ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240845

IFH: A Diffusion Framework for Flexible Design of Graph Generative Models

Samuel Cognolato[a,b,*], **Alessandro Sperduti**[a,b] **and Luciano Serafini**[b]

[a]University of Padova, Padova, Italy
[b]Fondazione Bruno Kessler, Trento, Italy

Abstract. Graph generative models can be classified into two prominent families: one-shot models, which generate a graph in one go, and sequential models, which generate a graph by successive additions of nodes and edges. Ideally, between these two extreme models lies a continuous range of models that adopt different levels of sequentiality. This paper proposes a graph generative model, called Insert-Fill-Halt (IFH), that supports the specification of a sequentiality degree. IFH is based upon the theory of Denoising Diffusion Probabilistic Models (DDPM), designing a node removal process that gradually destroys a graph. An insertion process learns to reverse this removal process by inserting arcs and nodes according to the specified sequentiality degree. We evaluate the performance of IFH in terms of quality, run time, and memory, depending on different sequentiality degrees. We also show that using DiGress, a diffusion-based one-shot model, as a generative step in IFH leads to improvement to the model itself, and is competitive with the current state-of-the-art.

1 Introduction

Graph generative models usually fall into two categories: one-shot models, which generate the entire graph in one go, and sequential models, which iteratively extend the graph with new nodes and edges. State-of-the-art one-shot models are built upon well-established generative frameworks such as Variational Autoencoders (VAE) [29], Normalizing Flows [35], and Diffusion Models [32]. The same techniques have been applied for developing sequential models [18, 20, 16]. This additional inductive bias may spark the idea that a better sample quality can be achieved due to regularities in the graphs [14]. However, this is not always the case, as one-shot models have entered the state-of-the-art on challenging datasets like ZINC250k [13], Ego [27], and many more. Still, they are not flexible on the size of generated graphs, which is usually sampled from the dataset empirical distribution of nodes, a pre-computed histogram of the graphs' sizes. This is not ideal in a conditional generation setup, where conditioning variables can influence the extent of the graph. In this sense, autoregressive sequential models are more flexible because they can also learn the distribution of the number of nodes. This begs the question of whether we can borrow the strength of powerful one-shot models, and enhance them with the mentioned flexibility.

We answer the question by showing that, between the one-shot and sequential models, lies a continuous range of models that generate a graph by iteratively extending it more than one node at a time (see

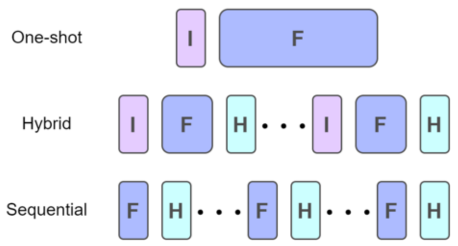

Figure 1. Graph generation can be seen as a sequence of Node Insertions (I), Labels and Connections Filling (F), and Halting Choices (H). One-shot models fill graphs in 1 big step after choosing the number of nodes. 1-node sequential models add one node, fill its value, connect it with the remainder graph, and choose whether to stop or continue iterating. Our IFH framework can model these situations and the intermediate block sequential generation.

Figure 1). Then, the one-shot model's most defining feature is not that of generating all nodes in one step, but the formulation of the nodes and edges sampling strategy, e.g., by sampling from the latent space with a VAE. In this paper, we propose a flexible graph framework obtained from the theory of diffusion models [11], called Insert-Fill-Halt (IFH), that allows us to specify the process of choosing how many and which nodes are added at each iteration. Once a formulation on the nodes and edges sampling is fixed, IFH can reproduce the behavior of the one-shot and sequential model while simultaneously allowing all possible intermediate sequentiality levels. Specifically, IFH identifies a node removal process and an insertion process. The former gradually deletes groups of nodes, while the latter tries to add them back, together with their labels and connections. The insertion process is carried out by (1) an Insertion Model, which decides how many new nodes to generate; (2) a Filler Model, which samples the new nodes' labels and connections; (3) a Halt Model, which chooses whether to halt the insertion process. The three components can be trained on their respective tasks on partial graphs, produced by the node removal process, starting from a clean graph.

Our framework provides a mathematical foundation that can be used both to adapt sampling strategies to any sequentiality level, and design new insertion schemes by customizing the removal process. On the former, we show that any current one-shot model can act as a Filler Model inside our IFH framework. On the latter, one can schedule how many nodes to add in a step and their ordering, depending on the dataset domain and size of graphs, taking into consideration time and memory constraints. We summarize our contributions as follows:

1. we propose Insert-Fill-Halt (IFH), a general graph generation framework based on Diffusion Models' theory [30, 11] that can

[*] Corresponding Author. Email: samuel.cognolato@phd.unipd.it

be specialized into many old and new graph generative models;

2. in the present work, we design two node-removal processes for IFH: naive and categorical, and use two node-orderings: random order and Breadth First Search (BFS) order. We evaluate their effectiveness in an ablation study on the QM9 molecular dataset;
3. we show empirically that adapting Digress [32], a state-of-the-art one-shot graph generative model, to 1-node sequential, leads to surpassing all autoregressive baselines, and is competitive with other state-of-the-art one-shot baselines such as CDGS [12];
4. we conduct a computational memory-time tradeoff survey when varying the degree of sequentiality, meaning going from one-shot to block-sequential, to 1-node sequential.

2 Notation and background

We introduce the notation we will use throughout the paper. For more details, we refer the reader to Appendix A.1 of the supplementary material [4]. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V} = \{v_1, .., v_n\}$ is the set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of directed arcs of $\mathcal{G}$. Let $n = |\mathcal{V}|$ and $m = |\mathcal{E}|$ be the number of nodes and edges of $\mathcal{G}$. $\mathcal{E}$ can be represented with the adjacency matrix $\boldsymbol{A} \in \{0,1\}^{n\times n}$ where $A_{i,j} = 1$ iff $(v_i, v_j) \in \mathcal{E}$. In the case of undirected graphs $\boldsymbol{A}$ is a symmetric matrix, i.e., $\boldsymbol{A} = \boldsymbol{A}^\top$. If $\mathcal{G}$ is labeled, then $\mathcal{V}$ and $\mathcal{E}$ are coupled with node features $\boldsymbol{X} \in \mathbb{R}^{n\times d_x}$ and edge features $\mathbf{E} \in \mathbb{R}^{n\times n\times d_e}$, where d_x and d_e are the dimensions of a single node/edge feature vector respectively. Global features are denoted as $\boldsymbol{y} \in \mathbb{R}^{d_y}$.

