soft assignment can be obtained by $p = 2$, which leads to $z_{ij} \in [0, 1]$. In this study, we choose $p = \infty$ in order to capture a hard assignment, i.e. binary $z_{ij}$ values.

Solving the convex optimization in (3) leads to a set of representatives $R_j = \{x_i | z_{ij} > 0\}$ for each item $x_j \in D$, resulting in the total representative set $R = \bigcup_{j: x_j \in D} R_j$. In fact, the solution induces a clustering of the data points with respect to the assignments of items to each representative. More specifically, for each representative $x_i \in R$, a cluster is defined as $A_i = \{j: x_j \in D \ s.t. \ z_{ij} > 0\}$ which contains all the items assigned to $x_i$.

III. Supervised Subset Selection

In this section, our new problem of supervised Subset Selection is introduced. Then, this problem is cast as a DS3 problem. We propose a deep metric learning algorithm, Vanilla DeepS3 (vDeepS3), to tackle the supervised DS3. Then, novel supervised loss functions are devised based on our theoretical guarantees to make sure that DS3 would choose the items in a supervised fashion; that is, the selected items $S$ are according to the supervised information obtained from user preferences or the problem domain. Our novel supervised Subset Selection algorithm, DeepS3, is the evolved version of our Vanilla DeepS3 model that makes use of the presented supervised loss functions.

A. Problem Definition

Given a ground set $D = \{x_i\} i = 1, 2, ..., N$ as the universe of all items, and a ground-truth set $\Psi \subseteq D$ as a selection of representative items chosen by some user or some constraints/criteria in a real-world problem domain, the objective is to learn a machine learning model that can choose a subset of items $S \subseteq D$ as close as possible to $\Psi$ while preserving the diversity of the selected items in $S$.

B. Vanilla DeepS3 (vDeepS3)

We cast the supervised problem as the dissimilarity-based sparse Subset Selection. A fundamental challenge is to learn a distance metric $d_{ij}, x_i, x_j \in \mathbb{R}^d$ to reflect the dissimilarities required for the DS3. Computing $d_{ij}$ introduces a $m < d$ dimensional embedding space in $\mathbb{R}^m$, where similar items would live in the vicinity of each other while dissimilar items would be further. This requires a representation learning algorithm that observes many similar/dissimilar training example pairs of items, in order to understand the notion of similarity/dissimilarity.
in the ground set $D$. In some problems such as object detection [30] or image classification/categorization [31], lots of similar/dissimilar example pairs retrieved from large datasets are available; however, in the case of our problem, we assume limitations on the number of data samples. Therefore, our algorithm should learn such an embedding space in an efficient way.

To define the notion of similarity/dissimilarity, here, the objective is to learn a data representation function $R_\theta : X \rightarrow R_\theta(X)$ with parameters $\theta$. We write $R_\theta$ as $R$ to simplify our notations. As shown by Fig. 1, this function maps any $d$ dimensional data point $x_i \in \mathbb{R}^d$ to some feature vector $R(x_i) \in \mathbb{R}^m$ with $m < d$, such that applying DS3 on the new transformed ground set $D' = \bigcup_i \{R(x_i)\}$ would lead to selecting a subset $S' = \{R(x_i)\mid x_i \in \Psi\}$ by the DS3 algorithm inside the embedding space, hence, resulting in the selection of $S = \{x_i\mid R(x_i) \in S'\} = \Psi$ by the supervised DS3 algorithm in the original space of $D$. The $L_2$ norm in the optimal target space corresponding to the optimal $R_\theta(.)$ estimates the “semantic” distances $d_{ij} 1 \leq i,j \leq n$ required to optimize (3) in such a way that the resulting subset $S$ is equal to the ground-truth set $\Psi$. In other words, $R_\theta(.)$ should be trained in such a way that (3) obtains the solution $S = \Psi$.

![Fig. 1: Mapping the data points $x_i \in D$ inside the highly nonlinear original space to the new data points $R_\theta(x_i) \in D'$ inside the embedding space. The data points with red outline in the original space are selected by the ground-truth set $\Psi$ while the items with red outline in the embedding space are the items in subset $S'$ chosen by DS3 using (3). Ideally, the items in $S'$ are the transformed versions of the items in $\Psi$ using the transformation $R_\theta$. Therefore, we have $S = \Psi$ in the ideal case inside the original space.](image)

Algorithm 1 is the pseudocode of vDeepS3 that computes the optimal representation function $R_\theta^*(.)$, the optimal regularizer $\lambda^*$, as well as the optimal $Z$ matrix denoted by $Z^*$ that shows the selected representative items. Here, $K_P$ and $K_N$ are respectively the numbers of similar and dissimilar pairs of items created to learn the distance metric $d_{ij}$ via learning $R_\theta(.)$. The presented approach is an iterative algorithm that updates $\theta(t)$ at each iteration $t$ until convergence. At each iteration, first, all ground-truth and non-ground-truth items are mapped by $R_\theta(.)$ into the embedding space. Then, the $L_2$ norm $d(x_i, x_j; \theta(t))$ is computed for all pairs $<x_i, x_j> \in D \times D$. The binary decision variable matrix $Z = [z_{ij}]_{1 \leq i,j \leq N}$ is then estimated by solving (3) for the fixed regularizer $\lambda$. We define two types of clustering:

1) Z-clustering $ZC = \{C^Z_k\} k = 1, 2, ..., N_Z$ where each cluster $C^Z_k$ is corresponding to a representative item $x_k \in S$ and is defined by $C^Z_k = \{x_i \in D|z_{ki} \neq 0\}$, that is, the cluster $C^Z_k$ is a group of items represented by the representative $x_k$. Notice that the number of clusters $N_z = |ZC|$ is equal to the number of non-zero rows in $Z$, i.e. the number of representatives captured in (3) for a given $\lambda$.

2) Ground-truth clustering $GC = \{C^G_k\} k = 1, 2, ..., N_\Psi$ where each cluster $C^G_k$ is defined as $C^G_k = \{x_i \in D|\forall x_j \in \Psi j \neq k : d_{ik} < d_{ij}\}$, that is, a cluster $C^G_k$ contains all elements whose distance is closest to $x_k \in \Psi$ compared to the other elements of $\Psi$. Ideally, the objective is to have $ZC = GC$ so that (3) would result in clustering the data points where in each cluster, a ground-truth item represents the members.

At each iteration, $ZC$ and $GC$ are computed; then, $\lambda$ is increased by $\varepsilon$ if $N_z > N_\Psi$ and decreased otherwise in order to hold the same number of representatives in $S$ as the number of ground-truth data points in $\Psi$. As we aim to learn the distance metric $d_{ij}$ for items of $D$ in a supervised fashion, we define $p_m = <x_i, x_j, y>$ as a training example where $x_i, x_j \in D$ and $y \in \{0, 1\}$ is the binary label showing whether the two data points make a positive example (i.e. similar pair of items with $y = 1$) or a negative one (i.e. dissimilar pair of items with $y = 0$). Given $\theta(t)$, we define a contrastive loss on our positive/negative (similar/dissimilar) example $pn$ computed as:

$$l_c(p_m; \theta) = y \cdot d(x_i, x_j; \theta) + (1 - y) \cdot \max(0, m - d(x_i, x_j; \theta))$$

$$\text{with } d(x_i, x_j; \theta) = ||R_\theta(t)(x_i) - R_\theta(t)(x_j)||_2$$

For a positive pair, $l_c$ punishes $R_\theta(t)$ when similar data points are mapped to distant regions of the embedding space, while for a negative pair, $l_c$ punishes $R_\theta(t)$ when the data points are closer than a margin $m > 0$ in the embedding space.

As shown in Algorithm 1, vDeepS3 creates positive examples $p_m^k = <x_i^k, x_j^k, y^k = 1 > 1 \leq k \leq K_P$ where $x_i^k$ and $x_j^k$ are randomly chosen from the same $GC$ clusters. In addition, negative examples $p_m^k = <x_i^k, x_j^k, y^k = 0 > 1 \leq k \leq K_N$ are added to the training set, in which $x_i^k$ and $x_j^k$ are picked randomly from two distinct clusters. Applying (4), the set of examples are used by a gradient descent method with learning rate $\eta$ to train $R_\theta(.)$. At each iteration, both $\lambda$ and $\theta$ are updated so that the Z-clustering would converge to the ground-truth clustering. A stopping criteria is defined as $\Delta \theta(t) < 10^{-2} \theta(t)$ for the update of $R_\theta(.)$.

