



FlowGEN: A Generative Model for Flow Graphs

Furkan Kocayusufoglu
University of California, Santa
Barbara, CA, USA
furkank@cs.ucsb.edu

Arlei Silva
Rice University,
Houston, TX, USA
arlei@rice.edu

Ambuj K. Singh
University of California, Santa
Barbara, CA, USA
ambuj@cs.ucsb.edu

挑战

ABSTRACT

Flow graphs capture the directed flow of a quantity of interest (e.g., water, power, vehicles) being transported through an underlying network. Modeling and generating realistic flow graphs is key in many applications in infrastructure design, transportation, and biomedical and social sciences. However, they pose a great challenge to existing generative models due to a complex dynamics that is often governed by domain-specific physical laws or patterns. We introduce FlowGEN, an implicit generative model for flow graphs, that learns how to jointly generate graph topologies and flows with diverse dynamics directly from data using a novel (flow) graph neural network. Experiments show that our approach is able to effectively reproduce relevant local and global properties of flow graphs, including flow conservation, cyclic trends, and congestion around hotspots.

CCS CONCEPTS

• Computing methodologies → Spatial and physical reasoning; Learning latent representations.

KEYWORDS

Representation learning; Graph generative models; Flow networks

ACM Reference Format:

Furkan Kocayusufoglu, Arlei Silva, and Ambuj K. Singh. 2022. FlowGEN: A Generative Model for Flow Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22)*, August 14–18, 2022, Washington, DC, USA. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3534678.3539406>

1 INTRODUCTION

Generative models for graph data have a long-standing history in network science [1, 15]. Major motivations for such models include unavailability of large datasets, due to privacy concerns, and the cost of data collection. These models are useful for data-driven discovery, anomaly detection, and large-scale simulations in the natural sciences. While there is an extensive literature on mathematical models for graph data, more recent approaches based on neural generative models have attracted great interest [5, 11, 20, 32, 34, 35, 37, 41, 48].

The main advantage of the recent generative approaches is the potential to learn how to generate a broad class of graphs from a limited number of samples. However, they are primarily focused on

reproducing undirected graph topologies observed in simple graph structures (e.g., grid, community). In the many real-world applications, the ability to mimic (binary) connectivity patterns among nodes alone is not sufficient to capture their key characteristics. To that end, there have been recent studies in generative models for other families of graphs that go beyond generic structures, such as molecular graphs [26, 39], directed acyclic graphs (DAGs) [49], city road layouts [10], and program (source code) graphs [7].

We focus on the generative modeling of a complex family of graphs called *flow graphs* (FGs). Besides structure, FGs capture flows of a quantity of interest (e.g., water, power, people) being transported through the graph. As these flows often possess higher-order graph-level dynamics driven by sources/destinations [8], hotspots [3], and domain-specific physics [40], generating FGs poses greater challenges than those only focused on generating a graph topology.

As an example, consider a transportation network where nodes correspond to locations, edges are routes between them, and flows indicate the number of people (or vehicles) transported from one location to another. Besides topological properties, such as the existence of central nodes and clusters, this FG also reveals specific flow patterns. For instance, hotspots (e.g. recreational areas) might become prominent sources and destinations depending on the time of the day. Moreover, the prevalence of cyclic flows can unveil diurnal variations in travel patterns. The ability to generate realistic mobility flows is key for the understanding of urban dynamics [3] and the spread of epidemics [9]. Similar examples can be constructed for the analysis of load distribution in power networks [14, 23] and flux balance in biological systems [38].

It is important to emphasize some of the key differences between flow graphs and general weighted graphs. In typical weighted graphs, weights represent the strength of a relationship between two nodes (e.g. homophily in social networks). Conversely, FG flows depend on higher order structures that induce patterns such as source-destinations, cycles, congestion [40], and the degree to which flows are conserved at nodes (does inflow equal outflow?).

We first investigate how existing graph generative models can be combined with a learnable flow generating function to produce FGs based on observed samples. While this two-step approach is flexible and can be integrated with any existing model, it is unable to learn the joint relationship between graph structure and flows in an end-to-end fashion. To address this limitation, we introduce FlowGEN, an implicit generative model for FGs based on the Generative Adversarial Networks (GAN) framework [18]. FlowGEN learns to generate both the (directed) graph topology and edge flows. The main ingredient of FlowGEN's architecture is a discriminator with the following components: (1) a permutation invariant flow-pooling layer, (2) bi-directional neural message-passing layers, and (3) an attention-based readout layer. This enables FlowGEN to



This work is licensed under a Creative Commons Attribution International 4.0 License.

KDD '22, August 14–18, 2022, Washington, DC, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9385-0/22/08.
<https://doi.org/10.1145/3534678.3539406>

learn hidden node representations (or states) capturing a complex coupling between the graph topology and edge flows.

We assess the performance of flow graph generative models both qualitatively and quantitatively, using novel evaluation metrics designed for FGs. Our results show that FlowGEN can effectively reproduce flow properties of a diverse set of real-world and simulated networks, including transportation, power transmission, and water networks, outperforming the alternative approaches.

2 RELATED WORK

The generation of realistic graphs that mirror the characteristics of real-world graphs is a long-standing research problem. The earliest graph generative models [1, 15, 46] focus on reproducing certain statistical properties (e.g., degree distribution, graph density) found in real-world graphs using a stochastic generation process. Although simple to use, these models are limited in that the selected properties may not sufficiently represent many other characteristics of real-world graphs.

To address the above problem and inspired by recent advancements in deep learning [31, 45], several neural graph generative models have been proposed [5, 11, 19, 20, 30, 32, 34, 35, 37, 41, 47, 48]. Compared to traditional models, neural models are able to learn rather complex structural properties from data (e.g., spectral coefficients and orbit counts), and to reproduce graphs with fundamentally different topologies (e.g., grids and egonets). Amongst the pioneer models, GraphVAE [41] utilizes variational autoencoders (VAE) [28] to generate a probabilistic fully-connected graph and then applies an expensive graph matching algorithm to calculate the reconstruction loss. As an alternative, NetGAN [5] converts graphs into random walks and learns an implicit probabilistic model for generating walks using adversarial training [18], thus eliminating the need for expensive graph matching. More recently, EDP-GNN [37] leverages a score-based generative modeling framework [42] to achieve permutation invariant graph generation.

Another recent paradigm is to model graphs via an auto-regressive process—as a sequence of additions of new nodes and edges, conditioned on the current sub-graph structures [11, 32, 34, 48]. GraphRNN [48] utilizes RNNs for the sequential modeling of nodes, while GRAN [34] considers the sequential modeling of blocks of nodes, employing GNNs with attention [45] for conditioning. BiGG [11] further exploits the sparsity of real-world graphs by combining a recursive edge generation scheme with auto-regressive conditioning, resulting in a technique that sidesteps the explicit generation of each entry in an adjacency matrix. Although these approaches are able to generate larger graphs relative to their counterparts, they rely on pre-defined node orderings of graphs (BFS/DFS) since computing the full likelihood is intractable due to all possible node orderings. However, this limits their capacity to model long-range (global) dependencies, which is a key element for generating realistic flow graphs, as we show in Section 4. Moreover, determining a suitable node ordering is subject to the task and data of interest.

Flow graphs are a richer model than simple graphs because they capture the dynamics of a system with node states [6]. The coupling between edge flows and node states is governed by domain-specific physical models, such as the Kirchhoff’s laws for power and the LWR model for traffic [16]. Recent work has shown how learning such models from data can improve the prediction of missing edge

flows [40]. Here, we focus on learning how to generate flow graphs by implicitly capturing the coupling between topology and flows. Measuring flows (of mobility, water etc.) on graphs requires much effort and resources. Generative models are especially useful in these settings in order to replicate the behavior of real data for analysis, simulation, and algorithm development.

3 PROPOSED MODELS

We start by providing some notation, background, and a formal definition of our problem.

A **flow matrix** $X \in \mathbb{R}_+^{d \times d}$ is a non-negative square matrix where the entry $x_{u,v}$ denotes a *directed flow* from the source node u to destination node v . A **flow graph** $G(V, E, X)$ is a *directed* graph, where V is the set of vertices ($|V| = N$), E is the set of edges, and $X \in \mathbb{R}_+^{N \times N}$ is the flow matrix denoting the observed edge flows, where $x_{u,v} = 0, \forall (u, v) \notin E$.

By convention, a negative edge flow indicates that the flow moves in a direction opposite to the direction of the edge. We can always change the sign of the flow to positive as long as we change the edge direction. Therefore, without loss of generality, we assume that all edge flows are non-negative, i.e., $x_{u,v} \geq 0, \forall (u, v) \in E$. Moreover, in cases like human crowd flows, some node pairs are likely to have flows in both directions (say, $\tilde{x}_{u,v} \geq \tilde{x}_{v,u} > 0$). In this work, we are only interested in *net-flows*, that is $x_{u,v} = \max(0, \tilde{x}_{u,v} - \tilde{x}_{v,u})$. We refer to *net-flow* as *flow* throughout this paper.