A node v can be *removed* from $\mathcal{G}$, meaning it is removed from $\mathcal{V}$, deleting all its connections and the labels attached to it. By removing a subset of nodes $\mathcal{V}_B \subseteq \mathcal{V}$ from a graph $\mathcal{G}$ we obtain the induced subgraph $\mathcal{G}_A$, with nodes $\mathcal{V}_A = \mathcal{V} \setminus \mathcal{V}_B$. The graph $\mathcal{G}$ can be *split* through the set of nodes $\mathcal{V}_A$ into the tuple $(\mathcal{G}_A, \mathcal{G}_B, \mathcal{E}_{AB}, \mathcal{E}_{BA})$, where $\mathcal{G}_A$ and $\mathcal{G}_B$ are the subgraphs induced by $\mathcal{V}_A$ and $\mathcal{V}_B$ respectively, and $\mathcal{E}_{AB}, \mathcal{E}_{BA}$ are the edges connecting them in one direction and the other. The inverse operation is a *merge*, which merges the two induced subgraphs and the edges back into $\mathcal{G}$. One of the main mathematical objects for this paper is the *forward graph removal sequence* $\mathcal{G}^{\rightarrow}_{0:T} = (\mathcal{G}_t)_{t=0}^T$ for graph $\mathcal{G}$, which is any sequence such that $\mathcal{G}_0 = \mathcal{G}$, $\mathcal{G}_T$ is the empty graph $\varnothing$, and $\mathcal{G}_t$ is an induced subgraph of $\mathcal{G}_{t-1}$ for all $t = 1, \ldots, T$. By reversing the order of $\mathcal{G}^{\rightarrow}_{0:T}$ we obtain the *reversed graph removal sequence* $\mathcal{G}^{\leftarrow}_{0:T}$, which will be useful to define the generative process from $\mathcal{G}_0 = \varnothing$ to $\mathcal{G}_T = \mathcal{G}$ as the resulting graph. We denote $\mathcal{F}(\mathcal{G}, T)$ and $\mathcal{R}(\mathcal{G}, T)$ as the sets of all forward and reversed removal sequences of $\mathcal{G}$ of length T, respectively.

Additionally, we define the halting process, borrowing the notation from [2]. Λ_t is a Markov chain with two states: *continue*, *halt*. The chain starts in the *continue* state, and proceeds at each step t with probability $\lambda(t)$ of being absorbed into the *halt* state. Once there, the process is stuck forever in the *halt* state.

3 Related works

3.1 Graph generation

Given a set of graph data points with unknown distribution $p_{\text{data}}(\mathcal{G})$, likelihood maximization methods aim to learn the parameters θ of a model $p_\theta(\mathcal{G})$ to approximate the true distribution $p_{\text{data}}(\mathcal{G})$. In the context of deep graph generation [9], $p_\theta(\mathcal{G})$ has been modeled as one-shot and sequential models.

One-shot models One-shot models employ a decoder network that maps a latent vector z to the resulting graph $\mathcal{G}$. The latent vector is usually sampled from a tractable distribution (such as a Normal distribution), and the number of nodes is either fixed, sampled from the frequencies of nodes in the dataset, or predicted from the latent code z. In general, one-shot models have the form:

$$p_{\theta,\phi}(\mathcal{G}) = p_\theta(\mathcal{G}|n)p_\phi(n). \tag{1}$$

When $p_\theta(\mathcal{G}|n)$ is implemented by a neural network architecture equivariant to node permutations, no node orderings are needed.

For one-shot generation, the classic generative paradigms are applied: VAE with GraphVAE [29], GAN with MolGAN [6] and SPECTRE [21], Normalizing Flows with MoFlow [35], diffusion with EDP-GNN [23], discrete diffusion with DiGress [32], energy-based models with GraphEBM [19], Stochastic Differential Equations (SDE) with GDSS [15] and CDGS [12].

Sequential models Sequential models frame graph generation as forming a sequence $\mathcal{G}^{\leftarrow}_{0:T} = (\mathcal{G}_0, \ldots, \mathcal{G}_t, \ldots, \mathcal{G}_T)$ of increasingly bigger graphs, where $\mathcal{G}_0$ is usually an empty graph, $\mathcal{G}_T$ is the generation result, and the transition from $\mathcal{G}_{t-1}$ to $\mathcal{G}_t$ introduces new nodes and edges. In the case of node-sequence generation, transitions always append exactly one node and the edges from that node to $\mathcal{G}_{t-1}$. In motif-sequence generation, blocks of nodes are inserted, together with new rows of the adjacency matrix. For the remainder of the paper, we will denote as *1-node sequential* the models based on a node-sequence generation, *block-sequential* for motif-based models, and *autoregressive models* for addressing both. Given a halting criteria $\lambda_\nu(\mathcal{G}_t, t)$ (see Section 2) based on current graph $\mathcal{G}_t$, the model distribution for a 1-node sequential model is of the form:

$$\begin{aligned} p_{\theta,\nu}(\mathcal{G}) = \sum_{\mathcal{G}^{\leftarrow}_{0:n} \in \mathcal{R}(\mathcal{G},n)} & \lambda_\nu(\mathcal{G}_n, n) p_\theta(\mathcal{G}_n|\mathcal{G}_{n-1}) \\ & \cdot p_\theta(\mathcal{G}_0) \prod_{t=1}^{n-1} (1 - \lambda_\nu(\mathcal{G}_t, t)) p_\theta(\mathcal{G}_t|\mathcal{G}_{t-1}). \end{aligned} \tag{2}$$

An essential ingredient in training autoregressive models is the node ordering, i.e., assigning a permutation π to the n nodes in $\mathcal{G}$ to build the sequence $\mathcal{G}^{\leftarrow}_{0:T}$. With random ordering, the model has to explore $n!$ permutations. In contrast, canonical orderings such as Breadth First Search (BFS) [34], Depth First Search (DFS), and many others [18, 3], decrease the size of the search space by introducing inductive biases, empirically increasing sample quality in most instances.