C. Deep Supervised Subset Selection (DeepS3)

As discussed in Section III-B, the vDeepS3 learns an embedding space by which our DS3 algorithm can cluster the ground set in such a way that each $C^Z_k$ is associated with a ground-truth item. Algorithm 1 discriminates the data points according to their distance to their corresponding
In fact, focusing on the optimizations in (2) and (3), we mathematically guarantee that our vDeepS3 would select $\Psi$ when our presented theoretical conditions hold. Furthermore, we enforce such conditions by introducing two novel supervised error functions on top of the contrastive loss function in line 19 of Algorithm 1. We present DeepS3 as a version of our vDeepS3 that is theoretically guaranteed to find the optimal subset with respect to the information obtained from $\Psi$. The theoretical guarantees are explained in the following.

1) Medoid Theoretical Guarantee: Let us analyze the properties of the solution of the non-convex optimization problem in (2). In fact, this is the solution one would like to achieve solving (3) by vDeepS3. The following theorem introduces a crucial condition for vDeepS$3$ when solving (2):

**Theorem 1** The ideal solution of the vDeepS3 algorithm should satisfy the following condition: For every $x_i \in D$ where $x_i \not\in \Psi$, with the embedding parameter $\theta$ and the ground-truth item $x_j \in \Psi$ s.t. $x_j \in C_{j}^{\Psi}$, we have:

$$\sum_{x_i \in C_{j}^{\Psi}} d(x_i, x_j; \theta) \leq \sum_{x_i \in C_{j}^{\Psi}} d(x_i, x_i; \theta)$$

That is, for any ground-truth cluster $C_{j}^{\Psi}$ such that $x_i \in C_{j}^{\Psi}$, in order for the corresponding ground-truth $x_j$ to be a representative item, the representation $R_\theta(x_j)$ should be the medoid of the ground-truth cluster $C_{j}^{\Psi}$. That is, $x_j$ should cause the minimum encoding cost among other items in $C_{j}^{\Psi}$ when they are all mapped into the embedding space.

The non-convex problem definition in (2) results in the ideal solution for the general unsupervised Subset Selection problem. The solution to this non-convex optimization program requires each representative $x_j$ to have a small total distance to the items that are represented by it. In fact, this total distance should be smaller than the total distance from any data point $x_i \in D$ represented by $x_j$, to all the other items represented by $x_j$. This leads to Theorem 1 for our supervised vDeepS3 algorithm.

2) Non-convex Equivalence Theoretical Guarantee: Here, we study the conditions under which our vDeepS3 algorithm finds the ideal solution of the non-convex optimization program in (2):

**Theorem 2** The solution found by the vDeepS3 is equivalent to the ideal solution of (2) with $p = \infty$, if the following conditions hold:

1. For every $x_i \in D$ where $x_i \not\in \Psi$ is assigned to a cluster $C_k^{\Psi}$ with the embedding parameter $\theta$, and for every $x_j \in D$ in $C_k^{\Psi}$ we have:

$$\frac{\lambda}{|C_k^{\Psi}|} + d(x_j, x_k; \theta) \leq d(x_i, x_j; \theta)$$

2. For every $x_i \in D$ where $x_i \not\in \Psi$ is assigned to a cluster $C_k^{\Psi}$ with the embedding parameter $\theta$, and for every $x_j \in D$
in the same cluster, we have:

\[
\frac{\lambda}{|C_k^\Psi|} + d(x_j, x_k; \theta) \geq d(x_i, x_j; \theta) \tag{7}
\]

The first condition requires that the closest item from other clusters \(C_{k\neq k}^\Psi\) to a cluster with representative \(x_k \in C_k^\Psi\) stays sufficiently far from \(x_k\) in the embedding space defined by \(R(\cdot)\). In fact, \([6]\) is our first condition that causes the solution of \([3]\) to be equivalent to the solution of non-convex program in \([2]\). In addition, the second condition \([7]\) ensures that the data points in each cluster \(C_k^\Psi\) stay close enough to each other in the captured embedding space. More specifically, each item in \(C_k^\Psi\) is at most \(\lambda/|C_k^\Psi|\) away from its corresponding representative \(x_k\). \([6]\) and \([7]\) are sufficient conditions that imply the capability of the vDeepS3 algorithm to recover the ideal solution in \([2]\).

Now, we propose DeepS3 as a supervised Subset Selection algorithm theoretically guaranteed to find an optimal subset selected \(S\) which is equal to the ground-truth \(\Psi\). Here, to address Theorem 1, we devise a supervised error function, named as the medoid loss \(l_{med}\), computed by:

\[
l_{med}(GC, \Psi; \theta) = \frac{1}{|\Psi|} \sum_{k=1}^{c} \left[ \sum_{i: x_i \in C_k^\Psi} \frac{1}{|C_k^\Psi|} \left| R_\theta(x_i) - R_\theta(x_k) \right| \right]^2 \tag{8}
\]

Minimizing \(l_{med}\) would cause the representation of \(x_k\), i.e. \(R_\theta(x_k)\), move towards the medoid in \(C_k^\Psi\) as the \(L_1\) norm averaged over all \(i: x_i \in C_k^\Psi\) is decreased. Ideally, each data point in \(\Psi\) will be the medoid of its corresponding cluster to satisfy Theorem 1.

In order to address condition 1 in Theorem 2, we propose the Non-convex Equivalence loss function \(l_{NE_1}\) on the ground-truth clustering, computed by:

\[
l_{NE_1}(GC, \Psi; \theta) = \sum_{k=1}^{c} \sum_{i: x_i \in C_k^\Psi} \sum_{k \neq j: x_j \in C_k^\Psi} \max(\gamma_1 - d(x_i, x_j, \theta), 0) \tag{9}
\]

s.t. \(\gamma_1 = \frac{\lambda}{|C_k^\Psi|} + d(x_j, x_k, \theta)\)

Here, if the two data points \(x_i\) and \(x_j\) come from different clusters, and their corresponding distance is more than the margin \(\gamma_1\), the parameters \(\theta\) are not punished; however, if their distance is smaller than \(\gamma_1\), we punish \(\theta\) by \(\frac{\lambda}{|C_k^\Psi|} + d(x_j, x_k, \theta) - d(x_i, x_j, \theta)\) to compensate for the insufficient distance and move \(R_\theta(x_j)\) by at least \(\frac{\lambda}{|C_k^\Psi|}\) units further from \(R_\theta(x_i)\) as required. Moreover, we introduce the following loss function for DeepS3 to comply with condition 2 of Theorem 2:

\[
l_{NE_2}(GC, \Psi; \theta) = \sum_{k=1}^{c} \sum_{i: x_i \in C_k^\Psi} \sum_{j: x_j \in C_k^\Psi} \max(d(x_i, x_j, \theta) - \gamma_2, 0) \tag{10}
\]

s.t. \(\gamma_2 = \frac{\lambda}{|C_k^\Psi|} + d(x_j, x_k, \theta)\)

Here, if two items \(x_i\) and \(x_j\) have an embedding distance smaller than the margin \(\gamma_2\), \(\theta\) is not punished; on the other hand, if \(x_j\) is more than \(\frac{\lambda}{|C_k^\Psi|}\) units away from \(x_i\) compared to its representative \(x_k\), \([10]\) punishes \(\theta\) by \((d(x_i, x_j, \theta) - d(x_j, x_k, \theta)) - \frac{\lambda}{|C_k^\Psi|}\) units; hence, leading to sufficiently dense clusters as required by Theorem 2.

Incorporating the contrastive loss \([4]\) defined for the vDeepS3 with the devised loss functions \([8]-[10]\), the DeepS3 algorithm is designed by introducing the following total update rule for algorithm 1:

\[
\theta(t+1) = \theta(t) + \Delta \theta(t) + \rho_{med} \Delta \theta_{med}(t) + \rho_{NE_1} \Delta \theta_{NE_1}(t) + \rho_{NE_2} \Delta \theta_{NE_2}(t) \tag{11}
\]

where \(\Delta \theta_{med}(t), \Delta \theta_{NE_1}(t), \) and \(\Delta \theta_{NE_2}(t)\) are the magnitudes of parameter update computed based on the gradients of \([8], [9]\), and \([10]\) using stochastic gradient descent, respectively. The coefficients \(\rho_{med}, \rho_{NE_1}, \rho_{NE_2} \in \mathbb{R}\) determine the weight of the contribution of \([8], [9]\), and \([10]\) in the total loss, respectively. The DeepS3 algorithm is algorithm 1 using the update rule \([11]\) with non-zero loss coefficients instead of the contrastive loss.