Problem: Given a set of flow graphs $\{G_1, \dots, G_L\}$ sampled from a ground-truth distribution (P_{data}), our aim is to *implicitly* learn to mimic these graphs’ complex characteristics such that the graphs (P_θ) generated by our model can simulate both the *graph topology* and the *flow dynamics* found in data. The proposed framework should be generic enough to capture various flow dynamics intrinsic to numerous real-world networks. (Examples are given in Section 4).

In the remainder of this Section, we first discuss how existing generative models for undirected (topological) graphs can be adapted to our problem. Next, we introduce an implicit flow graph generative model (named *FlowGEN*) that is designed to capture the graph structure as well as complex flow characteristics observed in real data, being the first end-to-end graph generative model to generate FGs from a domain-agnostic standpoint.

3.1 A Two-step Approach

One can think of most (undirected) graph generative models as a parametric function $\mathcal{H} : \mathcal{Z} \rightarrow \{0, 1\}^{N \times N}$ (typically a neural network of some kind) that given samples drawn from a prior distribution, learns to generate graphs, ultimately in the form of an adjacency matrix or its counterparts. This is true whether one generates the entire graph at once [5] or as a sequential process of adding new nodes and edges [48]. It is possible to adapt these models to the generation of flow graphs by a two-step model in which the second step learns an additional flow function $\mathcal{F} : \{0, 1\}^{N \times N} \rightarrow \mathbb{R}^{N \times N}$ that takes an adjacency matrix A as input and generates a non-negative flow matrix whose values correspond to the non-zero elements of A . Within this two-step framework (that can be summarized as $\mathcal{F} \circ \mathcal{H}$), one can choose to use any undirected graph generative model as function \mathcal{H} that is learnt on undirected versions of the observed FGs. We focus on the learning of \mathcal{F} , as illustrated in Figure 1.

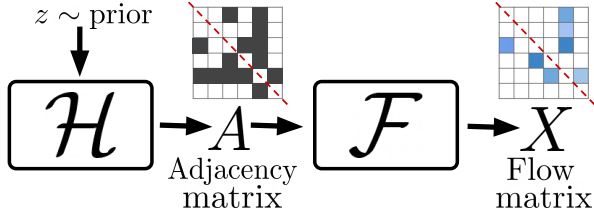


Figure 1: An illustration of our two-step approach for generating flow graphs.

The problem of learning \mathcal{F} is closely related to flow estimation, which can be formulated as a semi-supervised learning problem on graphs [31]. This problem was most recently studied by [25, 40] for a single FG snapshot, where the goal is to predict edge flows based on the graph topology and partial flow observations. [25] achieves this by solving a constrained optimization problem involving domain-specific physical constraints (flow conservation), while [40] further relaxes these constraints and instead enforces them as a regularizer whose weights are learned based on various node/edge features. However, these methods cannot be applied to our setting because we do not assume access to other information such as partial flows (or contextual features), and we aim to learn generative models in a domain-agnostic manner based purely on data.

We now discuss various ways of constructing \mathcal{F} . Similar to [47], given a graph in the form of an adjacency matrix $A \in \{0, 1\}^{|V| \times |V|}$, we first generate node features $\mathcal{M} \in \mathbb{R}^{|V| \times d_k}$ as a standard low-dimensional spectral embedding [4] based on A . Notice that this transformation allows us to handle variable sized graphs. Next, we learn node states based on the node features using a standard two-layer MLP:

$$H = \text{MLP}(\mathcal{M}) = \text{ReLU}(\mathcal{M}W_0)W_1 \quad (1)$$

Alternatively, we can better utilize the graph topology by using a GCN model instead of MLP:

$$H = \text{GCN}(\mathcal{M}, A) = \tilde{A}\text{ReLU}(\tilde{A}\mathcal{M}W_0)W_1 \quad (2)$$

where $H \in \mathbb{R}^{|V| \times d_l}$, $W_0 \in \mathbb{R}^{d_k \times d_l}$, and $W_1 \in \mathbb{R}^{d_l \times d_l}$. $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is symmetric normalized adjacency matrix of A , and D is the diagonal degree matrix with $D_{ii} = \sum_{j=1}^{|V|} A_{ij}$. Lastly, we adapt a final linear transformation to predict edge flows based on the topological encoding of source ($h_s \in \mathbb{R}^{d_l}$) and target ($h_t \in \mathbb{R}^{d_l}$) nodes:

$$\mathbf{g}(h_s, h_t) = (h_s - h_t)W_2 \quad (3)$$

where $W_2 \in \mathbb{R}^{d_l \times 1}$, and $h_i = H_{i,:}$ for $i \in V$. The bias terms are omitted for convenience.

Notice that $\mathbf{g}(h_s, h_t)$ intrinsically captures the *edge direction* between nodes s and t , where $\mathbf{g}(h_s, h_t) = -\mathbf{g}(h_t, h_s)$. This ensures consistency with our problem setting, *i.e.*, a negative flow means movement against the orientation of the edge. Therefore, the respective directions of non-negative flows are assigned for each node pair $(s, t) \in E$, such that $s \rightarrow t$ when $\mathbf{g}(h_s, h_t) > \mathbf{g}(h_t, h_s)$.

Finally, we can define the flow function \mathcal{F} as a stack of \mathbf{g} functions applied on each edge of the input graph. All layers of \mathcal{F} can be trained jointly using graphs from P_{data} , with MSE loss computed between predicted ($\mathbf{g}(h_s, h_t)$) and ground-truth ($x_{s,t}$) flows. We refer to our Appendix for all training details including the formulation of the loss function, hyper-parameters, and more.

3.2 FlowGEN Framework

Though the two-step approach presented in the previous section can reuse existing graph generative models, it does not account for the joint relationship between graph structure and flow distribution. To this end, we introduce FlowGEN (illustrated in Figure 2), an implicit **flow** graph **generative** model that can generate FGs in an end-to-end fashion based on the GAN framework [18]. As in the standard GAN framework, our model consists of two components trained jointly: (1) A *generator* \mathcal{G}_θ that takes a sample from a prior distribution and learns to generate flow graph samples that approximate those drawn from the dataset, and (2) a *discriminator* \mathcal{D}_ϕ that learns to distinguish whether an input graph is sampled from the dataset or the generator.

3.2.1 Generator. The generator \mathcal{G}_θ (parameterized by θ) takes as input a random vector $z \in \mathbb{R}^{d_0}$ sampled from $\mathcal{N}(0, I_{d_0})$ and outputs a flow graph. Precisely, a flow matrix $X \in \mathbb{R}_+^{N \times N}$ is computed by passing the sampled vector z through multiple layers of fully connected neural networks (MLP):

$$X = \text{ReLU}\left(f_\theta^k \circ \dots \circ f_\theta^1(z)\right) \quad (4)$$

where \circ represents function composition. We use ReLU activations for all layers including the final layer to obtain a non-negative flow matrix. Note that the mapping $\theta \mapsto P_\theta$ is continuous (P_θ denotes the model distribution) which differentiates our problem space from the discrete graph generative models [34, 48].

3.2.2 Discriminator. The discriminator \mathcal{D}_ϕ (parameterized by ϕ) takes a flow graph as input and outputs a scalar value indicating whether an input graph is likely to be sampled from the dataset or the generator. Our discriminator is a customized (flow) graph neural network (GNN) consisting of three main components: (1) permutation invariant flow-pooling layer, (2) bi-directional neural message-passing layers, and (3) an attention-based readout layer, each designed to collectively account for domain-agnostic and complex flow dynamics in directed flow graphs.

More formally, given a flow graph $G(V, E, X)$, we first compute two node-level flow (feature) vectors—*in* and *out*—using a permutation invariant flow-pooling function over the column and row vectors of the flow matrix, respectively.

$$\text{in}h_u^1 = f_{\text{pool}}(X_{:,u}), \text{ out}h_u^1 = f_{\text{pool}}(X_{u,:}) \quad u \in V \quad (5)$$

Here, $f_{\text{pool}}(x) = [\text{MEAN}(x); \text{MAX}(x); \text{SUM}(x)]$ is a stack of element-wise pooling aggregators—the symbol ‘;’ denotes concatenation and $x \in \mathbb{R}^d$. $X_{:,u}$ and $X_{u,:}$ $\in \mathbb{R}^{|V|}$ indicate the columns and rows of X , which, respectively, hold the incoming and outgoing flow information of node u from and to its neighbors. Node flow vectors $\text{in}h_u^1$ and $\text{out}h_u^1 \in \mathbb{R}^{d_1}$ ($d_1 = 3$ in this case) are then used as initial hidden states to *in* and *out* neural message-passing channels, where respective node updates are computed independently as follows:

$$\text{in}h_u^{t+1} = f_i^t(\text{in}h_u^t, \sum_{v \in G_{\text{out}}(u)} m_i^t(\text{in}h_v^t, x_{u,v})) \quad (6)$$

$$\text{out}h_u^{t+1} = f_o^t(\text{out}h_u^t, \sum_{v \in G_{\text{in}}(u)} m_o^t(\text{out}h_v^t, x_{v,u})) \quad (7)$$