3.2 Diffusion models

We briefly introduce the Diffusion Model [30, 11], on which we build IFH. Let $\mathbf{x}_0$ be a data point sampled from an unknown distribution $q(\mathbf{x}_0)$. Denoising diffusion models are latent variable models with two components: (1) a diffusion process gradually corrupts $\mathbf{x}_0$ in T steps with Markov transitions $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ until $\mathbf{x}_T$ has some tractable distribution $p_\theta(\mathbf{x}_T)$; (2) a learned reverse Markov process with transition $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ denoises $\mathbf{x}_T$ back to the original data distribution $q(\mathbf{x}_0)$. The trajectories formed by the two processes are:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1}), \qquad \text{Forward} \tag{3}$$

$$p_\theta(\mathbf{x}_{0:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \qquad \text{Reverse} \tag{4}$$

For $T \to +\infty$, the forward and reverse transitions share the same functional form [7], and choosing $q(\mathbf{x}_T|\mathbf{x}_0) = q(\mathbf{x}_T)$ allows in fact to easily sample $\mathbf{x}_T$. The first successful attempt with diffusion models defined the transitions as $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \boldsymbol{I})$ [11] where β_t is a variance schedule. Later, diffusion models were adapted for discrete spaces [1], introducing concepts like uniform transitions, used in DiGress [32] with node and edge labels, and absorbing states diffusion, adopted in GraphARM [16] for masking nodes.

The distribution $p_\theta(\mathbf{x}_0)$ can be made to fit the data distribution $q(\mathbf{x}_0)$ by minimizing the variational upper bound:

$$L_{\text{vub}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0)} \Bigg[\underbrace{D_{\text{KL}}\big(q(\mathbf{x}_T|\mathbf{x}_0) \| p(\mathbf{x}_T)\big)}_{L_T} + \\ + \sum_{t=2}^{T} \underbrace{D_{\text{KL}}\big(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\big)}_{L_{t-1}} + \\ \underbrace{-\mathbb{E}_{\mathbf{x}_1 \sim q(\mathbf{x}_1|\mathbf{x}_0)} \left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right]}_{L_0} \Bigg]. \quad (5)$$

A necessary property to make diffusion models feasible to train is for $q(\mathbf{x}_t|\mathbf{x}_0)$ and $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ to have a closed form formula, to respectively (1) efficiently sample many time steps in parallel and (2) compute the KL divergences.

4 Removing nodes as a graph noise process

In this work, we frame the process of removing nodes from a graph $\mathcal{G}$ as an absorbing state diffusion process [1], gradually corrupting $\mathcal{G}$ until it collapses to the empty graph $\varnothing$. Differently from the absorbing diffusion of GraphARM [16], we do not limit the process to the choice of one node per step, but we include both the node ordering and number of nodes removed in the transitions.

Formally, given a graph data point $\mathcal{G}$, the removal process generates a removal sequence $\mathcal{G}_{0:T}^{\rightarrow}$ with $\mathcal{G}_0 = \mathcal{G}$ and $\mathcal{G}_T$ being the empty graph. We define the Markov removal transition $q(\mathcal{G}_t|\mathcal{G}_{t-1})$ as the probability of sampling a set of nodes $\mathcal{V}_t \subseteq \mathcal{V}_{t-1}$, and computing the induced subgraph $\mathcal{G}_t$ from $\mathcal{G}_{t-1}$ by $\mathcal{V}_t$. Following from eq. (4), the forward process is defined as:

$$q(\mathcal{G}_{1:T}^{\rightarrow}|\mathcal{G}_0) = \prod_{t=1}^{T} q(\mathcal{G}_t|\mathcal{G}_{t-1}). \quad (6)$$

Now we show the key insight that, because the number of nodes $n_t = |\mathcal{V}_t|$ is a known property of $\mathcal{G}_t$, the removal transition can be factorized into two components:

$$q(\mathcal{G}_t|\mathcal{G}_{t-1}) = q(\mathcal{G}_t, n_t|\mathcal{G}_{t-1}) = q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1}) q(n_t|\mathcal{G}_{t-1}), \quad (7)$$

where $q(n_t|\mathcal{G}_{t-1})$ is the probability that $\mathcal{V}_t$ will have exactly n_t nodes, and, fixed this number, $q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1})$ is the probability of choosing the nodes in $\mathcal{V}_t$ from $\mathcal{V}_{t-1}$. In simpler words, $q(n_t|\mathcal{G}_{t-1})$ tells *how many* nodes to keep alive, and once this fact is known, $q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1})$ select *which* nodes. In some special cases of the removal process, we will show that the number of nodes n_{t-1} is enough information to sample n_t, i.e., $q(n_t|\mathcal{G}_{t-1}) = q(n_t|n_{t-1})$.

4.1 *Parameterizing the reverse of the removal process*

Again, following the theory of diffusion models (Section 3.2), we introduce the insertion process, which learns to regenerate the graphs corrupted by the removal process. Define $p_{\theta,\phi}(\mathcal{G}_{t-1}|\mathcal{G}_t)$ as the Markov insertion transition which, given a partial graph $\mathcal{G}_t = \mathcal{G}_{t,A}$, samples a new subgraph $\mathcal{G}_{t,B} = (\mathcal{V}_{t,B}, \mathcal{E}_{t,B})$ with $r_t = n_{t-1} - n_t$, together with edges $\mathcal{E}_{t,AB}, \mathcal{E}_{t,BA}$ to connect the two graphs. Then, through a merge operation (as explained in Section 2), graph $\mathcal{G}_{t-1}$ is composed. The process reversing eq. (6) is defined as:

$$p_{\theta,\phi}(\mathcal{G}_{0:T}^{\rightarrow}) = \prod_{t=1}^{T} p_{\theta,\phi}(\mathcal{G}_{t-1}|\mathcal{G}_t), \quad (8)$$