IV. GRAPH DEEP SUPERVISED SUBSET SELECTION (GDEEPSS3): A DEEPSS3 FOR GRAPH DATASETS

In this section, the problem of graph DeepS3 is introduced. Then, in order to solve the problem using DeepS3, a graph representation learning model, Graph Representation Neural Network (GRNN), is designed to model \(R_\theta(\cdot)\) for graph-structured data. A generative learning algorithm is proposed for GRNN; hence, the proposed representation model is able to generate graphs with similar patterns as the graphs in its training set.

A. Problem Definition

Given a graph dataset \(G = \{G_i\} i = 1, 2, ..., n\) where each undirected weighted graph \(G_i = \langle V_i, E_i \rangle >\) with node set \(V_i\) and edge set \(E_i\) can have arbitrary number of nodes/edges, and a set of ground-truth graphs \(\mathcal{G} = \{G_i\} \subseteq \mathbb{G}\), the objective is to find a subset \(S \subseteq \mathcal{G}\) whose members have topological patterns and nodal/edge features as close as possible to the ground-truth set \(\mathcal{G}\).

We cast our GDeepS3 problem as a DeepS3 problem where each graph is a data point in \(D\), i.e. \(vi x_i = G_i\). The challenge would be to define the embedding function \(R_\theta(G_i)\) mapping each arbitrary graph \(G_i \in \mathcal{G}\) to some representation in the graph embedding space, such that the graphs with similar patterns would be close to each other in the captured embedding space while graphs with dissimilar patterns lie further.

B. Graph Representation Neural Network (GRNN)

Here, we propose GRNN to model \(R_\theta(G)\) for an arbitrary undirected weighted graph \(G = (V, E)\) with nodes
to be able to generate (create) leads to learning a generative model that can create

\[ F^\pi = f_F(G, \pi) = (F^\pi_1, F^\pi_2, ..., F^\pi_{|V|}) \text{ with } F^\pi_i \in \mathbb{R}^d \]  

(12)

Here, each \( F^\pi_i \) is a \( d \)-dimensional feature vector corresponding to node \( \pi(i) \); hence, \( F^\pi = f_F(G, \pi) \) is a tensor of nodal features captured from graph \( G \) with respect to the ordering \( \pi \).

Similarly, for the weighted adjacency entries, we define the function \( f_A \) to map \( (G, \pi) \) to a 2-dimensional tensor \( \Lambda^\pi \) calculated as:

\[ \Lambda^\pi = f_A(G, \pi) = (\Lambda^\pi_1, \Lambda^\pi_2, ..., \Lambda^\pi_{|V|}) \]  

with \( \Lambda^\pi_i = (A^\pi_{i,1}, A^\pi_{i,2}, ..., A^\pi_{i-1,i}) \), \( i = 2, ..., |V| \) \n
(13)

As shown in (13), \( \Lambda^\pi_i \) is a weighted adjacency vector for node \( \pi(i) \); thus, \( \Lambda^\pi = f_A(G, \pi) \) is the total feature tensor captured from the weighted edges of \( G \) with respect to the node ordering \( \pi \).

Notice that, for an undirected weighted graph, the feature \( \langle \Lambda^\pi, F^\pi \rangle \) defines a unique \( G \); hence, learning \( \langle \Lambda^\pi, F^\pi \rangle \) is equivalent to learning to create \( G \). In fact, a mapping \( f_g(\Lambda^\pi, F^\pi) = G \) recovers the graph \( G \). Therefore, in order to create a graph \( G \) observed in the dataset, one can maximize the probability of observing \( G \) by:

\[ \max \left[ P(G) = \sum_{\langle \Lambda^\pi, F^\pi \rangle} P(\Lambda^\pi, F^\pi) \mathbb{I}[f_g(\Lambda^\pi, F^\pi) = G] \right] \]  

\[ \mathbb{I}[x] = \begin{cases} 1 & x = \text{True} \\ 0 & \text{Otherwise} \end{cases} \]  

(14)

That is, the probability of observing a graph \( G \) is equivalent to the joint probability of observing nodal features and edge weights \( P(\Lambda^\pi, F^\pi) \). Hence, learning \( P(\Lambda^\pi, F^\pi) \)
leads to learning a generative model that can create \( G \).

In this section, we learn \( \langle \Lambda^\pi, F^\pi \rangle \) to represent \( G \) as a set of nodal/edge features. Moreover, we learn \( P(\Lambda^\pi, F^\pi) \) to be able to generate (create) \( G \) by sampling from \( P(G) \). Let us decompose \( P(F^\pi, \Lambda^\pi) \) in (14) using the product of conditional distributions, hence, we have:

\[ P(\Lambda^\pi, F^\pi) = \prod_{i=1}^{|V|} P(F^\pi_i | \Lambda^\pi_1, ..., \Lambda^\pi_{i-1}) P(\Lambda^\pi_i) \]  

(15)

Due to the recurrent structure of (15), in this study, in order to represent \( G \), we model \( P(F^\pi, \Lambda^\pi) \) by a recurrent neural network named as the Graph Representation Neural Network. The feed-forward algorithm as well as the training procedure of the proposed neural network are explained in the following subsections.

C. Feed-forward Algorithm for GRNN

Algorithm 2 shows the feed-forward computations of our Graph Representation Neural Network, while Fig. 2 depicts the structure of this neural network. Given a graph \( G = (V, E) \), the nodes and edges are observed one by one through multiple iterations. At each iteration \( i \), a Gated Recurrent Unit (GRU) neural network is trained for \( i \) rounds. As shown in Fig. 2 and (16), at each round \( 1 \leq j \leq i \), the input vector \( I(j) \) is computed by concatenating the tensors \( \Lambda^\pi_i(j) = A^\pi_{i,j} \) and \( \delta(j) = F^\pi_j \). Then, the temporal feature vectors, \( z(j) \) and \( r(j) \), are computed using sigmoid function \( \sigma \) with weights \( W_z \) and \( W_r \), respectively. The update value of the GRU state \( C(j) \) is computed using \( r(j), I(j) \), and the GRU state at round \( j - 1 \) denoted by \( C(j-1) \). The tangent hyperbolic function \( \tanh \) with weights \( W_r \) is used for this computation. Then, the GRU state (graph state) at round \( j \), i.e. \( C(j) \), is finally obtained by a weighted sum of the state at previous round \( C(j-1) \) and the update value \( \tilde{C}(j) \).

At each iteration \( i \), the graph state \( S_i \) is considered as the computed GRU state \( C(i) \) at the corresponding iteration. The graph state evolves as new nodes and edges are observed until the whole graph is traversed. Finally, the G’s representation vector, i.e. \( R_\theta(G) \), is considered to be the last graph state, i.e. \( S_{|V|} \), which contains the information of the whole nodes and edges of \( G \).