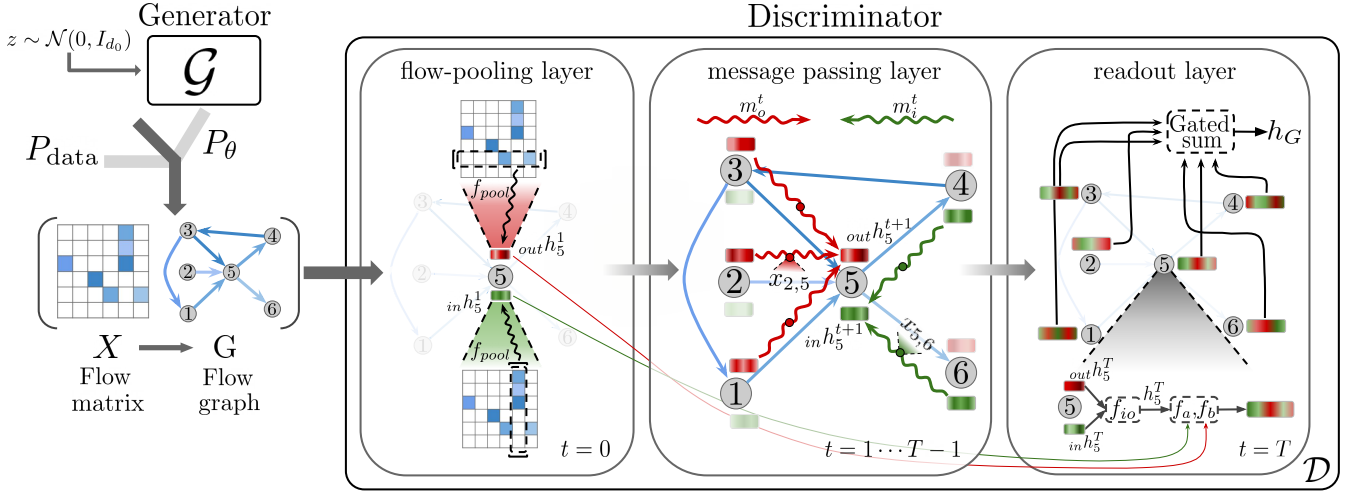


Figure 2: A high-level illustration of FlowGEN architecture (*best viewed in color*). The left part pictures the adversarial training process. The right part shows a detailed sketch of our discriminator: (1) The flow pooling layer assigns two initial feature vectors (*in* and *out*) to each node in a permutation-invariant manner. (2) The message-passing layers comprise bi-directional channels that exchange *in* and *out* flow messages (spiral arrows) *independently* in opposing directions. (3) The read-out layer fuses *in* and *out* states of each node, and further aggregates them to construct a *global* flow graph representation. All components are trained jointly. See Section 3.2 for details.

where, for each node u at layer $t \in \{1, \dots, T-1\}$, the *in* and *out* message functions (m_i^t and m_o^t), respectively take as input the concatenation of *out* and *in* neighbors' corresponding to hidden states and edge flow values to compute latent *flow message* vectors. The *in* and *out* node update functions (f_i^t and f_o^t) take as input the concatenation of current *in* and *out*-states of u and aggregated flow messages from its neighbors to update its next hidden states $in h_u^{t+1}$ and $out h_u^{t+1} \in \mathbb{R}^{d_{i+1}}$ accordingly. $\{f_i^t, m_i^t, f_o^t, m_o^t | t = 1, \dots, T-1\}$ are all modeled independently as MLPs, with no weight sharing.

After $T-1$ message passing layers, we apply another node update function, which takes as input the *in* and *out* node states, and combines them into final node representations.

$$h_u^T = f_{io}(in h_u^T, out h_u^T) \quad u \in V \quad (8)$$

where f_{io} is another MLP. The vector $h_u^T \in \mathbb{R}^{d_T}$ is a fused latent representation of node u , capturing *in* and *out* flow dynamics conditioned on its neighbors as well as its relative location in the graph.

We aggregate these node representations and obtain a latent graph representation $h_G \in \mathbb{R}^{d_G}$ using an attention-based readout layer (a.k.a. *gated sum*) following [33]:

$$h_G = \sum_{u \in G} \sigma(f_a(\tilde{h}_u)) \odot f_b(\tilde{h}_u) \quad (9)$$

where $\tilde{h}_u = [h_u^T; h_u^1] \in \mathbb{R}^{d_T+2d_1}$ and $h_u^1 = [in h_u^1; out h_u^1] \in \mathbb{R}^{2d_1}$ is the concatenation of initial *in* and *out* flow vectors (Eq. 5). This is loosely analogous to skip connections in residual networks [22] which we empirically find to be effective in capturing global flow statistics. The first term in the summation (Eq. 9) essentially serves as a soft attention mechanism assigning contextual importance scores to nodes, where σ is the *sigmoid* activation and \odot denotes element-wise multiplication. The context functions f_a and f_b are again modeled as small MLPs. Note that such gated sum can model

injective multiset functions, and is invariant to input (node) order. Lastly, h_G is processed by a final MLP layer (f_G) which outputs a graph-level scalar. Model parameters $\{\theta, \phi\}$ are trained jointly using the Wasserstein GAN objective [2] with gradient penalty [21]. For all training details (hyper-parameters, activation functions etc.), see the Appendix.

3.3 Discussion

损失函数

We now present a brief rationale behind the design of some of FlowGEN's components. Since we generate the entire flow graph at once, this limits us to graphs whose sizes do not exceed a certain threshold. However, generating an entire flow graph at once is more appropriate for capturing global statistics of flows (e.g. number of nodes with conserved flows) and graph topology (e.g. number of k -cycles), as well as correlations between the two (e.g. hotspot nodes and their relative locations in the graph). That said, our model is capable of generating graphs with varying number of edges and fewer nodes than the predetermined maximum (by removing disconnected nodes). In addition, since f_{pool} is invariant to permutations of the flow matrix and can handle flow distributions of varying sizes, our discriminator also remains permutation invariant and is able to model variable-sized flow graphs.

One potential shortcoming of FlowGEN lies in its generator, which has a quadratic memory footprint in the number of nodes, similar to other non-autoregressive models [5, 35, 37]. Here we emphasize that our primary goal is to generate high-quality flow graphs. For larger flow graphs, one can resort to the proposed two-step approach by leveraging a more scalable topological graph generative model [11].

4 EXPERIMENTS

In this section, we present an experimental analysis of the proposed approaches on both synthetic and real graph datasets with various

flow characteristics (Section 4.2). We also examine the diversity of the generated graphs (Section 4.3), analyze the topology of the generated graphs (Section 4.3), and perform an ablation study of our model components (Section 4.5). Additional results are presented in the Appendix, including multiple case studies demonstrating the downstream utilities of the generated flow graphs.

4.1 Experimental Setup

Datasets.

(1) **Power:** European power transmission graphs [24] where nodes ($|V|=25$) represent power infrastructure (buses, plants etc.) belonging to European countries, edges ($|E|_{avg}=45$) represent transmission lines between the countries, and edge flows measure the **total active power** being transmitted through these lines. We randomly sample 2000 hourly snapshots from the year 2013.

(2) **Water:** 1000 directed barbell graphs with 20 nodes and 10-40% of edges inside the communities removed uniformly at random. Flows are synthetically generated based on a simple **water distribution** process with either a *source* or a *sink* node assigned to each community. For each node pair (u, v) , the flow is the *net* amount of water that passed through either direction during the simulation.

(3) **Taxi Trips:** Real flow graphs with nodes and edges representing taxi zones in Manhattan, NYC, and routes between zones, respectively. Flows are net **volumes of passengers** traveled between zones during 584 different snapshots (weekdays in [2017,2019]). We create two versions of this dataset, with data from **8-9am** ($|V|_{max} = 32, |E|_{avg} \approx 133.1$) and **6-7pm** ($|V|_{max} = 32, |E|_{avg} \approx 156.2$).

Evaluation Metrics. Evaluating generative models is known to be challenging [44]. In our case, this evaluation requires a comparison between the *generated* and the *ground truth* flow graphs while accounting for both the graph structures and edge flow characteristics. To this end, we identify a comprehensive set of metrics which allow us to quantitatively evaluate the effectiveness of our model on this novel task. These metrics include: (1-2) **in/out degree distributions**, (3) **edge flow distributions**, (4) **node divergence distributions**, and (5-6) **number of directed k-cycles** ($k \in \{3, 4\}$). We define the *divergence* on node u as the difference between total in-flow and total out-flow, normalized by their maximum, in order to compare graphs across different datasets. Formally, $X_u^{div} = (X_u^{out} - X_u^{in}) / \max(X_u^{out}, X_u^{in})$ where $X_u^{out} = \sum_{\{u \rightarrow v\}} x_{u,v}$ and $X_u^{in} = \sum_{\{v \rightarrow u\}} x_{v,u}$. Note that $X_u^{div} \in [-1, 1]$ for all $u \in V$ and node u is a *source* (*sink*) node of the flow graph if $X_u^{div} \approx 1$ ($X_u^{div} \approx -1$). There can be multiple source and sink nodes in a graph.