where we omitted the $p_{\theta,\phi}(\mathcal{G}_T)$ term, as all the probability mass is already placed on the empty graph $\varnothing$. Again, we can factorize the transition into two components

$$p_{\theta,\phi}(\mathcal{G}_{t-1}|\mathcal{G}_t) = p_{\theta,\phi}(\mathcal{G}_{t-1}, r_t|\mathcal{G}_t) = p_\theta(\mathcal{G}_{t-1}|r_t, \mathcal{G}_t) p_\phi(r_t|\mathcal{G}_t), \quad (9)$$

where we call $p_\phi(r_t|\mathcal{G}_t)$ the *insertion model*, with parameters ϕ, and $p_\theta(\mathcal{G}_{t-1}|r_t, \mathcal{G}_t)$ the *filler model*, with parameters θ. The role of each is respectively: (1) given a partial subgraph $\mathcal{G}_t$ decide how many nodes r_t to add, (2) known this number and $\mathcal{G}_t$, generate the content of the new nodes and respective edges, and how to connect them to $\mathcal{G}_t$. Expanding $p_{\theta,\phi}(\mathcal{G}_{t-1}, r_t|\mathcal{G}_t)$, we have:

$$p_{\theta,\phi}(\mathcal{G}_{t,A}, \mathcal{G}_{t,B}, \mathcal{E}_{t,AB}, \mathcal{E}_{t,BA}, r_t|\mathcal{G}_{t,A}) = \\ = p_\theta(\mathcal{G}_{t,B}, \mathcal{E}_{t,AB}, \mathcal{E}_{t,BA}|r_t, \mathcal{G}_{t,A}) p_\phi(r_t|\mathcal{G}_{t,A}) = \\ = p_\theta(\mathcal{W}_t|r_t, \mathcal{G}_t) p_\phi(r_t|\mathcal{G}_t), \quad (10)$$

where we packed the tuple $\mathcal{W}_t = (\mathcal{G}_{t,B}, \mathcal{E}_{t,AB}, \mathcal{E}_{t,BA})$ for brevity. In Appendix A.4 [4], we show that, similarly to eq. (5), when $q(r_t|\mathcal{G}_t, \mathcal{G}_0)$ and $q(\mathcal{W}_t|r_t, \mathcal{G}_t, \mathcal{G}_0)$ can be expressed in closed form they can be estimated by minimizing the variational upper bound:

$$L_{\text{vub}} = \mathbb{E}_{\mathcal{G}_0 \sim q(\mathcal{G}_0)} \Bigg[\sum_{t=2}^{T} D_{\text{KL}}\big(q(r_t|\mathcal{G}_t, \mathcal{G}_0) \| p_\phi(r_t|\mathcal{G}_t)\big) + \\ - \mathbb{E}_{\mathcal{G}_1 \sim q(\mathcal{G}_1|\mathcal{G}_0)} \left[\log p_\phi(r_1|\mathcal{G}_1)\right] + \\ + \sum_{t=2}^{T} D_{\text{KL}}\big(q(\mathcal{W}_t|r_t, \mathcal{G}_t, \mathcal{G}_0) \| p_\theta(\mathcal{W}_t|r_t, \mathcal{G}_t)\big) + \\ - \mathbb{E}_{\mathcal{G}_1 \sim q(\mathcal{G}_1|\mathcal{G}_0)} \left[\log p_\theta(\mathcal{W}_1|r_1, \mathcal{G}_1)\right] \Bigg]. \quad (11)$$

The KL divergence of the filler model term can be replaced by the negative log-likelihood, at the price of increasing the upper bound. On the other hand, this allows to train $p_\theta(\mathcal{W}_t|r_t, \mathcal{G}_t)$ through any likelihood maximization method, such as VAE, Normalizing Flow, and Diffusion. In particular, noticing the resemblance with eq. 1, the filler model can be any likelihood-based one-shot model, although adapted to be conditioned on an external graph $\mathcal{G}_t$, and able to generate the interconnections, which we explain in Section 5.4.

4.2 *Choosing the removal process*

The design of $q(\mathcal{G}_t|\mathcal{G}_{t-1})$ influences the graph generative model $p_{\theta,\phi}(\mathcal{G}_{t-1}|\mathcal{G}_t)$ in several ways. We start by proposing a naive *coin flip* approach for removing nodes, moving afterward to a more effective way of choosing the number of nodes to remove and how to incorporate node ordering. Definitions and proofs can respectively be found in Appendix A.3 and A.4 of the supplementary material [4].

Naive/binomial The simplest way to remove nodes from a graph $\mathcal{G}_{t-1}$ is to assign a Bernoulli random variable with probability q_t to each node. All nodes with a positive outcome are removed. It follows

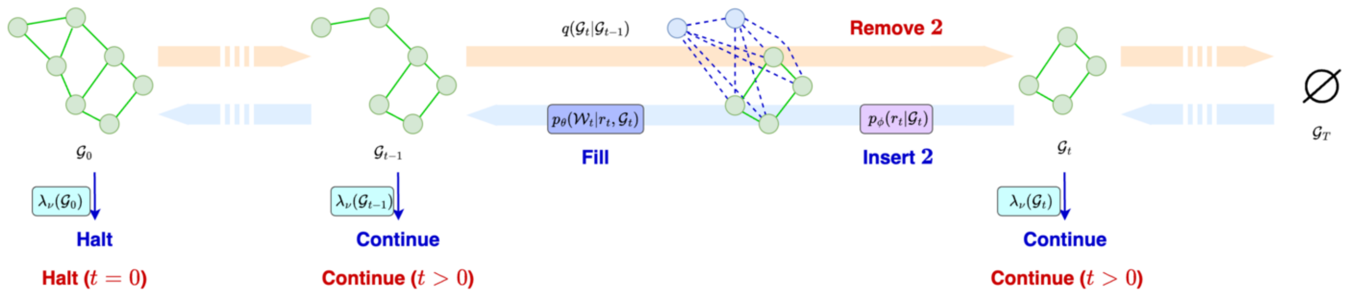
$q(\mathcal{G}_t|\mathcal{G}_{t-1})$
Remove 2
$p_\theta(\mathcal{W}_t|r_t,\mathcal{G}_t)$
Fill
$p_\phi(r_t|\mathcal{G}_t)$
Insert 2
$\varnothing$
$\mathcal{G}_0$
$\mathcal{G}_{t-1}$
$\mathcal{G}_t$
$\mathcal{G}_T$
$\lambda_\nu(\mathcal{G}_0)$
$\lambda_\nu(\mathcal{G}_{t-1})$
$\lambda_\nu(\mathcal{G}_t)$
Halt
Continue
Continue
Halt ($t=0$)
Continue ($t>0$)
Continue ($t>0$)