![Fig. 2: Structure of the proposed Graph Representation Neural Network](image)

D. Generative Training of GRNN’s parameters \( \theta \) =< \( W_z, W_r, W_C \) >

We train GRNN in a generative fashion; having a sequence of \( N \) graph-structured example data in a training
Algorithm 2: Graph Representation Neural Network feed-forward algorithm

Input: Graph $G = (V, E)$, and an ordering $\pi$
Output: Graph representation $R_\theta(G)$
1. Initialize time step $i = 1$
2. Define a GRU with input vector $I$, update gate vector $z$, reset gate vector $r$, and state vector $C$ with sigmoid ($\sigma$) and tangent hyperbolic (tanh) activation functions for graph state transitions. Parameters are: $\theta = < W_z, W_r, W_C >$ with update weight $W_z$, reset weight $W_r$, and state estimation weight $W_C$.
3. while $i \leq |V|$ do
   4. Initialize GRU’s internal counter $j = 1$
   5. $C(0) = S_{i-1}$
   6. while $j \leq i$ do
      7. $I(j) = [\Lambda^i_j, \delta(j) = F^i_j]$
      8. $z(j) = \sigma(W_z \cdot [C(j), I(j)])$
      9. $r(j) = \sigma(W_r \cdot [C(j), I(j)])$
     10. $\tilde{C}(j) = \tanh(W_C \cdot [r(j) \cdot C(j-1), I(j)])$
     11. $C(j) = (1 - z(j)) \cdot C(j-1) + z(j) \cdot \tilde{C}(j)$
     12. $j = j + 1$
   13. $S_i = C(i)$
   14. $i = i + 1$
15. return $R_\theta(G) = S_{|V|}$

Algorithm 3: Generative training of GRNN

Input: Graph dataset $TS = \{G_1, G_2, ..., G_N\}$, and number of graphs to generate $K_{gen}$
Output: Set of generated graphs $\hat{G} = \{\hat{G}_1, \hat{G}_2, ..., \hat{G}_{K_{gen}}\}$
1. Define $START$ and $END$ as starting and ending tokens for the GRU network, respectively.
2. Initialize training sample counter $n = 1$
3. while $n \leq N$ do
   4. Initialize generated graph counter $g = 1$
   5. while $g \leq K_{gen}$ do
      6. Initialize GRU state: $[\Lambda^0_0, \theta^0_0] = START$
      7. Initialize node counter $i = 1$
      8. while $[\Lambda^i_0, \theta^i_0] \neq END$ do
         9. Compute graph state $S_i$ using (16)
         10. $\theta^i_f = f^{mlp}_{\theta_f}(S_i)$
         11. $\theta^i_A = f^{mlp}_{\theta_A}(S_i)$
         12. $F^i_\pi \sim P_{\theta_f}\{\text{Sample features of } i \text{-th node}\}$
         13. $\Lambda^i_\pi \sim P_{\theta_A}\{\text{Sample weight of } i \text{-th edge}\}$
         14. $i = i + 1$
   15. Generate $\hat{G}_g$ having all features $F^i_\pi$ and edges $\Lambda^i_\pi$
   16. Append $\hat{G}_g$ to the set of generated graphs $\hat{G}$
   17. Maximize likelihood of $P(F^i_\pi, \Lambda^i_\pi)$ using gradient descent: For each generated graph $\hat{G}_g$, compute the error as the Euclidean distance between $(F^i_\pi, \Lambda^i_\pi)$ in $\hat{G}_g$ and the actual observed graphs. Then, update $\theta$ as well as MLPs $f^{mlp}_{\theta_f}$ and $f^{mlp}_{\theta_A}$ using the gradient of this error.
18. return set of generated graphs $\hat{G}$

set $TS = \{G_1, G_2, ..., G_N\}$, one can optimize $\theta$ for all $G \in TS$ and learn $\theta$ by maximizing the likelihood of observing $(F^i_\pi, \Lambda^i_\pi)$ pairs available in the dataset $TS$. Algorithm 3 shows our generative approach to train the parameters $\theta$ of our GRNN, i.e. $R_\theta(G)$, while generating $K_{gen}$ new graphs that follow the same patterns (with respect to the nodal features and edge weights) as the graphs in the dataset $TS$. Using our recurrent graph state modeling in (16), a GRU updates the graph state $S_i$ at each iteration $i$. Each state $S_i$ is mapped to $\theta_i^f$ and $\theta_i^A$ using mapping functions $f^{mlp}_\theta$ and $f^{mlp}_{\Lambda_\pi}$ respectively. The vectors $\theta_i^f$ and $\theta_i^A$ model the distribution of the nodal features and the weighted adjacency of the new node generated at iteration $i$, respectively. We implement both mappings $f^{mlp}_\theta$ and $f^{mlp}_{\Lambda_\pi}$ by a Multilayer Perceptron (MLP) network. The nodes are generated one by one by sampling from $\theta_i^f$ and $\theta_i^A$ once at each iteration $i$. When the END token is observed, the generated graph $\hat{G}_g$ is added to the set of generated items $\hat{G}$. The stochastic gradient descent method is employed to update GRU parameters $\theta = < W_z, W_r, W_C >$ as well as $f^{mlp}_\theta$'s and $f^{mlp}_{\Lambda_\pi}$'s weights and biases. The structure of our MLP networks are discussed in section VI-A.

E. Deep Distance Metric Learning for Graphs

As discussed in Algorithm 1, the DeepS3 algorithm computes the distance between any pair of items in the groundset; that is, for any $(x_i, x_j) \in D$ a distance metric $d(x_i, x_j; \theta)$ is learned where $\theta$ is the set of parameters of the representation function $R_\theta(.)$. In the case of Graph DeepS3, one needs to compute the distance metric between any pair of graphs $(G_i, G_j) \in D$; hence, we define a Siamese Architecture neural network based on our proposed GRNN to compute this metric. As shown in Fig. 4, our distance metric model contains two GRNNs with shared parameters $\theta$. $G_i$ is fed to a GRNN and its representation $R_\theta(G_i) = S_{|V_i|}$ is computed by Algorithm 2. Similarly, $R_\theta(G_j) = S_{|V_j|}$ is computed for $G_j$ and a distance metric is computed by:

$$d(G_i, G_j; \theta) = \|R_\theta(G_i) - R_\theta(G_j)\|_2 = \|S_{|V_i|} - S_{|V_j|}\|_2$$

We use (18) to compute the distance between any two graphs in the proposed Graph DeepS3 algorithm.

V. Power Network Synthesis

In this section, we introduce the problem of power network synthesis (PNS) as a Graph DeepS3 problem discussed in Section IV. Incorporating GRNN with our
presented DeepS3 algorithm, we propose a deep generative learning model to solve PNS and create a diverse set of realistic power networks that follow the structural and spatial characteristics of a realistic power network.

A. Problem Definition

Let us assume a real power network $P$ defined as a graph $G_P = (V_P, E_P)$ such that each node $v_i \in V_P$ represents a substation and each edge $e_{i,j} \in E_P$ represents a power transmission line between nodes (substations) $v_i$ and $v_j$. Each substation $v_i$ contains four real-valued features: the latitude $lat(v_i) \in \mathbb{R}$, longitude $lon(v_i) \in \mathbb{R}$, power supply $sup(v_i) \in \mathbb{R}$ that shows the amount of power generation at the corresponding node, and power demand $dem(v_i) \in \mathbb{R}$ which is the amount of power consumption at that node. Each edge $e_{i,j}$ is associated with a real-valued weight $W_{i,j} = X(e_{i,j}) \in \mathbb{R}$ showing the reactance ($X$) of the corresponding transmission line.

The data of actual power grid $G_P$ is confidential; therefore, the objective of the PNS problem is to generate synthetic networks that imitate the actual network for public use. That is, given $G_P$ we generate a diverse set of synthetic power grids that follow similar node and edge patterns (i.e. similar node features and edge weights), as well as similar topological characteristics (e.g. similar node degree and line length distributions) as $G_P$.

B. Power Grid Dataset

Since the data for actual power networks is confidential due to security and vulnerability issues, we consider the Columbia University Synthetic Power Grid (CUSPG) data \[32\] as the actual power grid $G_P$ we opt to synthesize. Fig. 5 depicts the CUSPG data with 14430 substations (nodes) and 18884 lines (edges) inside Western Canada, Western USA, and Northern Mexico. Each node is associated with its latitude, longitude, as well as power supply/demand; Moreover, each line is associated with a reactance value.

C. Deep Diverse Power Network Synthesis

Algorithm 4 is the pseudocode of our proposed deep diverse PNS (DeepDPNS) algorithm employing the presented Graph DeepS3 algorithm. First, We pretrain the GRNN on the CUSPG data using Breadth First Search (BFS). We randomly pick a node as the root node of BFS and traverse $G_P$ using BFS with randomly picked maximum depth $D_{max} \in \{1, 2, ..., 10\}$. The resulting subgraph is used as a training data to pretrain the GRNN by Algorithm 3.

Applying the modularity optimization in \[33\], we decompose $G_P$ into a set of $K = 72$ communities (subgraphs) $\mathcal{CM} = \{CM_k\}_{k=1}^K$ shown in Fig. 6 where each community contains dense interconnections among its own nodes while having sparse connections with other communities. We consider each community $CM_k$ as a ground-truth item inside the ground-truth set $\Psi$, that is, $\mathcal{CM} = \Psi$.