For graph-level metrics such as number of directed k-cycles, we compute the average statistics of the entire set. For the remaining node-level and edge-level metrics, following previous work [34, 48], we compute the maximum mean discrepancy (MMD) over the two sets of distributions using the total variation (TV) distance.

Methods. To the best of our knowledge, no baseline is available for the novel task of generating FGs that exhibit the characteristics of a given set of (ground-truth) graphs. Therefore, we experiment with approaches introduced in Section 3 including FlowGEN and a variety of two-step alternatives. Following our discussion on two-step approaches (Section 3.1), we consider two state-of-the-art deep (undirected) graph generative models —NetGAN [5] and GRAN [34]— as the first step (function \mathcal{H}), which are trained on the undirected versions of the datasets. As for the second step

(function \mathcal{F}), we implement three approaches namely FMLP (based on Equation 1), FGCN (based on Equation 2), and DFF (short for Divergence-Free Flows proposed by [25]). This results in six variants: *NetGAN-FMLP*, *NetGAN-FGCN*, *NetGAN-DFF*, *GRAN-FMLP*, *GRAN-FGCN*, *GRAN-DFF*. More details on these methods are provided in the Appendix.

Experiment settings. Following [34], for each dataset, we randomly split the graphs into train (80%) and test (20%) sets. We use 20% of the training graphs as the validation set. For fair comparison, we use the same splits for all the models, and each model—at testing time—generates the same number of graph samples as the test set. For all the methods, we fix the size of input and output graphs as the size of the largest graph in the dataset. All evaluations are performed on the test set.

4.2 Experimental Results

Table 1 reports the performance of all seven methods on all four datasets. The set of metrics we use evaluates graphs from varying perspectives and the proposed end-to-end model consistently displays top results on all of them with very few exceptions. In more detail, we observe that:

- FlowGEN excels at capturing global cyclic trends in all datasets compared to other methods. This demonstrates that FlowGEN is able to successfully replicate various higher-order patterns of directed graphs including acyclic water flows and real human-crowd flows at different rush hours.
- FlowGEN fits the underlying edge flow and node divergence distributions in the data considerably better than the competing methods on nearly all the measures. This suggests that while FlowGEN is effective in learning the edge flow distributions directly from data, it also learns the *coupling between edge flows and graph topology* which is intrinsically tied to the node-divergence distribution.
- GRAN-based two-step variants are particularly good at recovering degree statistics and often outperform other methods regarding in/out-degree distributions. This is a natural outcome given that GRAN is considered to be the state-of-the-art (undirected) graph generative model. FlowGEN, on the other hand, has competitive—if not better, as for Water—scores than GRAN variants regarding these metrics, while consistently outperforming on flow-related metrics.

We display graph examples generated by FlowGEN and the best-performing two-step method, together with training samples for the taxi trips datasets in Figure 3. We can observe that, in addition to generating similar graph structures to those in the training set, FlowGEN remarkably exhibits similar numbers of *source* (green) and *sink* (red) nodes (as opposed to divergence-free nodes) together with their relative locations in the graph. For instance, the morning hour exhibits unique flow patterns where the vast majority of mobility flow is directed towards a few high centrality nodes corresponding to business centers in Manhattan. Conversely, during the evening hour, such nodes become the source of mobility flows being directed towards periphery nodes, coupled with more cycles due to variations in travel patterns.

To further illustrate these findings, Figure 4 plots node-level flow divergence distributions of graphs randomly sampled from training data (1-5), FlowGEN (6-10), NetGAN-FGCN (11-15), and GRAN-FGCN (16-20) for both dataset variants, with the top plot corresponding to 8-9am and the bottom plot corresponding to 6-7pm.





| | Simulated Data | | | | | | | Taxi Trips | | | | | | |
|-------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|---|---|--------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|---|---|--------------|
| | Top: Power / Bottom: Water | | | | | | | Top: 8-9am / Bottom: 6-7pm | | | | | | |
| | In-deg. dist. | Out-deg. dist. | $\{x_{u,v}\}$ | $\{X_u^{div}\}$ |  |  | Avg. rank | In-deg. dist. | Out-deg. dist. | $\{x_{u,v}\}$ | $\{X_u^{div}\}$ |  |  | Avg. rank |
| test data | — | — | — | — | 2.58 | 2.33 | — | — | — | — | — | 10.34 | 17.71 | — |
| NetGAN-FMLP | $1.69e^{-2}$ | $3.73e^{-2}$ | $4.71e^{-2}$ | $7.26e^{-2}$ | 2.21 | 1.45 | 5.2 | $3.01e^{-2}$ | $4.45e^{-2}$ | $9.81e^{-2}$ | $6.62e^{-2}$ | 2.56 | 5.68 | 4.5 |
| NetGAN-FGCN | $1.88e^{-2}$ | $1.21e^{-2}$ | $1.89e^{-2}$ | $5.95e^{-2}$ | 2.01 | 1.77 | 3.6 | $3.56e^{-2}$ | $2.49e^{-2}$ | $5.32e^{-2}$ | $3.73e^{-2}$ | 6.74 | 10.85 | 2.6 |
| NetGAN-DFF | $5.85e^{-2}$ | $2.76e^{-2}$ | $4.22e^{-2}$ | 0.126 | 3.74 | 5.01 | 5.8 | 0.128 | $4.76e^{-2}$ | $8.51e^{-2}$ | 0.322 | 26.41 | 67.84 | 5.8 |
| GRAN-FMLP | $3.16e^{-3}$ | $2.28e^{-3}$ | $3.94e^{-2}$ | $5.62e^{-2}$ | 4.24 | 3.16 | 3.6 | $1.96e^{-2}$ | $2.33e^{-2}$ | 0.122 | $5.45e^{-2}$ | 17.53 | 55.37 | 3.8 |
| GRAN-FGCN | $1.18e^{-3}$ | $1.79e^{-3}$ | $7.02e^{-3}$ | $4.51e^{-2}$ | 3.98 | 3.14 | 2.3 | $2.28e^{-2}$ | $9.81e^{-3}$ | $6.77e^{-2}$ | $2.85e^{-2}$ | 34.19 | 110.51 | 3.3 |
| GRAN-DFF | $1.56e^{-2}$ | $1.73e^{-2}$ | $3.61e^{-2}$ | 0.164 | 7.23 | 6.62 | 5.6 | $8.97e^{-2}$ | $1.11e^{-2}$ | 0.101 | 0.315 | 61.24 | 263.43 | 5.6 |
| FlowGEN | $1.35e^{-2}$ | $9.36e^{-3}$ | $3.73e^{-3}$ | $2.98e^{-3}$ | 2.66 | 2.41 | 1.6 | $1.42e^{-2}$ | $3.47e^{-2}$ | $8.32e^{-2}$ | $1.99e^{-2}$ | 11.76 | 28.02 | 2.1 |
| test data | — | — | — | — | 0.0 | 0.0 | — | — | — | — | — | 34.10 | 92.56 | — |
| NetGAN-FMLP | $4.92e^{-2}$ | $4.53e^{-2}$ | 0.107 | 0.497 | 5.58 | 6.89 | 5.3 | $6.44e^{-2}$ | $1.27e^{-2}$ | $3.07e^{-2}$ | $4.32e^{-2}$ | 8.59 | 20.25 | 5.2 |
| NetGAN-FGCN | $2.65e^{-2}$ | $2.16e^{-2}$ | $8.94e^{-2}$ | 0.419 | 4.84 | 6.07 | 3.2 | $4.09e^{-2}$ | $1.32e^{-2}$ | $2.81e^{-2}$ | $2.96e^{-2}$ | 11.45 | 27.02 | 4.2 |
| NetGAN-DFF | $7.52e^{-2}$ | $7.64e^{-2}$ | $8.99e^{-2}$ | 0.264 | 9.16 | 12.87 | 5.3 | $5.03e^{-2}$ | $5.67e^{-2}$ | $9.34e^{-2}$ | 0.243 | 28.36 | 72.30 | 4.8 |
| GRAN-FMLP | $3.91e^{-2}$ | $4.18e^{-2}$ | $9.98e^{-2}$ | 0.464 | 9.78 | 14.96 | 3.5 | $3.14e^{-2}$ | $5.98e^{-3}$ | $3.04e^{-2}$ | $2.44e^{-2}$ | 22.17 | 75.58 | 2.6 |
| GRAN-FGCN | $1.81e^{-2}$ | $1.78e^{-2}$ | $8.51e^{-2}$ | 0.408 | 8.23 | 12.56 | 3.0 | $1.39e^{-2}$ | $8.76e^{-3}$ | $2.66e^{-2}$ | $2.21e^{-2}$ | 53.21 | 205.41 | 3.0 |
| GRAN-DFF | $3.66e^{-2}$ | $4.35e^{-2}$ | $8.16e^{-2}$ | 0.235 | 13.54 | 25.32 | 4.5 | $5.12e^{-3}$ | $4.27e^{-2}$ | 0.140 | 0.289 | 95.18 | 417.35 | 5.8 |
| FlowGEN | $1.99e^{-2}$ | $1.76e^{-2}$ | $2.41e^{-2}$ | $5.53e^{-2}$ | 3.39 | 5.82 | 1.2 | $6.77e^{-3}$ | $1.23e^{-2}$ | $7.44e^{-2}$ | $1.93e^{-2}$ | 23.69 | 81.48 | 2.3 |