Figure 2. Our Insert-Fill-Halt model. During training, a graph is corrupted (left to right) by iteratively removing nodes until the empty graph $\varnothing$ is left. At each step, the insertion (violet), filler (blue), and halt (cyan) models have to predict how many nodes were removed, what content they had, and whether the graph is terminal, respectively (right to left).

that, given n_{t-1} nodes at step $t-1$, the count of survived nodes n_t is distributed as a Binomial $B(n_t; n_{t-1}, 1-q_t)$. Iterating this process for t steps from graph $\mathcal{G}_0$, we still get that $n_t|n_0$ is distributed as a Binomial, where the probability of being alive at step t is the product of being alive at all steps. Finally, the posterior $q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1})$, needed for computing the loss, is again distributed as a Binomial on the removed nodes $\Delta n_t = n_0 - n_t$.

Categorical One drawback of the binomial removal is that, in principle, any block size can be sampled. This can be a problem when batching multiple examples (see Section 5.5), and leads to high variability in block sizes during training. To control the size of blocks and limit the options available to the model, we developed a categorical removal process where the Insertion Model can choose from a predefined set of options. We based our formulation on the change-making problem [33], interpreting the number of nodes as the amount to be made using a set of coin denominations $D = \{d_1, .., d_c\}$. Then, a removal transition is defined as a categorical distribution over D, where each denomination's probability is its normalized count to reach n with the lowest number of coins. We find that categorical removals also admit a closed form for $n_t|n_0$, distributed as a multivariate hypergeometric, and Δn_t, distributed as a mirrored version of n_t.

Node ordering Until now, we assumed nodes were removed uniformly in all possible permutations. This doesn't need to be the case, as the whole removal process can be conditioned on a particular node ordering π. The transitions will then obey $q(\mathcal{G}_t|\mathcal{G}_{t-1}, \pi) = q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1}, \pi)q(n_t|\mathcal{G}_{t-1}, \pi) = q(n_t|\mathcal{G}_{t-1}, \pi)$.

Halting process Due to the arbitrary size of graphs, one needs to know when to stop sampling. One possibility is to stop after a fixed number of steps T, or when some property of the removal process is met (e.g., in the binomial process). Learning a halting process (see Section 2) can be a solution when this is not possible. A halting model $\lambda_\nu(\mathcal{G}_t, t)$ can be trained in a binary classification setup by setting the halting ground truth signal to 1 for the actual data graph $\mathcal{G}$, and 0 for all its induced subgraphs.

5 Uncovering the spectrum of sequentiality

One-shot and sequential graph generative models (Section 3.1) are seen as two different families of graph generative models. Here we show that these are actually the two extremes of a spectrum, captured by our Insert-Fill-Halt (IFH) framework (Figure 2). First of all, let's consider the reversed removal sequence (see Section 2) $\mathcal{G}^{\leftarrow}_{0:T} = (\mathcal{G}_t)_{t=0}^{T} = (\varnothing, \dots, \mathcal{G})$. The three modules are: (1) a *Node Insertion Module* $p_\phi(r_{t-1}|\mathcal{G}_{t-1})$, deciding how many nodes are going to be inserted in $\mathcal{G}_{t-1}$; (2) a *Filler Module* $p_\theta(\mathcal{W}_{t-1}|r_{t-1}, \mathcal{G}_{t-1})$, filling the new r_{t-1} nodes and edges $\mathcal{W}_{t-1}$, merging them with the existing graph $\mathcal{G}_{t-1}$ to get $\mathcal{G}_t$; (3) a *Halting Module* $\lambda_\nu(\mathcal{G}_t)$, determining, through some halting criteria, whether to stop the generative process at t or to continue. The overall model distribution is:

$$p_{\theta,\phi,\nu}(\mathcal{G}) = \sum_{T=1}^{\infty} \sum_{\mathcal{G}^{\leftarrow}_{0:T} \in \mathcal{R}(\mathcal{G},T)} p_\theta(\mathcal{G}_0)$$
$$\underbrace{\lambda_\nu(G_T)}_{\text{halt at last step}}\ p_\theta(\mathcal{W}_{T-1}|r_{T-1}, \mathcal{G}_{T-1})p_\phi(r_{T-1}|\mathcal{G}_{T-1})$$
$$\prod_{t=1}^{T-1} \underbrace{(1-\lambda_\nu(\mathcal{G}_t))}_{\text{do not halt}} \underbrace{p_\theta(\mathcal{W}_{t-1}|r_{t-1}, \mathcal{G}_{t-1})}_{\text{fill}} \underbrace{p_\phi(r_{t-1}|\mathcal{G}_{t-1})}_{\text{insert}}. \quad (12)$$

5.1 *Specializing to one-shot and sequential models*

One-shot One-shot models (eq. (1)) are 1-step instances of our IFH model with the insertion module set to be a sampler of the total number of nodes, i.e., $p_\phi(r_0|\varnothing) = p_\phi(n_1) = p_\phi(n)$. The filler model is the actual one-shot model, sampling all nodes in one go. The halting model always stops after 1 step.

Sequential Sequential models (eq. (2)) are n-step instances of our IFH model, with the insertion module always choosing 1 as the nodes to insert. The filler model samples a new node and links it with graph $\mathcal{G}_{t-1}$ to compose $\mathcal{G}_t$. The halting model is dependent on the architecture: in [34], an End-Of-Sequence (EOS) token is sampled to end generation; in [28] it is not clear, but we assume they fix the number of nodes at the start; in [20] generation stops when a limit on n is reached, or if the model does not link the newly generated node to the previous subgraph; [10] trains a neural network to predict the halting signal from the adjacency matrix.