Applying Algorithm 3, we train the GRNN with parameters $\theta$ on $\mathcal{CM}$ to create subgraphs with the same nodal and edge distribution as $\{CM_k\}_{k=1}^K$. Now, having the set of generated subgraphs $D_k$ corresponding to each ground-truth subgraph $CM_k$, we define the set of all generated items $D = \bigcup_{k=1}^K D_k$. We employ the Graph DeepS3 algorithm for the subgraphs in $D \cup \Psi$ where $R_{\theta}(G)$ is the GRNN computed by Algorithm 2 and the graph distance metric is implemented as discussed in Section IV-E. Having a large ground set $D \cup \Psi$, our objective is to train GRNN using Graph DeepS3 such that our GDeepS3 algorithm chooses the ground-truth set $\Psi = \mathcal{CM}$. That is, we want GDeepS3 to be able to choose the actual power grid subgraphs $\mathcal{CM}$ from the generated ground set $D \cup \Psi$.

After training the GRNN by GDeepS3 and finding the optimal representation function $R^*_\theta(G)$ as well as the optimal regularization coefficient $\lambda^*$, one can choose the regularization coefficient $0 \leq \lambda \leq \lambda^*$ in the GDeepS3 in...
order to select a set of generated graphs $\Gamma_k$ corresponding to each ground set $D_k$. The subgraphs in $\Gamma_k$ are the most representative subgraphs for the whole items in $D_k$. We optimize 4 by running GDeepS3 on the ground set $D$ using $\lambda$ and select a representative subset $\Gamma = \bigcup_{k=1}^{K} \Gamma_k$ of generated subgraphs $D$, where each $\Gamma_k$ corresponds to one community $CM_k$.

Finally, in order to generate (synthesize) a realistic power network, we randomly pick one subgraph from each $\Gamma_k$ where $1 \leq k \leq K$ which results in a synthetic power network with $K = 72$ components. We further apply Algorithm 5 on the generated graph $G_{syn}$ to merge the subgraphs (components) into a single component and increase the robustness of the generated synthetic power network. As shown by 27, in a real power grid, the majority of the nodes have degrees more than 2. Moreover, the nodes in dense areas are more likely to have high degrees. Based on such observations, Algorithm 5 connects low-degree nodes $v_i \in V_{syn}$ in high density areas, to nodes with higher degrees $v_j \in V_{syn}$ until $G_{syn}$ is a single-component network.

VI. Simulation Results

In this section, we demonstrate the performance of our Deep Diverse Power Network Synthesis algorithm in creating realistic power networks. First, we explain the values assigned to various hyperparameters of our algorithm. Then, the performance metrics of the PNS problem as well as the compared benchmarks are introduced. Finally, the qualitative and quantitative results and comparisons are presented.

A. Experimental Settings

Here, we determine the hyperparameters for the GRNN model as well as our presented DeepDPNS algorithm.

Algorithm 4: Deep Diverse Power Network Synthesis

Input: Actual Power Network $G_P = (V_P, E_P)$
Output: A synthesized power network $G_{syn} = (V_{syn}, E_{syn})$

1 GRNN Pretraining: Pretrain GRNN on $G_P$ using Algorithm 3 and BFS with maximum depth $D_{max} \in \{1, 2, ..., 10\}$ on randomly chosen root nodes $v_i \in V_P$.

2 Graph Decomposition: Decompose $G_P$ into $K = 72$ communities (subgraphs) $CM = \{CM_k\}_{k=1}^{K}$. Define the ground-truth set $\Psi = CM$.

3 Generating subgraphs: Train GRNN to generate subgraphs in $CM$ using Algorithm 3. Create a set of subgraphs $D_k$ corresponding to each ground-truth subgraph $CM_k$ in $CM$. Define the set of all generated subgraphs $D = \bigcup_{k=1}^{K} D_k$.

4 Learning GRNN to pick the best subgraphs: Run Graph DeepS3 with ground-set $D \cup \Psi$ and representation function $R_\theta(G)$ defined as GRNN and computed by Algorithm 2 for each subgraph $G$. The ground-truth set is $\Psi$. The optimal GRNN, i.e $R_\theta^*(G)$, and optimal regularization coefficient $\lambda^*$ are computed by the Graph DeepS3 in this step.

5 Selecting the best generated subgraphs: Choose $0 < \lambda \leq \lambda^*$ and run Graph DeepS3 on the groundset $D$ (i.e. our generated subgraphs) using the optimal representation $R_\theta^*(G)$ obtained by Graph DeepS3 in previous step. A set of representative subgraphs $\Gamma_k$ is obtained for each set of generated subgraphs $D_k$.

6 Creating the synthetic network: Randomly pick a subgraph from each representative set $\Gamma_k$ to create a synthetic graph with $K$ components denoted by $G_{syn}$.

7 Connecting the components: Merge the components of $G_{syn}$ by Algorithm 5 to make the resulting graph connected and robust.

8 return Generated (synthesized) graph $G_{syn}$

Algorithm 5: Connecting the components

Input: Power network $G_{syn} = (V_{syn}, E_{syn})$ with $K$ components
Output: Single-component power network $G_{syn}$ contains more than one components do

1 while $G_{syn}$ contains more than one components do

2 Pick node $v_i \in V_{syn}$ from all nodes with degree 2 or less with probability $\rho(v_i)^{-1}$ where $\rho(v_i)$ is the average Euclidean distance of $v_i$ from its $k = 10$ nearest neighbors.

3 Connect $v_i$ to a node $v_j \in V_{syn}$ with probability $\deg(v_j)(\text{dist}(v_i, v_j))^1$ where $\deg(v_j)$ is the degree of node $v_j$, and $\text{dist}(v_i, v_j)$ denotes the Euclidean distance between the two nodes $v_i$ and $v_j$.

4 return $G_{syn}$
DeepDPNS applies GDeepS3 (Algorithms 2 and 3) based on DeepS3 (algorithm 1); hence, some hyperparameters are corresponding to these algorithms.

1) GRNN settings: As each node in the power network contains four features, the feature vector for the i-th node, $F^i$ is four dimensional; thus, the input $I$ is a five dimensional vector. Moreover, the graph state variable $S_i$ and temporal features $z$ and $r$ have 128 dimensions. The mapping functions $f^m_{mlp}$ and $f^a_{mlp}$ are both defined by two-layered MLP neural networks with 40 and 30 sigmoid units in their first and second hidden layers, respectively. The learning rate of our generative learning algorithm (Algorithm 3) is $10^{-5}$.

2) DeepDPNS settings: For the "GRNN pretraining" step, the GRNN is pretrained on $10^5$ subgraphs of $G_P$ determined by BFS as discussed in Algorithm 4. In the "graph decomposition" step, the number of communities is set to $K = 72$ as stated in Algorithm 4. In the "generating subgraphs" step, the cardinality of the set of created subgraphs $D_k$ is set to 100. In the fourth step i.e. "Learning GRNN to pick the best subgraphs", the Graph DeepS3 algorithm uses DeepS3 (Algorithm 1) with margin $m = 1$, number of positive examples $K_P = 5 \times 10^5$ and number of negative examples $K_N = K_P$. We consider similar contribution for our presented loss functions in (11) by setting $p_{med} = p_{NE_i} = p_{NE} = 1$. Moreover, the DeepS3 learning rate $\eta$ in (11) is set to $10^{-3}$. In the fifth step, "Selecting the best generated subgraphs", we choose $\lambda = 0.8 \times \lambda^*$. 

B. Topological Performance Metrics

In order to comprehensively compare the generated power networks $G_{syn}$ with the actual power grid $G_P$, we compute and compare several structural metrics including average degree $\bar{deg}$, degree distribution, average path length $\omega$, and edge length distribution. Moreover, we compute the following topological metrics:

1) Diameter ($D$): The diameter of a graph $G = (V,E)$ is the greatest eccentricity $\epsilon(v)$ of any $v \in V$. That is, $D$ is the greatest distance between any pair of vertices computed by $D = \max_{v \in V} \epsilon(v)$.

2) Density ($\rho$): For an undirected graph $G = (V,E)$, the density is computed by $\rho = \frac{|E|}{|V|^2}$.