Table 1: Flow graph generation results for simulated (*left*) and real-world (*right*) datasets. Metrics from left to right: (1-2) in&out degree distribution, (3) edge flow distribution, (4) node divergence distribution, (5-6) average number of directed 3&4-cycles. For MMD scores (1-4), the smaller the better; for average statistics (5-6), the closer to test data, the better. Last column (7) shows average rank of models per each dataset for the reader’s convenience. The symbol ‘—’ means not applicable as MMD scores are computed with respect to test data. *FlowGEN* consistently ranks top across all four datasets by showing superior performance on flow-related metrics (3-6), while being competitive with other methods on topological metrics (1-2).

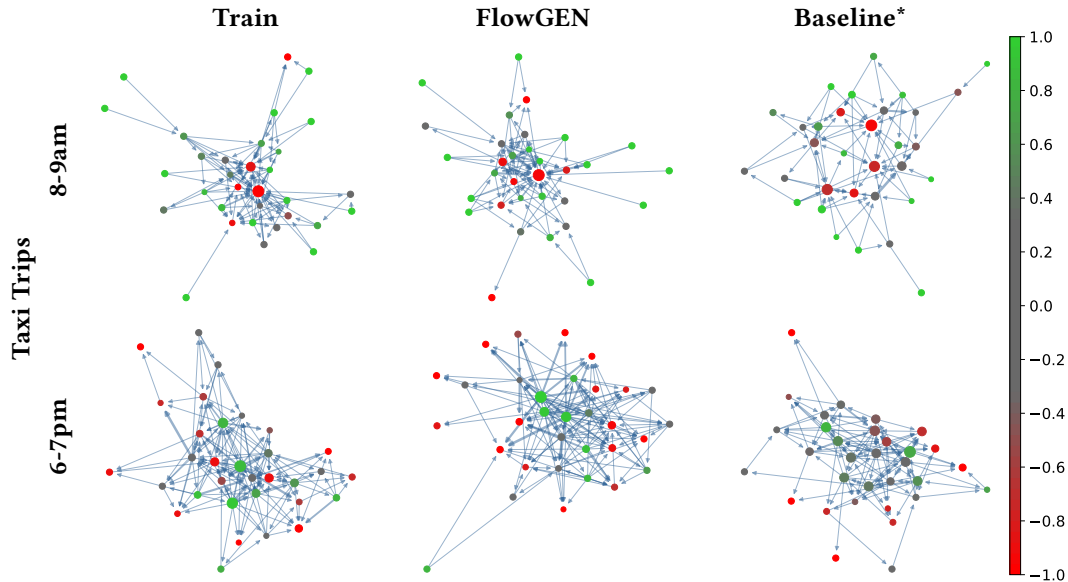


Figure 3: Visualization of flow graphs (FGs) sampled from train splits and top-2 models for taxi trips. “*” indicates the best performing baseline variant (e.g., *NetGAN-FGCN* for 8-9am). Node colors represent flow divergence with green for source, red for sink and gray for divergence-free node. Node sizes indicate total in/out flow normalized per graph. See color map on the right for scale. FGs generated by *FlowGEN* exhibit similar patterns to those observed in data regarding the coupling of graph structure and flow distributions.

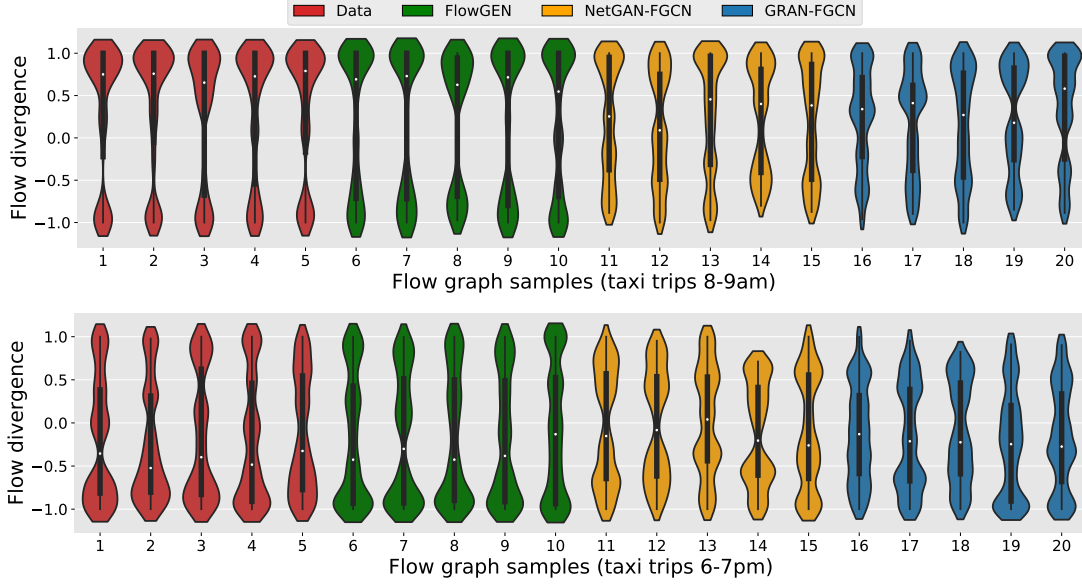


Figure 4: Node-level flow divergence distributions of randomly sampled graphs from training data (1-5), FlowGEN (6-10), NetGAN-FGCN (11-15), and GRAN-FGCN (16-20) for taxi trips 8-9am (top) and 6-7pm (bottom) variants. The horizontal axis corresponds to color-coded graph samples (five each), while the vertical axis represents flow divergence distributions. FlowGEN successfully replicates the flow divergence distribution observed in data, outperforming the two-step approaches.

The results show that FlowGEN consistently matches the ground-truth distribution observed in both settings of the data, while the two-step approaches NetGAN-FGCN and GRAN-FGCN struggle to reproduce such complex characteristics. The ability to capture the joint relationship between underlying graph topology and global flow distribution is one of the key desiderata in generating *realistic* flow graphs observed in diverse settings.

Through both qualitative and quantitative analyses, we conclude that FlowGEN can effectively capture flow dynamics of graphs with vastly differing characteristics—being able to learn complex distributions like divergence-free flows as well as more natural distributions of flows like human crowds and power transmissions. Furthermore, we refer to the Appendix for multiple case studies demonstrating the downstream utilities of generated flow graphs.

4.3 Diversity Analysis

We also analyze the diversity of generated graphs, an important aspect in assessing generative models’ capacity as they are known to suffer from mode collapse [2, 43]. In the context of graph generative models, the notion of diversity is typically defined for undirected graphs and does not directly apply to our case (e.g. graphs with the same topology but different flow distributions are considered different). We observe that FlowGEN is able to generate diverse sets of flow graphs with varying topological and flow characteristics.

Figure 5 demonstrates the distribution of generated graph structures observed in all four datasets, as well as graphs generated by NetGAN, GRAN, and FlowGEN. To generate this plot, we choose (undirected) degree assortativity coefficient [36] as our metric, which is a graph-level scalar value allowing us to conveniently examine and compare structural diversity across generated graphs. For FlowGEN, we use the undirected versions of the generated flow graphs for a fair comparison. Our results show that while FlowGEN

and GRAN generate diverse underlying graph structures, NetGAN tends to generate graphs with similar structures, particularly for Taxi 8-9am (Figure 5a) and Power (Figure 5c) datasets.