Differences from GRAN GRAN [18] is a block-sequential model that upon a superficial analysis could be considered similar to our IFH. A key difference with IFH is that GRAN actually generates blocks until reaching a maximum number of nodes. This number is fixed, computed as the biggest multiple of the block size nearest to the maximum number of nodes of the dataset. Then, the actual number of nodes n is sampled from the empirical distribution of nodes, and the subgraph with the first n nodes is extracted. In a sense, GRAN acts as a one-shot model regarding the number of nodes, although nodes are filled in an autoregressive way. This means that GRAN does not learn the nodes distribution, and does not adapt the number of generation steps to size, implying that many sequential models cannot be built from GRAN.

5.2 Training

Given an example graph $\mathcal{G}$, training is performed over the entire removal sequence $\mathcal{G}_{\overrightarrow{0:T}}$. The models parameters ϕ, θ, ν can be optimized by minimizing the loss function:

$$\mathcal{L}(\phi, \theta, \nu) = L_{\text{vub}}(\phi, \theta) + L_{\text{halt}}(\nu), \tag{13}$$

with $L_{\text{vub}}(\phi, \theta)$ defined as in eq. (11), and the halting loss $L_{\text{halt}}(\nu)$ is an optional binary classification loss as described in Section 4.2. The full training algorithm is provided in Algorithm 1. Notice that, apart from the removal sequence sampling, every iteration of the while loop can be computed in parallel on GPU, by batching all the obtained graphs.

Algorithm 1 Training

```
 1: repeat
 2:     $\mathcal{G}_0 \sim q(\mathcal{G})$
 3:     while $\mathcal{G}_{t-1} \neq \varnothing$ do
 4:         $\mathcal{G}_t \sim q(\mathcal{G}_t|\mathcal{G}_{t-1})$                          ▷ remove nodes
 5:         $r_t \leftarrow n_{t-1} - n_t$                             ▷ get true number of nodes
 6:         $\mathcal{W}_t \leftarrow \text{split}(\mathcal{G}_{t-1}, \mathcal{G}_t)$                ▷ get true nodes and edges
 7:         $h_t \leftarrow \delta(t-1)$                               ▷ get true halting signal
 8:         $L_{\text{ins},t}(\phi) \leftarrow D_{\text{KL}}\big(q(r_t|\mathcal{G}_t, \mathcal{G}_0)\|p_\phi(r_t|\mathcal{G}_t)\big)$
 9:         $L_{\text{fill},t}(\theta) \leftarrow D_{\text{KL}}\big(q(\mathcal{W}_t|r_t, \mathcal{G}_t, \mathcal{G}_0)\|p_\theta(\mathcal{W}_t|r_t, \mathcal{G}_t)\big)$
10:         $L_{\text{halt},t}(\nu) \leftarrow \mathcal{L}_{\text{halt}}(h_t, \lambda_\nu(\mathcal{G}_{t-1}))$
11:     end while
12:     Perform gradient descent step on
            $\frac{1}{T}\sum_{t=1}^{T}(L_{\text{ins},t}(\phi) + L_{\text{fill},t}(\theta) + L_{\text{halt},t}(\nu))$
13: until converged
```

5.3 Sampling

The sampling process is a sequence of Insert, Fill, Halt operations, which is terminated by a positive halting signaling (Algorithm 2).

Algorithm 2 Sampling

```
1: $\mathcal{G}_0 \leftarrow \varnothing$                                   ▷ start from the empty graph
2: repeat
3:     $r_t \sim p_\phi(r_t|\mathcal{G}_t)$                       ▷ sample how many nodes to add
4:     $\mathcal{W}_t \sim p_\theta(\mathcal{W}_t|r_t, \mathcal{G}_t)$                ▷ sample new nodes and edges
5:     $\mathcal{G}_{t+1} \leftarrow \text{merge}(\mathcal{G}_t, \mathcal{W}_t)$
6:     $h_t \sim \lambda_\nu(\mathcal{G}_{t+1})$                           ▷ sample halting signal
7: until $h_t = 1$
8: return $\mathcal{G}_T$
```

5.4 Adapting one-shot models to sequential

In Section 5.1, we showed how one-shot models are 1-step IFH models, and our parametrization in Section 4.1 allows the use of any one-shot model inside a multi-step instance. Usually, one-shot models operate by sampling n nodes and the $n \times n$ adjacency matrix. For this reason, they need to be adapted to generate the edges linking the new nodes with a previous subgraph, and to condition the former on the latter. Consider the already generated subgraph $\mathcal{G}_{t-1}$, and denote $\mathcal{W}_{t-1}$ as the new subgraph of size r_{t-1} and the inter-connections with $\mathcal{G}_{t-1}$ to be sampled. We propose the following adaptation to the T-step setup for undirected graphs: (1) encode the n_{t-1} nodes of graph $\mathcal{G}_{t-1}$ into vector representations through a Graph Neural Network such as GraphConv [22] or RGCN [25] for labeled data; (2) generate the new r_{t-1} nodes and a rectangular adjacency matrix with size $r_{t-1} \times n_s$ using the encoded node vectors, where $n_t = r_{t-1} + n_{t-1}$; (3) merge $\mathcal{G}_{t-1}$ and $\mathcal{W}_{t-1}$ into $\mathcal{G}_t$ by concatenating nodes and the adjacency matrix. Summarizing, the strategy entails adding r_{t-1} new rows to the previous adjacency matrix, without materializing it. We motivate this choice in the following section.

5.5 Complexity considerations

One-shot models generating adjacency matrices have a quadratic dependency on the number of nodes for both time and memory. However, they are very fast to train and sample from using parallelizable computing architectures such as GPUs. It is not the case for autoregressive models where, due to their iterative nature, they cannot fully benefit from parallelization [18]. Still, these do not need to generate the whole adjacency matrix in one go, and can more efficiently store the already-generated graph representation, e.g., converting to a sparse edge list (as implemented in Torch Geometric [8]). Another factor affecting memory and time is *batching*, that is, generating or training on many graphs simultaneously, stacking their features in tensors. For dense representations, like adjacency matrices, the size of the resulting batched tensor is always the biggest of the batch, and the rest are padded with zeroes. This implies that memory consumption depends on the maximum size of generated blocks, so one-shot models fall on the most expensive side. This is true both when training and sampling. Still, when parallelizing training of autoregressive models on all steps, the price is paid by replicating the same example many times, just with masked nodes. We show empirically in Section 6 that these considerations are confirmed in reality.