3) Modularity ($Q$): The network modularity for any network modeled by a weighted graph $G = (V,E)$ with adjacency matrix $A$ is computed by:

$$Q = \frac{1}{2m} \sum_{i,j} \left[ A_{ij} - \frac{k_i k_j}{2m} \right] \zeta(CM(i),CM(j))$$ (19)

where $A_{ij}$ is the weight between of nodes $v_i$ and $v_j$, $k_i = \sum_j A_{ij}$ is the sum of weights connected to node $v_i$, $CM(i)$ is the community assigned to node $v_i$, and the $\zeta(CM(i),CM(j))$ function is 1 if $CM(i) = CM(j)$ and zero otherwise. $m$ is computed by $m = \frac{1}{2} \sum_{i,j} A_{ij}$. In the case of a power network, the nodes are substations and the edge weights are the reactance values of the transmission lines between pairs of substations.

4) Average Clustering Coefficient (ACC) ($C$): For each node $v_i$ in $G = (V,E)$ with degree $\bar{deg}(v_i)$, the maximum number of edges from $v_i$ to its neighbors $N(v_i) = \bar{deg}(v_i) (\bar{deg}(v_i) - 1) / 2$. Let us compute the fraction of such potential edges that actually exists in the graph by $C(v_i) = \frac{|{(r,s)\mid r,s \in N(v_i),\{r,s\} \in E}|}{\bar{deg}(v_i) (\bar{deg}(v_i) - 1) / 2}$. The average clustering coefficient is then computed by $C = \frac{1}{|V|} \sum_{v_i \in V} C(v_i)$.

C. Power Grid Performance Metrics

Following [21], we employ the DC power flow model widely studied as an approximation of the AC power flow model. We consider similar notation as Section V-A for the power network $G_P = (V_P,E_P)$ with each node $v_i \in V_P$ having a net power supply/demand value $P(v_i) = sup(v_i) - dem(v_i)$. Also, each edge (transmission line) $e_{i,j} \in E$ between $v_i$ and $v_j$ is defined by its reactance $X(e_{i,j}) > 0$.

Given the total supply/demand vector $P = [P(v_1), P(v_2),...,P(v_{|V|})] \in \mathbb{R}^{|V| \times 1}$, and the line reactance values, the power flow is defined as a solution ($f, \theta$) of the following problem:

$$\sum_{i \text{ s.t. } v_i \in N(v_j)} f_{ij} = P(v_i) \forall v_j \in V_P$$ (20)

$$f_{ij} = (\theta_i - \theta_j)/X(e_{i,j}) \forall e_{i,j} \in E_P$$ (21)

$N(v_j)$ is the set of neighbors of $v_j$, $f_{ij}$ is the power flow from $v_i$ to $v_j$, and $\theta$ denotes the phase angle corresponding to $v_i$. Here, [20] takes care of the classic flow conservations while [21] defines the flow as a function of phase angle and transmission line reactance. Following the cascade model for line overloads in power systems, we have the following definition for line failure: a line $e_{i,j}$ fails if its flow magnitude $|f_{ij}|$ exceeds the line capacity $c_{p_{i,j}}$. The line capacity is computed by $c_{p_{i,j}} = (\alpha+1)\max(f_{ij},\bar{f})$ with the median of the initial magnitude of line flows denoted by $\bar{f}$. The line safety factor is denoted by $\alpha$. Following [21], the minimum capacity of the lines in the generated network is set to the median of initial magnitude of line flows in the actual network $G_P$.

When a line fails, it is removed from $G_P$; thus, the network structure is changed. In fact, the network might be divided into several components. The connected components operate autonomously; that is, each component with nonzero supply/demand balances its supply and demand values by scaling down the supply values (when supply exceeds demand inside the component) or scaling down the demand values (when demand exceeds supply).

When the supply/demand values are balanced in all components, the power flow problem (20 and 21) is solved again to update the flows in the system. Considering deterministic outage rules, a new set of lines are failed, and the cascading failure continues by removing failed lines.
The cascade terminates when no line is overloaded in the whole power system.

To compare the performance of $G_{\text{Syn}}$ with the actual power grid $G_P$, we compute $\text{Yield}$, as the ratio between the demand supplied at the end of cascading failure and the actual demand before the cascade. Also, we compute the number of failed lines and the number of connected components after the cascading failure in order to measure the severity of the event.

### D. Benchmarks

As discussed in Section I, all current graph generative models lack scalability and flexibility. In fact, the time and memory complexity of such models do not let such approaches be applicable for creating large-scale graphs. In very recent literature, novel deep learning models including Graph Variational Auto-encoder (GraphVAE) \cite{15}, Deep Generative Model of Graphs (DeepGMG) \cite{12}, and Graph Recurrent Neural Network (GraphRNN) \cite{16} have been proposed; however, such approaches could be only applied to create graphs with at most 2025 nodes.

Moreover, such models are only capable of generating unweighted graphs with binary edges showing whether an edge exists or not.

In this study, we generate power networks as large-scale graphs with around $14 \times 10^3$ nodes and $18 \times 10^3$ edges. Also, the generated graphs need to be weighted as each edge (transmission line) is associated with a real-valued feature (i.e. reactance value). Therefore, previous models cannot be applied to solve the presented problem. However, in order to have a fair comparison with recent benchmarks, we define three baselines as three versions of our DeepDPNS algorithm: DeepDPNS(GraphVAE), DeepDPNS(DeepGMG), and DeepDPNS(GraphRNN) that respectively employ GraphVAE, DeepGMG, and GraphRNN instead of our GRNN model for modeling $R_0(G)$. Notice that, since these models can merely generate unweighted edges, they are not capable of generating reactance values for the transmission lines (edges). Hence, following \cite{27}, we assume that the reactance of each line is equal to its length.

### E. Results and Comparisons

First, we discuss the topological properties of the generated subgraphs. Then, we explain the performance of the presented model for the PNS problem in terms of topological properties as well as power system metrics.

1) Generated Subgraphs: Fig. 7(a) shows the 31st community denoted by $CM_{31}$, with the net power generation (amount of supply after subtracting the demand) at each node. Moreover, Fig. 7(b), (c), and (d) respectively show the corresponding generated subgraphs $Gen_{31}(1)$, $Gen_{31}(2)$, and $Gen_{31}(3)$ stored in $\Gamma_{31}$. These subgraphs are selected by the Graph DeepS3 algorithm as they best represent $CM_{31}$ among all members of $D_k$. As shown in these plots, the generated subgraphs follow the same geometric structure as the actual power grid. Moreover, it is shown that the nodes of our generated subgraphs accurately follow the patterns of net power generation in the actual networks. This indicates that DeepDPNS can effectively learn the probability distributions of power demand and supply of a real-world power grid. Fig. 8 visualizes the generated graphs corresponding to communities $CM_{31}$ and $CM_{64}$ inside the embedding space. Using the principal component analysis (PCA), we plot the first three most significant principal variables $PC_1$, $PC_2$, and $PC_3$ of $R_0$ to visualize the generated graphs. As shown here, the selected graphs are the most representative data points in the embedding space, since the other data points can be constructed as a linear combination of these points with minimum error (as seen in (3)).

Table I shows the topological properties of $Gen_{31}(1)$, $Gen_{31}(2)$, and $Gen_{31}(3)$. As shown in this table, the generated graphs can have a wide variety of number of nodes and edges; however, they all share similar graph properties. For instance, the generated graphs lead to a $deg_{avg} \in [2.331, 2.418]$ while the actual network $CM_{31}$ has a value of 2.299. Moreover, graph modularity is $Q \in [0.784, 0.805]$ for the generated networks while the actual power network leads to $Q = 0.790$. We also see that the ACC measurement of generated graphs is $C \in [0.081, 0.084]$ while the actual network has an ACC of $C = 0.086$. Also, it is shown that the diameter $D$ of the generated graphs change in the range of [19, 21] while $CM_{31}$ has a diameter $D = 19$. Table I also shows similar topological properties in terms of $deg_{avg}$, $\omega$, $D$, $\rho$, $Q$, and $C$ for the actual network $CM_{64}$ and its corresponding synthetic graphs. The similar topological characteristics justify that our created synthetic networks using the DeepDPNS algorithm can successfully imitate the structural properties of real-world power networks.