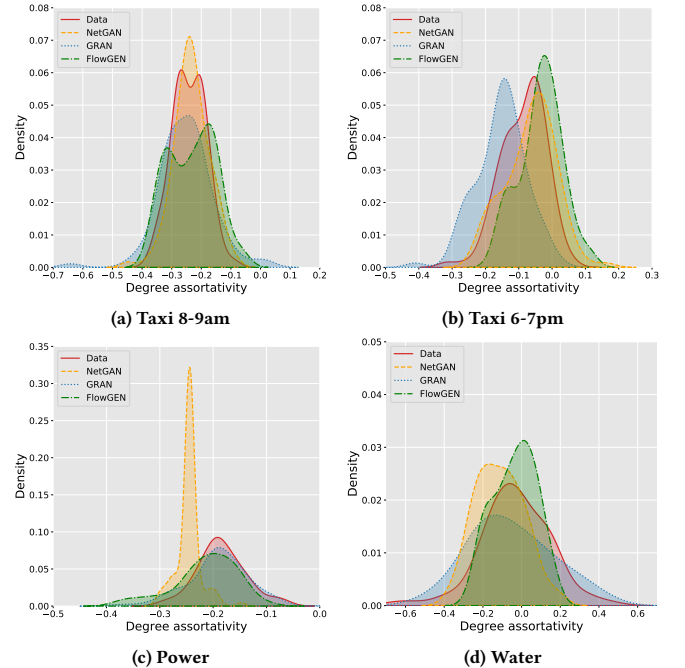


Figure 5: Distributions of degree assortativity coefficients observed in graphs from training data, NetGAN, GRAN, and FlowGEN for all four datasets used in our experiments.





| | Simulated Data | | | | | Taxi Trips | | | | |
|-----------|------------------------------|-----------------|---------------------------------|---|---|------------------------------|-----------------|---------------------------------|---|---|
| | Power (top) / Water (bottom) | | | | | 8-9am (top) / 6-7pm (bottom) | | | | |
| | Deg. dist. | Clus. coeff. | Assort- ativity |  |  | Deg. dist. | Clus. coeff. | Assort- ativity |  |  |
| test data | — | — | -0.196 | 16.0 | 23.0 | — | — | -0.255 | 275.3 | 2031.4 |
| E-R | 0.177 | 0.114 | $-8.55e^{-2}$ | 7.92 | 18.63 | $8.53e^{-2}$ | $1.99e^{-2}$ | $-7.29e^{-2}$ | 113.4 | 749.5 |
| B-A | 0.479 | 0.998 | -0.428 | 0.0 | 0.0 | $6.80e^{-2}$ | $2.02e^{-2}$ | -0.195 | 120.2 | 771.2 |
| NetGAN | $5.32e^{-2}$ | 0.172 | -0.247 | 7.95 | 12.19 | $3.54e^{-2}$ | $2.03e^{-2}$ | -0.223 | 87.3 | 416.7 |
| GRAN | $3.86e^{-3}$ | $4.27e^{-2}$ | -0.188 | 16.65 | 25.60 | $5.32e^{-3}$ | $1.97e^{-2}$ | -0.251 | 231.9 | 1768.8 |
| FlowGEN | $1.91e^{-2}$ | 0.106 | -0.221 | 13.20 | 21.23 | $1.48e^{-2}$ | $1.99e^{-2}$ | -0.234 | 256.4 | 2170.2 |
| test data | — | — | $-2.63e^{-2}$ | 36.69 | 108.84 | — | — | $-7.86e^{-2}$ | 340.3 | 2523.0 |
| E-R | $2.48e^{-2}$ | $2.51e^{-2}$ | -0.131 | 17.5 | 57.96 | $9.21e^{-2}$ | $2.01e^{-2}$ | $-6.36e^{-2}$ | 173.1 | 1277.0 |
| B-A | 0.101 | $4.62e^{-2}$ | -0.322 | 13.32 | 38.78 | $6.11e^{-2}$ | $2.00e^{-2}$ | -0.186 | 197.4 | 1460.3 |
| NetGAN | $2.59e^{-2}$ | $2.23e^{-2}$ | -0.138 | 22.2 | 56.4 | $6.20e^{-2}$ | $2.02e^{-2}$ | $-7.23e^{-2}$ | 94.5 | 441.4 |
| GRAN | $4.20e^{-3}$ | $2.08e^{-2}$ | $-5.54e^{-2}$ | 38.37 | 116.39 | $3.01e^{-3}$ | $2.00e^{-2}$ | -0.152 | 358.5 | 2876.4 |
| FlowGEN | $9.27e^{-3}$ | $2.19e^{-2}$ | $-3.49e^{-2}$ | 36.01 | 102.38 | $1.57e^{-2}$ | $1.99e^{-2}$ | $-6.89e^{-2}$ | 299.7 | 2671.3 |

Table 2: Results with respect to graph topology. Metrics from left: (1) degree and (2) clustering coefficient distributions; average (3) degree assortativity and (4-5) number of 3&4-cycles. For MMD scores (1-2), the smaller the better; for average statistics (3-5), the closer to test data, the better.

4.4 Reproducing graph topology

The previous experimental results given in Table 1 primarily assess the generated graphs from a flow-centric standpoint, mainly because our objective is to generate realistic flow graphs. That said, an important step towards achieving this goal is the ability to reproduce underlying graph topologies found in data. Thus, we present a comprehensive analysis on all four datasets with additional graph statistics in Table 2. We also include two more traditional graph generative models as baselines: Erdos-Renyi (E-R) [15] and Barabasi-Albert (B-A) [1]. Results show that FlowGEN consistently yields competitive results against neural baselines while outperforming the traditional approaches.



| | In-deg. dist. | Out-deg. dist. | $\{x_{u,v}\}$ | $\{X_u^{div}\}$ |  |  |
|-----------|------------------|-------------------|---------------|-----------------|---|---|
| test data | — | — | — | — | 2.58 | 2.33 |
| FlowGEN | $1.35e^{-2}$ | $9.36e^{-3}$ | $3.73e^{-3}$ | $2.98e^{-3}$ | 2.66 | 2.41 |
| w/o FPL | $2.51e^{-2}$ | $3.17e^{-2}$ | $4.19e^{-3}$ | $9.23e^{-3}$ | 2.60 | 2.56 |
| w/o BMPL | $4.25e^{-2}$ | $8.63e^{-2}$ | $4.39e^{-3}$ | $5.73e^{-2}$ | 0.85 | 1.48 |
| w/o ARL | $1.81e^{-2}$ | $2.38e^{-2}$ | $3.08e^{-3}$ | $7.16e^{-3}$ | 1.83 | 3.64 |

Table 3: Ablation study of model components on Power dataset.

4.5 Ablation Study

Lastly, we conduct a detailed ablation study in order to demonstrate the contributions of each FlowGEN component by disabling these components one by one, and replacing them with standard approaches from the literature. We name these variants as follows:

- FlowGEN without Flow-Pooling Layer (w/o FPL): This variant disables our flow-pooling component and instead uses Gaussian random vectors as initial *in* and *out* node features ($[in h_u^1; out h_u^1] \sim \mathcal{N}(0, I_6)$). This is a well-known approach for cases where external node features are not available [47].

- FlowGEN without Bidirectional Message-Passing Layer (w/o BMPL):

This variant disables our bidirectional message passing scheme where the node states are computed using two independent (*in* and *out*) neural message-passing channels (Equations 6 and 7). We instead employ a single message-passing channel to simultaneously update both node representations. More formally, for each node u at layer t , the respective node update is computed as follows:

$$h_u^{t+1} = f^t(h_u^t, m_u^t), \quad t = 1, \dots, T-1 \quad (10)$$

where

$$m_u^t = \sum_{v \in G_{in}(u)} m^t(h_v^t, x_{v,u}) + \sum_{v \in G_{out}(u)} m^t(h_v^t, x_{u,v})$$

$$h_u^1 = [in h_u^1; out h_u^1] \quad (11)$$

$\{f^t, m^t | t = 1, \dots, T-1\}$ are again modeled as MLPs.

- FlowGEN without Attention-based Readout Layer (w/o ARL):

This variant replaces the attention-based readout layer [33] with a standard SUM aggregator. More formally, we modify Equation 9 and compute the latent graph representation as $h_G = \frac{1}{|V|} \sum_{u \in V} \hat{h}_u$ which simply assigns equal weights to all nodes in the graph.

Analysis: Table 3 demonstrates that each FlowGEN component contributes to final model performance, where different components have varying effects on different evaluation metrics. We observe that the proposed bidirectional message-passing layer (BMPL) enables our discriminator to better capture *higher-order directed* flow characteristics found in data, which results in significant improvements on related metrics such as in/out degree distributions, node-divergence distribution and directed k -cycles. With that observation, we hypothesize that bidirectional message-passing channels can account for heterogeneous in-flow and out-flow dynamics of nodes. As an example, for certain nodes like divergence-free nodes, larger total in-flow naturally indicates larger total out-flow, while this does not hold for some other nodes like *sink* or *source* nodes, for which the relationship is clearly not linear.

Moreover, while the flow-pooling layer (FPL) does not seem to have a notable impact on directed cyclic trends, it leads to improvements on in/out degree distributions and (more prominently) on node-divergence distribution. This suggests that explicitly assigning rather simple flow statistics of nodes as initial node states—in addition to inducing the key property of permutation invariance—results in improved capacity in capturing global flow characteristics.

The third component of our discriminator is the attention-based readout layer (ARL). Such a layer is widely used in the literature to adjust the contributions of each node to the final graph representation, as intuitively, certain nodes play a more critical role than others [17]. We also leverage this technique to improve our model’s ability to replicate critical nodes in flow graphs (e.g. *source* and *sink* nodes), which directly leads to desirable improvements observed through both qualitative (Figure 3) and quantitative (Table 3) evaluations. More specifically, we observe that ARL leads to enhanced performance on the node-divergence distribution and directed k -cycle count metrics compared to the standard SUM aggregator.