6 Experiments

We experimentally evaluate how changing the formulation of the removal process changes sample quality, time, and memory consumption. In GRAN [18] a sample quality/time trade-off analysis on a grid graphs dataset was already performed, changing the fixed block size, stride, and node ordering. We extend this analysis to many more molecular and generic graph datasets, evaluating different degrees of sequentiality, i.e., scheduled sizes of blocks.

To showcase our framework[1], we adapt DiGress [32] following the procedure described in Section 5.4. We focus on domain-agnostic learning. Our method can be applied as-is to any graph dataset, apart from one-shot variants needing the node frequencies (see Section 3.1). Thus, we use the base version of DiGress, without optimal prior and domain-specific features, replacing them with nodes in-degrees and the number of nodes. As halting and insertion models we use Relational Graph Convolutional Networks [25]. We finetuned the architecture hyperparameters through a Bayesian Search on each dataset. Then, we followed the approach of [12] and evaluated the sampling quality in several datasets. We use early stopping with validation losses to individually stop each module, as they could have different training times. Time and memory are evaluated using the same hyperparameters to avoid differences in model size. More details on experiments can be found in Appendix A.5 of the supplementary material [4].

[1] Our code can be found at https://github.com/CognacS/ifh-model-graphgen

Table 1. QM9 ablation study for binomial (bin) vs. categorical removal (cat), uniform (unif) vs. BFS ordering.

Method	Valid↑ (%)	Unique↑ (%)	Novel↑ (%)	NSPDK↓	FCD↓	Time (m)	Memory (GB)
bin unif	91.45	97.50	94.84	6.88e-4	1.310	61.85	1.96
bin BFS	92.72	96.34	93.01	0.001	1.175	63.00	1.96
cat unif	89.40	98.45	93.80	4.82e-4	1.171	28.61	0.83
cat BFS	92.30	97.70	91.81	4.78e-4	0.918	26.48	0.80

Table 2. Sequentiality levels on generic graphs, i.e., block sizes used.

Method	Ego-small	Enzymes	Ego	Comm-small
seq-1	1	1	1	1
seq-small	$\{1,2\}$	$\{1,3\}$	$\{1,3\}$	$\{1,2\}$
seq-big	$\{1,2,8\}$	$\{1,2,8\}$	$\{1,4,16\}$	$\{1,2,8\}$
one-shot	n	n	n	n

Molecular datasets We report results on two of the most popular molecular datasets: QM9 [24], and ZINC250k [13] with 133K and 250K molecules, respectively. As usual, we kekulize the molecules, i.e., remove the hydrogen atoms and replace aromatic bonds with single and double bonds, using the chemistry library RDKit [17]. To measure sample quality we compute the Fréchet ChemNet Distance (FCD) and Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) metrics. We also compute the ratio of Valid, Unique, and Novel molecules, allowing partial charges. For both datasets, we generate 10K molecules, and evaluate FCD and NSPDK on the respective test sets of QM9 and ZINC250k. We use 10% molecules from QM9 and ZINC250k training sets for validation, respectively.

Generic graphs datasets On generic graphs we evaluate our approach on: Community-small, with 100 graphs [34]; Ego-small and Ego with 200 and 757 graphs [27]; Enzymes with 563 protein graphs [26]. We split the train/validation/test sets with the 60/20/20 proportion. We strictly follow [12] and compute the Maximum Mean Discrepancy (MMD) with radial basis on the distribution of Degree, Clustering coefficient, Laplacian Spectrum coefficient (Spec.) and random GIN embeddings [31], which are a replacement of FCD for generic graphs. For Community-small and Ego-small we generate 1024 graphs, and for Ego and Enzymes we generate the same number of graphs as the test set.

Baselines For each dataset, we define 4 degrees of sequentiality of our model: 1-node, small blocks, big blocks, and one-shot. Details on their definition for generic graphs can be found in Table 2. On molecular datasets we compare versus most of the state-of-the-art models reported in [12]. Specifically, we consider the autoregressive models GraphAF [28], GraphDF [20], GraphARM [16], and the one-shot models MoFlow [35], EDP-GNN [23], GraphEBM [19], DiGress [32], GDSS [15], CDGS [12]. On generic graphs datasets, we compare IFH versus the autoregressive models GraphRNN [34], GRAN [18], and the one-shot models VGAE [29], EDP-GNN [23], GDSS [15], CDGS [12].

6.1 *Experimental results*

Ablation study We conducted a preliminary ablation study on QM9 (shown in Table 1) to evaluate the best-performing formulation for the removal process from those we proposed. For binomial removals, we used the adaptive linear scheduling explained in Appendix A.3.2, and for categorical removals we used $D = \{1, 4\}$ as block sizes, where 4 is approximately half the size of the biggest QM9's molecules. As predicted in Section 5.5, the models trained with binomial removals have a huge memory footprint and worse sampling time than categorical removal. When comparing sampling quality, the categorical removal process is still superior. On the ordering, BFS improves quality compared to the uniformly random order, confirming the results of [18].

Performance of the spectrum Table 3 shows that the fully sequential model achieves competitive results with CDGS, surpassing all autoregressive baselines on both QM9 and ZINC250k. Specifically, moving from sequential down shows a drop in general performance. Regarding generic graph generation, we also see competitive results with the state-of-the-art. Still, good performance can also be achieved through small and big block generation, for example, in Ego-small. We observed worse results regarding Community-small and Ego.

Time and memory consumption Looking at Tables 3c, 3f the level of sequentiality towards 1-node sequential always reduces the memory footprint during generation, as smaller and smaller adjacency matrices are generated, but time goes up, as predicted in Section 5.5. Due to the paper's page limit, we refer the reader to Appendix A.2 [4] to find the generic graphs' time/memory consumption tables. We highlight the case in Table 4b in the Appendix with the Enzymes and Ego datasets containing very large graphs. On these datasets, the sequential model uses respectively 1/50 and 1/88 of the memory of the one-shot model for generation, although with an increased computational time. During training (Tables 3b, 3e), for small graph datasets such as QM9, memory usage is higher in sequential models, differently from larger graph datasets like ZINC, where the cost of storing big adjacency matrices outweighs that of split sparse graphs.