2) Synthetic Power Grid $G_{\text{Syn}}$: In order to provide comprehensive results, we generate $3 \times 10^2$ synthetic power grids $G_{\text{Syn}}$ by repeating steps 6 and 7 of our DeepPNS algorithm (algorithm 4). We refer to two of the generated synthetic power networks as $G_{\text{Syn}}(1)$ and $G_{\text{Syn}}(2)$. As shown by Fig. 9, the synthetic networks visually resemble the original power grid $G_P$ in Fig. 6. Similar to the original network $G_P$, both $G_{\text{Syn}}(1)$ and $G_{\text{Syn}}(2)$ have sparse connections on the upper regions (near Canada) while showing dense communities on the lower left side (near California). It is shown that the distribution of the transmission lines are preserved in the synthesized networks. Also, the distribution of the node locations follows the original network,

### TABLE I: Topological characteristics of the generated subgraphs

| Graph   | #Nodes | #Edges | $deg_{avg}$ | $\omega$ | $D_1$ | $\rho$ | $Q$ | $C$   |
|---------|--------|--------|-------------|----------|-------|-------|-----|------|
| $CM_{31}$ | 107    | 125    | 2.299       | 9.369    | 19    | 0.019 | 0.805| 0.089|
| $CM_{64}$ | 139    | 172    | 2.475       | 9.816    | 21    | 0.022 | 0.790| 0.081|
| $Gen_{31}(1)$ | 121    | 141    | 2.331       | 9.009    | 19    | 0.019 | 0.805| 0.089|
| $Gen_{31}(2)$ | 91     | 106    | 2.330       | 8.381    | 20    | 0.026 | 0.784| 0.084|
| $Gen_{31}(3)$ | 110    | 133    | 2.418       | 8.148    | 21    | 0.022 | 0.790| 0.081|
| $Gen_{64}(1)$ | 160    | 201    | 2.539       | 10.257   | 26    | 0.016 | 0.833| 0.063|
| $Gen_{64}(2)$ | 118    | 140    | 2.375       | 10.071   | 27    | 0.020 | 0.817| 0.063|
| $Gen_{64}(3)$ | 141    | 179    | 2.599       | 10.257   | 25    | 0.018 | 0.828| 0.062|
which shows that DeepDPNS successfully imitates the positions of the substations (nodes) in a real power grid.

Fig. 10 compares the degree distributions of the nodes and length distributions of the lines in $G_P$, $G_{Syn}(1)$ and $G_{Syn}(2)$. As shown in this figure, the nodes and edges of our synthetic networks obtain similar topological distributions as the original power grid. This is due to the fact that the GRNN in DeepDPNS captures the distribution of nodes and edges in the observed graph rather than the actual measurements. Table III summarizes the topological properties of $G_P$, $G_{Syn}(1)$, $G_{Syn}(2)$ as well as the mean and standard deviation (STDV) of all generated graphs. As shown in this table, our DeepDPNS algorithm is able to generate a large variety of networks with a wide range of number of nodes/edges that share similar structural characteristics.

3) Power Systems Analysis: Here, we compare the physical behaviors of $G_P$ with the generated power networks. The DC power flow equations discussed in Section VI-C are used to investigate the robustness of the created networks against the cascading failure. Fig. 11(a) shows the power demand values at each substation in the actual power grid $G_P$ while Fig. 11(b) and (c) are the similar results for $G_{Syn}(1)$ and $G_{Syn}(2)$, respectively. As shown in these figures, the generated networks show same demand distribution as the original power network. The majority of substations with demand values in the range $[0.25, 1.75](10^2MW)$ are generally in the upper regions (sparse regions) of the networks. Also, it is shown that the substations with high demand values (higher than $1.75(10^2MW)$) are uniformly distributed over the substations in the networks.

The power flow statistics of the actual network and the synthetic power grids are reported in Table III. The average flow of $G_P$ is $168.93MW$ with a STDV of $320.93MW$, while the average flow in the generated networks has a mean value of $171.18 \pm 11.34MW$ with STDV equal to $307.18 \pm 15.59$. The median and maximum values of the powerflows also show similar results for $G_P$ and the generated networks. Also, the experiments show that the backup lines (i.e. lines that do not initially carry any significant flow) lead to similar flow values in $G_P$ and our synthetic networks.

In order to have a comprehensive comparison of DeepDPNS with the baselines, we conduct two experiments using cascading failures to show the robustness of the generated networks. In the first experiment, we compute the severity of cascades initiated by all possible double line failures chosen from the lines that carry the top 25 largest power flows as a function of the lines’ safety factor ($\alpha$). Fig. 12 (a) depicts the the box and whisker diagram of the Yield metric corresponding to all our generated networks using the DeepDPNS algorithm and the baselines. The safety factor is considered to be $\alpha \in \{0.2, 0.4, 0.6, 0.8, 1\}$. Moreover, Fig. 12 (b) shows the number of failed lines (NFL) at the end of the cascades. In the second experiment, we measure the average severity of the cascades initiated by failures in $10^4$ uniformly distributed regions of
TABLE II: Topological characteristics of the generated networks and the original power grid

| Graph       | #Nodes  | #Edges  | \(\text{deg}_{	ext{avg}}\) | w | \(D\) | \(\rho \times 10^{-4}\) | Q | C |
|-------------|---------|---------|-----------------------------|---|------|----------------------|---|---|
| Actual GP   | 14430   | 18554   | 2.572                       | 37 | 1.782 | 0.953                | 0.072 |   |
| GSyn(1)     | 11500   | 13143   | 2.480                       | 37 | 1.987 | 0.961                | 0.072 |   |
| GSyn(2)     | 14569   | 18552   | 2.539                       | 37 | 1.748 | 0.939                | 0.071 |   |
| Mean        | 14283   | 17954   | 2.529                       | 37 | 1.887 | 0.941                | 0.071 |   |

TABLE III: Flow statistics of the generated networks and the original power grid

| Power Networks | Average (MW) | Median (MW) | STDV (MW) | Max (MW) | Backup Lines (MW) |
|----------------|-------------|-------------|-----------|----------|-------------------|
| G_P            | 168.93      | 33.32       | 320.93    | 4,777.07 | 3,419             |
| GSyn(1)        | 174.97      | 29.81       | 298.71    | 4,392.91 | 3,442             |
| GSyn(2)        | 165.12      | 31.64       | 326.08    | 4,692.37 | 3,482             |
| Mean           | 171.18      | 30.42       | 307.18    | 4,590.25 | 3,369             |

radius 20km. Fig. 12(c) shows the box and whisker plot of the Yield metric corresponding to all generated networks using DeepDPNS and other benchmarks. Furthermore, Fig. 12(d) depicts the NFL when cascades are terminated. As shown in these plots, our DeepDPNS algorithm can accurately imitate the behavior of the real power system when a failure happens. The distribution of the Yield and NFL measurements of DeepDPNS accurately follow the actual measurements in the system. Moreover, it is shown that DeepDPNS provides more accurate results compared to the baselines since DeepDPNS employs the proposed GRNN, which can learn line features (reactance values) while other methodologies cannot capture such features from their training data. Hence, the experiments show that learning reactance values of the lines improves our generated networks to imitate the reaction of the actual power grid when facing failures.

To study the spatial behavior of the generated power systems when the failure occurs, Fig. 13 plots the Yield of cascades initiated by failures in \(10^4\) regions defined in our second experiment using \(\alpha = 0.6\). Each point represents the center of a region with a radius of 20km where the failure begins. The color of each point shows the Yield of cascade initiated by the failure in the corresponding region centered at that point. The Yield measurements obtained by our synthetic networks \(G_{\text{syn}}(1)\) and \(G_{\text{syn}}(2)\) have the same spatial structures as the original power grid \(G_P\). Similar to \(G_P\), the synthetic networks obtain a Yield equal to or greater than 0.6 around the upper regions (sparse regions) as well as on the right border. The Yield metric is decreased to below 0.4 in most regions in the center and the left borders of the networks. Moreover, \(G_P\) contains a few regions colored in yellow with Yield values in the range [0.4, 0.6]. Similar Yield results are obtained by our networks, \(G_{\text{syn}}(1)\) and \(G_{\text{syn}}(2)\), in the same regions close to the center of the graphs as well as near left borders. In addition to Yield measurements, we show the average number of connected components (NCC) when the cascade is terminated for various \(\alpha\) values. Fig. 14(a) and (b) depict the NCC corresponding to the first and second experiments, respectively. As shown in these plots, the NCCs of our synthetic networks decline with the same slope as the original power grid. Similar to the Yield and NFL measurements, here we see that DeepDPNS can better follow the behavior of the actual network compared to the baselines as the proposed GRNN is able to learn the distribution of edge features (e.g. line reactance values) while other methods are not capable of learning such features as they can merely handle binary edges.