The results discussed in this section justify the different choices made in the design of the FlowGEN architecture. As discussed above, each component contributes to our model’s ability to reproduce one or more desired properties of real flow graphs, which explains why FlowGEN outperforms the baselines (see Table 1).

5 CONCLUSION

We introduce the problem of generating flow graphs, which poses greater challenges than those only related to the graph topology (or even weighted graphs), due to the complexity of higher-order flow patterns, such as flow conservation, cyclic trends, and congestion around hotspots. We further introduce domain-agnostic generative models for FGs, including FlowGEN—our main contribution. At the heart of FlowGEN’s architecture is a custom graph neural network designed to capture the complex coupling between graph topology and edge flows. Our experiments demonstrate how these components collectively address limitations of existing graph generative models to handle flow graphs, enabling FlowGEN to reproduce relevant local and global flow properties in multiple domains.

ACKNOWLEDGMENTS

This project was partially supported by funding from the National Science Foundation under grant IIS-1817046 and the Defense Threat Reduction Agency under grant HDTRA1-19-1-0017.

REFERENCES

- [1] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.
- [2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.
- [3] Aleix Bassolas et al. Hierarchical organization of urban mobility and its connection with city livability. *Nature communications*, 10(1):1–10, 2019.
- [4] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.
- [5] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *ICML*, 2018.
- [6] Alberto Bressan et al. Flows on networks: recent results and perspectives. *EMS Surveys in Mathematical Sciences*, 1(1):47–111, 2014.
- [7] Marc Brockschmidt et al. Generative code modeling with graphs. In *ICLR*, 2019.
- [8] Francesco Calabrese, Giusy Di Lorenzo, et al. Estimating origin-destination flows using mobile phone location data. *IEEE Pervasive Computing*, (4):36–44, 2011.
- [9] Matteo Chinazzi et al. The effect of travel restrictions on the spread of the 2019 novel coronavirus (covid-19) outbreak. *Science*, 368(6489):395–400, 2020.
- [10] Hang Chu, Daiqing Li, et al. Neural turtle graphics for modeling city road layouts. In *ICCV*, 2019.
- [11] Hanjun Dai et al. Scalable deep generative modeling for sparse graphs. In *ICML*, 2020.
- [12] Hermann W Dommel and William F Tinney. Optimal power flow solutions. *IEEE Transactions on power apparatus and systems*, (10):1866–1876, 1968.
- [13] Florian Dörfler et al. Electrical networks and algebraic graph theory: Models, properties, and applications. *Proceedings of the IEEE*, 106(5):977–1005, 2018.
- [14] Florian Dörfler and Francesco Bullo. Synchronization in complex networks of phase oscillators: A survey. *Automatica*, 50(6):1539–1564, 2014.
- [15] Paul Erdős and Alfréd Rényi. On random graphs. *Publicationes mathematicae*, 6(26):290–297, 1959.
- [16] M. Garavello and B. Piccoli. *Traffic Flow on Networks: Conservation Laws Model*. AIMS series on applied mathematics. AIMS, 2006.
- [17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.
- [18] Ian Goodfellow et al. Generative adversarial nets. In *NeurIPS*, 2014.
- [19] Nikhil Goyal et al. Graphgen: a scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*, 2020.
- [20] Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In *ICML*, 2019.
- [21] Ishaan Gulrajani et al. Improved training of wasserstein gans. In *NeurIPS*, 2017.
- [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.
- [23] D.J. Hill and Guanrong Chen. Power systems as dynamic networks. In *2006 IEEE International Symposium on Circuits and Systems*, 2006.
- [24] Jonas Hörsch et al. Pypsa-eur: An open optimisation model of the european transmission system. *Energy strategy reviews*, 22:207–215, 2018.
- [25] Junteng Jia, Michael T Schaub, Santiago Segarra, and Austin R Benson. Graph-based semi-supervised & active learning for edge flows. In *KDD*, 2019.
- [26] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *ICML*, 2018.
- [27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [28] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [29] Ryan Kinney et al. Modeling cascading failures in the north american power grid. *The European Physical Journal B-Condensed Matter and Complex Systems*, 46(1):101–107, 2005.
- [30] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.
- [31] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.
- [32] Yujia Li et al. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.
- [33] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *ICLR*, 2016.
- [34] Renjie Liao et al. Efficient graph generation with graph recurrent attention networks. In *NeurIPS*, 2019.
- [35] Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. Graph normalizing flows. In *NeurIPS*, 2019.
- [36] Mark EJ Newman. Mixing patterns in networks. *Physical review E*, 2003.
- [37] Chenhao Niu et al. Permutation invariant graph generation via score-based generative modeling. In *AISTATS*, 2020.
- [38] J. D. Orth, I. Thiele, and B. Ø. Palsson. What is flux balance analysis? *Nat Biotechnol*, 28(3):245–248, Mar 2010.
- [39] Chence Shi et al. Graphaf: a flow-based autoregressive model for molecular graph generation. In *ICLR*, 2020.
- [40] Arlei Silva et al. Combining physics and machine learning for network flow estimation. *ICLR*, 2021.
- [41] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *ICANN*, 2018.
- [42] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, 2019.
- [43] Akash Srivastava et al. Veegan: Reducing mode collapse in gans using implicit variational learning. In *NeurIPS*, 2017.
- [44] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.
- [45] Petar Veličković et al. Graph attention networks. In *ICLR*, 2017.
- [46] Duncan J Watts and Steven H Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393(6684):440, 1998.
- [47] Carl Yang et al. Conditional structure generation through graph variational generative adversarial nets. In *NeurIPS*, 2019.
- [48] Jiaxuan You et al. Graphrnn: Generating realistic graphs with deep autoregressive models. In *ICML*, 2018.
- [49] Xun Zheng et al. Dags with no tears: Continuous optimization for structure learning. In *NeurIPS*, 2018.

6 APPENDIX

6.1 Case Study: European Power Network

We perform a case study focused on the generation of a particular type of flow graphs, namely power networks. Such networks are often applied in the analysis of large power distribution infrastructure, including their robustness [29] and optimization [12]. Moreover, the dynamics of power networks is governed by physical laws—in this case, Kirchhoff’s laws—posing a great challenge to domain-agnostic generative models for flow graphs. In this case study, we apply a new domain-specific evaluation metric in order to assess whether the generated graphs are consistent with the expected properties of power networks.

Evaluation: Besides the metrics described in Section 4.1, we also apply a more specific one to evaluate power flow graphs. Our goal is to assess whether the generated flows are consistent with the dynamics observed in real power networks. For each generated flow graph $G(V, E, X)$, we compute node divergences $X_u^{div} = (X_u^{out} - X_u^{in})$, where X_u^{in} and X_u^{out} are total in and out flows for node u , respectively. These values capture the net power consumed ($X_u^{div} < 0$) or generated ($X_u^{div} > 0$) at each node in the flow graph. However, notice that different assignments of flows can satisfy the same node divergences—flows can be increased arbitrarily along the cycles in G —and thus one can define a notion of optimality to compare them. We define the optimal flow assignment X^* given node divergences X^{div} and a set of edges E using the following formulation:

$$\begin{aligned} X^* &= \arg \min_X \|(X - X^T)\mathbf{1} - X^{div}\|_2^2 + \lambda \|X\|_F^2 \\ \text{s.t. } &\forall X_{u,v} > 0, (u, v) \in E \end{aligned} \quad (12)$$

where λ (set to 0.1) is a regularization parameter. X^* attempts to satisfy the node divergences as those in X^{div} while also minimizing a regularization term. Besides making the solution unique, this regularization has a physical interpretation from Kirchhoff’s voltage law [13]. Intuitively, it penalizes cyclic flows, which are not driven by differences of potentials, while smoothly distributing flow values as if line resistances are uniform. Therefore, given the same node divergences and graph topology, one would expect an effective model to produce flows that are close to X^* . We define the *efficiency score* of a flow matrix X as $1 - \|X - X^*\|_F$, where both flow matrices are normalized.

| | Eff. score |
|-------------|-------------|
| test data | 0.69 |
| NetGAN-FGCN | 0.47 |
| GRAN-FGCN | 0.52 |
| FlowGEN | 0.58 |

Table 4: Evaluation of power network generated by the different methods using the defined efficiency score. A higher score means better performance.

Experimental Results: Table 4 shows the performance of FlowGEN and the best performing baseline variants on the power dataset. In Section 4.2 we showed that while GRAN variants excel at reproducing topological properties, FlowGEN achieves the best results regarding flow-related metrics (*i.e.*, edge flows, node divergences and directed cycles) at both local and global levels. Similarly, regarding the efficiency score, results show that FlowGEN produces

flow graphs that are closer to optimal compared to the baselines. Moreover, notice that neither FlowGEN nor the baselines take into account the objective function from Equation 12 explicitly. These results give substantial evidence that FlowGEN is able to capture complex domain-specific flow dynamics observed in the data.