VII. CONCLUSIONS

Modeling and generating a large set of realistic graphs that imitate the topological behaviors and physical characteristics of real-world graphs/networks is a crucial problem in various fields of science including electrical engineering, social science, and biology; however, this problem has never been addressed due to lack of generalization and lack of knowledge obtained by the models about the graph distributions. In this paper, we introduce the novel problem of creating diverse synthetic graphs. First, we propose the deep supervised subset selection (DeepS3) algorithm that applies deep learning to select a diverse
imitate the Columbia University Synthetic Power Grid (CUSPG). Simulation results show that DeepDPNS resembles CUSPG in terms of various topological properties including average degree, degree distribution, average path length, edge length distribution, diameter, density, modularity, and average clustering coefficient. The power flow analysis shows similar statistics for the power flow in the synthetic networks compared to the original grid. Moreover, the conducted power flow analysis shows the robustness of the generated networks against different failures in the power system. The experimental results indicate that DeepDPNS accurately follow the same performance as the original grid in terms of Yield, the number of failed lines, and the number of connected components after the cascading failure.

**References**

[1] J. Li, D. Cai, and X. He, “Learning graph-level representation for drug discovery,” arXiv preprint arXiv:1709.09741, 2017.

[2] B. Liu, T. Zhang, F. X. Han, D. Niu, K. Lai, and Y. Xu, “Matching natural language sentences with hierarchical sentence factorization,” arXiv preprint arXiv:1803.00179, 2018.

[3] S. Banerjee and S. Ghosal, “Bayesian structure learning in graphical models,” Journal of Multivariate Analysis, vol. 136, pp. 147–162, 2015.

[4] P. Erdos and A. Rényi, “On the evolution of random graphs,” Publ. Math. Inst. Hung. Acad. Sci, vol. 5, no. 1, pp. 17–60, 1960.

[5] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” science, vol. 286, no. 5439, pp. 509–512, 1999.

[6] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, “Kronecker graphs: An approach to modeling networks,” Journal of Machine Learning Research, vol. 11, no. Feb, pp. 985–1042, 2010.

[7] R. Albert and A.-L. Barabási, “Statistical mechanics of complex networks,” Reviews of modern physics, vol. 74, no. 1, p. 47, 2002.

[8] E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing, “Mixed membership stochastic blockmodels,” Journal of Machine Learning Research, vol. 9, no. Sep, pp. 1981–2014, 2008.

[9] G. Robins, P. Pattison, Y. Kalish, and D. Lusher, “An introduction to exponential random graph (p*) models for social networks,” Social networks, vol. 29, no. 2, pp. 173–191, 2007.

[10] H. Ehrig, G. Rozenberg, and H.-J. rg Krewosw, Handbook of graph grammars and computing by graph transformation, world Scientific, 1999, vol. 3.

[11] S. Aguiñaga, R. Palacios, D. Chiang, and T. Weninger, “Growing graphs from hyperedge replacement graph grammars,” in Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. ACM, 2016, pp. 469–478.

[12] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, “Learning deep generative models of graphs,” arXiv preprint arXiv:1803.03924, 2018.

[13] T. N. Kipf and M. Welling, “Variational graph auto-encoders,” arXiv preprint arXiv:1611.07308, 2016.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. X., D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in Advances in neural information processing systems, 2014, pp. 2672–2680.

[15] M. Simonovsky and N. Komodakis, “Graphvae: Towards generation of small graphs using variational autoencoders,” arXiv preprint arXiv:1802.03480, 2018.

[16] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, “Graphrnn: Generating realistic graphs with deep autoregressive models,” in International Conference on Machine Learning, 2018, pp. 5694–5703.

[17] D. Bienstock, M. Chertkov, and S. Harnett, “Chance-constrained optimal power flow: Risk-aware network control under uncertainty,” Siam Review, vol. 56, no. 3, pp. 461–495, 2014.

[18] Y. Zhao, A. Goldsmith, and H. V. Poor, “On pmu location selection for line outage detection in wide-area transmission networks,” arXiv preprint arXiv:1207.6617, 2012.
Fig. 11: Power demand of the actual power grid $G_P$ and the generated networks $G_{Syn}(1)$ and $G_{Syn}(2)$.

Fig. 12: Severity of cascades initiated by double line failures in the first and second experiments.

[19] G. Latorre, R. D. Cruz, J. M. Areiza, and A. Villegas, “Classification of publications and models on transmission expansion planning,” IEEE Transactions on Power Systems, vol. 18, no. 2, pp. 938–946, 2003.

[20] W.-L. Chin, C.-H. Lee, and T. Jiang, “Blind false data attacks against ac state estimation based on geometric approach in smart grid communications,” IEEE Transactions on Smart Grid, vol. 9, no. 6, pp. 6298–6306, 2018.

[21] M. M. Danziger, L. M. Shekhtman, Y. Berezin, and S. Havlin, “Two distinct transitions in spatially embedded multiplex networks,” arXiv preprint arXiv:1505.01688, 2015.

[22] M. A. S. Monfared, M. Jalili, and Z. Alipour, “Topology and vulnerability of the iranian power grid,” Physica A: Statistical Mechanics and its Applications, vol. 406, pp. 24–33, 2014.

[23] P. Crucitti, V. Latora, and M. Marchiori, “A topological analysis of the italian electric power grid,” Physica A: Statistical Mechanics and its Applications, vol. 338, no. 1-2, pp. 92–97, 2004.

[24] D. P. Chassin and C. Posse, “Evaluating north american electric grid reliability using the barabási–albert network model,” Physica A: Statistical Mechanics and its Applications, vol. 355, no. 2-4, pp. 667–677, 2005.

[25] K. M. Gegner, A. B. Birchfield, T. Xu, K. S. Shetye, and T. J. Overbye, “A methodology for the creation of geographically realistic synthetic power flow models,” in Power and Energy Conference at Illinois (PECI), 2016 IEEE. IEEE, 2016, pp. 1–6.

[26] A. B. Birchfield, T. Xu, K. M. Gegner, K. S. Shetye, and T. J. Overbye, “Grid structural characteristics as validation criteria for synthetic networks,” IEEE Transactions on Power Systems, vol. 32, no. 4, pp. 3258–3265, 2017.

[27] S. Soltan, A. Loh, and G. Zussman, “A learning-based method for generating synthetic power grids,” IEEE Systems Journal, no. 99, pp. 1–10, 2018.

[28] E. Elhamifar, G. Sapiro, and S. S. Sastry, “Dissimilarity-based sparse subset selection,” IEEE transactions on pattern analysis and machine intelligence, vol. 38, no. 11, pp. 2182–2197, 2016.

[29] E. Elhamifar and M. C. D. P. Kaluza, “Subset selection and summarization in sequential data,” in Advances in Neural Information Processing Systems, 2017, pp. 1035–1045.

[30] S. Javed, A. Mahmood, S. Al-Maadeed, T. Bouwmans, and
Fig. 13: Yield of cascades initiated by failures in $10^4$ regions uniformly distributed for $\alpha = 0.6$.

S. K. Jung, “Moving object detection in complex scene using spatiotemporal structured-sparse rpca,” *IEEE Transactions on Image Processing*, vol. 28, no. 2, pp. 1007–1022, 2019.

[31] X. Fang, S. Teng, Z. Lai, Z. He, S. Xie, and W. K. Wong, “Robust latent subspace learning for image classification,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2502–2515, 2018.

[32] S. Soltan, A. Loh, and G. Zussman, “Columbia university synthetic power grid with geographical coordinates,” 1 2018.

[33] X. Lu, K. Kuzmin, M. Chen, and B. K. Szymanski, “Adaptive modularity maximization via edge weighting scheme,” *Information Sciences*, vol. 424, pp. 55–68, 2018.