6.2 Case Study: Downstream Utility of Generated Flow Graphs

Network flow estimation [25, 40] is one of the mainstream problems that has a broad practical use in many real-world applications, especially in which the availability of flow information can be limited due to their measurements requiring a lot of effort and resources. Moreover, one of the exciting motivations behind high-fidelity synthetic data generation is that it can serve as a valuable data augmentation tool to provide researchers with an opportunity to develop and test analytical tools on numerous real-world problems where certain data limitations and/or privacy measures may apply.

Combining these two motivations, we present an additional case study, which aims to quantitatively analyze whether the generated flow graphs can be employed in combination with the ground-truth data to improve network flow estimation performance. In particular, we conduct the following additional experiments on the Taxi Trips 8-9am dataset. First, we randomly sample 40 flow graphs from the ground-truth data, which are split 20/10/10 for training/validation/testing. Following the same procedure in our two-step framework (see Section 3.1), we train a GCN model on the training set and evaluate its performance on the test set. Next, we further sample 20 more flow graphs, but this time from FlowGEN, which are then combined with the previously sampled graphs for training. We again train another GCN model on the extended training set and evaluate its performance on the same test set used in the previous setting. Training is continued until no improvements are observed on the validation set for 10 consecutive epochs. We repeat the experiments 10 times and compute the average RMSE scores normalized per graph. Our results show that adding the synthetically generated flow graphs by FlowGEN leads to 16.7% relative improvement on the RMSE metric ($8.08e - 2$ vs $6.73e - 2$). This provides a strong evidence towards the broader utility of our framework.

6.3 FlowGEN Training Details

Our generator (\mathcal{G}_θ) comprises a 4-layer MLP with hidden units of $\{64, 128, 256, N^2\}$, where N is the maximum number of nodes assigned accordingly per each dataset. A *ReLU* activation is applied after each layer. The input to the generator is a 16-dimensional vector drawn from a multivariate standard normal distribution. In the discriminator (\mathcal{D}_ϕ), we employ four bi-directional message-passing steps. The *in* and *out* message functions ($\{m_i^t, m_o^t\}_{t=1}^4$) are single layer MLPs without activation, where the hidden units are 16 for all steps. Node update functions ($\{f_i^t, f_o^t\}_{t=1}^4$) are two-layer MLPs with hidden units of $\{32, 16\}$, and *ReLU* activations. In the read-out layer, f_{io} is a single layer MLP with 16 hidden units and a *tanh* activation. The context functions f_a and f_b are both single-layer MLPs with hidden units of 16, and the former is followed by a *sigmoid* activation while the latter has no activation. Lastly, the graph-level representation is processed by f_G which is modeled as a 2-layer linear MLP with hidden units of $\{8, 1\}$. Notice that there is

no activation for the final layer, and the output is a scalar score $\in \mathbb{R}$ rather than a probability. The model parameters $\{\theta, \phi\}$ are trained using mini-batch SGD with the Adam optimizer [27]. The learning rate is set to $1e-3$. L2 regularization is applied for all weights with a penalty of $1e-6$. We set the Wasserstein gradient penalty to 10 as suggested by [21] (applied to the discriminator). To allow for a warm start during the training, for the first 10 epochs, we perform four update steps for the parameters of the discriminator for each single update step of the parameters of the generator (*i.e.*, n_critic ratio = 4:1). The hyper-parameters we tune are the n_critic ratio $\in \{4:1, 2:1, 1:1, 1:2, 1:4\}$ (effective after 10th epoch), the mini-batch size $\in \{32, 64, 128\}$, and the dropout ratio $\in \{0, 0.5\}$ (applied after all MLP layers).

6.4 Training Details of Two-Step Approaches

For both topological graph generative models (NetGAN and GRAN), we set the hyper-parameters based on recommendations made in their respective papers, except for the mini-batch size which is tuned $\in \{32, 64, 128\}$. Recall from Section 4.1 that we also employ three approaches as flow generating function \mathcal{F} , namely *FMLP*, *FGCN* and *DFF*. The first two extensions predict flows on edges based on topological encoding of participating nodes with the help of neural networks. The latter extension instead assigns flows on edges by solving a flow-balance equation (Equation 14). We now detail how these flows are derived.

FMLP and FGCN: As discussed in Section 3.1, for both variants, we first generate 16-dim node features $\mathcal{M} \in \mathbb{R}^{|V| \times 16}$ using a standard spectral embedding based on the Laplacian eigenmaps of the input graph’s adjacency matrix A . This step allows our two-step variants to handle variable sized graphs as the baseline models are likely to generate graphs with varying number of nodes and edges. Next, we experiment with two different encoder layers: (1) The *FMLP* variant consists of two-layer fully connected neural network, (2) the *FGCN* variant consists of two graph convolution layers [31]. Both variants apply ReLU activations after each layer. The hidden units are tuned $\in \{8, 16, 32\}$ and mini-batch size is tuned $\in \{32, 64, 128\}$. The loss objective is formally defined as follows:

$$\mathcal{L} = \sum_{A, X \sim P_{data}} \sqrt{\frac{1}{|A_{>0}|} \sum_{(u,v) \in A_{>0}} (\mathcal{F}(A)_{u,v} - X_{u,v})^2} \quad (13)$$

where A and X respectively denote the adjacency and flow matrices. $A_{>0} = \{(u,v) | A_{u,v} > 0\}$ represents the non-zero elements of A , while $X_{u,v} \in \mathbb{R}$ is the corresponding flow value directed from node u to v . The parameters of \mathcal{F} are jointly learned by minimizing \mathcal{L} using Adam optimizer with a learning rate of $1e-3$. We use the same train/validation/test split across all graph generative models for a fair comparison. Early stopping is applied based on the MSE score computed on the validation set with a patience of 10 epochs.

Divergence-free flows (DFF): The third variant aims to distribute flow values on the graph’s edges so that flows on majority of the nodes are nearly conserved, *i.e.*, total flow that enters a node should be approximately equal to the total flow that leaves. This is a well-studied phenomenon observed in many real-world settings including transportation networks, water supply networks and power grid networks. For this variant, for each graph $G(V, E)$ generated by undirected graph generative model (*e.g.* GRAN), we

first randomly pick 20% of the edges (E^L) and assign them flow values sampled from the ground truth distribution ($X_{u,v} \sim P_{data}$). Next, ensuring that flows are almost conserved, we compute flow values for the rest of the edges by solving the following constrained optimization problem introduced by [25]:¹

$$\begin{aligned} X^* &= \arg \min_X \|(X - X^T)\mathbf{1}\|_2^2 + \lambda \|X\|_F^2 \\ \text{s.t. } X_{u,v} &= X_{u,v}, \forall (u,v) \in E^L \text{ and } X_{u,v} = 0, \forall (u,v) \notin E^L \end{aligned} \quad (14)$$

where $\mathbf{1}$ is a column vector of size $|V|$ whose entries are all 1’s. λ is a constant that controls the regularization term—which guarantees a unique optimal solution—and is set to 0.1. Note that X^* may have negative flows after solving the above objective. Therefore, we further apply the following transformation in order to ensure the non-negative flow matrix: $X = \max(0, X^* - X^{*T})$.

6.5 Stopping Criterion

Notice that all the models employed in our experiments have different underlying objectives. Consequently, an early stopping criterion based on a non-decreasing loss is not appropriate and is likely to result in unfair comparisons. So, we apply an early stopping criterion that is based on the model performance measured with respect to evaluation metrics. Since we employ a collection of metrics that measure the performance from various perspectives (recall from Section 4.1), considering only a single metric (*e.g.* degree distribution) might not translate to similar performance in other metrics (*e.g.* divergence distribution) due to the complex nature of flow graphs. Therefore, we continue training the models—evaluating after each epoch—until more than half of the metrics stop improving with a patience of 10 epochs. For the baseline models, we re-generate the flows before each evaluation step as the generated graphs are likely to change. Once the training is complete, we rank all the snapshots based on the evaluation metrics and report the best-ranked snapshot for each model.

6.6 Efficiency comparison

Due to our rather complicated early stopping criterion, it is not reasonable to compare the total training times of models. Especially, since baseline models are not designed to generate flow graphs, they tend to have a fluctuating performance on flow-related metrics, which often results in longer training times. Instead, we report one full pass (single iteration) time for each model on *Taxi Trips 8-9am* in Table 5. Times are measured with batch size of 32 on a GeForce GTX 1070. As evident, NetGAN and FlowGEN have similar running times, while GRAN is significantly slower due to its sequential generation process.

| NetGAN | GRAN | FlowGEN |
|--------|-------|---------|
| 0.176 | 1.293 | 0.154 |

Table 5: Running time comparisons of the models (in seconds) on the Taxi Trip dataset.

¹Notation is modified from the original version to be consistent with our setting.