7 Discussion

In Section 6, we showed that adapting our chosen one-shot model to sequential generation led to an improvement of the state-of-the-art for autoregressive generation and being competitive with the state-of-the-art model CDGS. At the same time, one can trade off generation time for memory and performance, although there seems to be a sweet spot inside the spectrum for larger graphs. This shows that the optimal removal process is dataset and task-dependent, and could be considered as a hyperparameter to be tuned when investigating new graph generative models. Our conjecture is that for smaller graph datasets, one-shot models are the fastest and best-performing solution, as seen with the results of CDGS, while as size increases, sequential models should be the go-to, particularly where memory is highly constrained. At huge scales, autoregressive techniques become the only feasible solution [5]. There is still room for improvement on this work's current limitations. For instance, designing better halting models is critical, as larger graphs imply sparser halting signals to train on. Additionally, we found that block-sequential models are susceptible to how information is routed from the previous graph. Then, finding better one-shot models adaptation schemas is crucial.

8 Conclusion

In this work, we proposed the IFH framework, which unifies the one-shot and autoregressive paradigms, leaving plenty of room for customization. We showed that high-quality, task-agnostic, autoregressive graph generative models are feasible by adapting DiGress to sequential. In the future, we would like to explore how to better mix the advantages of the two modalities, building upon our framework, gaining the one-shot time efficiency, better memory management, and improved sample quality.

Table 3. Results on the molecule generation task on QM9 (a-c), ZINC250k (d-f) and generic graphs (g) averaged over 3 runs after model selection. For molecular datasets, the tables on the left report performance results, while the tables on the right show the time/memory cost for different levels of sequentiality. On the comparison tables, the best results are in bold, and the second best are underlined.

(a) Performance results on the QM9 dataset

Method		Valid (%)↑	NSPDK↓	FCD↓	Unique (%)↑	Novel (%)↑
Metrics on Training Set		-	1.36e-4	0.057	-	-
Autoreg.	GraphAF	74.43	0.021	5.625	88.64	86.59
	GraphDF	93.88	0.064	10.928	98.58	98.54
	GraphARM	90.25	0.002	1.220	95.62	70.39
One-shot	MoFlow	91.36	0.017	4.467	98.65	94.72
	EDP-GNN	47.52	0.005	2.680	99.25	86.58
	GraphEBM	8.22	0.030	6.143	97.90	97.01
	DiGress	99.00	5e-4	0.360	96.66	33.40
	GDSS	95.72	0.003	2.900	98.46	86.27
	CDGS	99.68	3.08e-4	**0.200**	96.83	69.62
Ours	seq-1	**99.92**	**2.99e-4**	0.902	96.63	88.33
	$\{1, 2\}$	94.34	4.19e-4	0.904	97.08	89.11
	$\{1, 4\}$	92.51	7.53e-4	0.995	97.72	92.16
	one-shot	95.31	0.002	1.512	96.93	94.65

(b) Training time/memory QM9 dataset

Method	Time/epoch (m)	Memory (GB)
seq-1	3.9	6.52
$\{1, 2\}$	3.54	5.40
$\{1, 4\}$	3.36	6.05
one-shot	1.98	3.73

(c) Generation time/memory QM9 dataset

Method	Time (m)	Memory (GB)
seq-1	23.30	0.38
$\{1, 2\}$	20.55	0.48
$\{1, 4\}$	25.54	0.83
one-shot	16.92	1.22

(d) Performance results on the ZINC250K dataset

Method		Valid (%)↑	NSPDK↓	FCD↓	Unique (%)↑	Novel (%)↑
Metrics on Training Set		-	5.91e-5	0.985	-	-
Autoreg.	GraphAF	68.47	0.044	16.023	98.64	99.99
	GraphDF	90.61	0.177	33.546	99.63	100.00
	GraphARM	88.23	0.055	16.260	99.46	100.00
One-shot	MoFlow	63.11	0.046	20.931	99.99	100.00
	EDP-GNN	82.97	0.049	16.737	99.79	100.00
	GraphEBM	5.29	0.212	35.471	98.79	100.00
	DiGress	91.02	0.082	23.06	81.23	100.00
	GDSS	97.01	0.019	14.656	99.64	100.00
	CDGS	98.13	**7.03e-4**	**2.069**	99.99	99.99
Ours	seq-1	**98.56**	0.002	2.387	99.87	99.89
	$\{1, 3\}$	80.59	0.004	3.312	99.98	99.95
	$\{1, 4, 8\}$	65.68	0.015	9.229	99.94	100.00
	one-shot	60.48	0.033	15.174	100.00	100.00

(e) Training time/memory ZINK250K dataset

Method	Time/epoch (m)	Memory (GB)
seq-1	30.48	15.09
$\{1, 3\}$	20.64	16.53
$\{1, 4, 8\}$	20.88	15.37
one-shot	15.84	19.56

(f) Generation time/memory ZINK250K dataset

Method	Time (m)	Memory (GB)
seq-1	51.09	0.59
$\{1, 3\}$	26.71	1.08
$\{1, 4, 8\}$	36.39	3.05
one-shot	44.43	18.03

(g) Performance results on generic graphs datasets

		Community				Ego-small				Enzymes				Ego			
		$\lvert V\rvert_{\max} = 20,\ \lvert E\rvert_{\max} = 62$ $\lvert V\rvert_{\text{avg}} \approx 15,\ \lvert E\rvert_{\text{avg}} \approx 36$				$\lvert V\rvert_{\max} = 17,\ \lvert E\rvert_{\max} = 66$ $\lvert V\rvert_{\text{avg}} \approx 6,\ \lvert E\rvert_{\text{avg}} \approx 9$				$\lvert V\rvert_{\max} = 125,\ \lvert E\rvert_{\max} = 149$ $\lvert V\rvert_{\text{avg}} \approx 33,\ \lvert E\rvert_{\text{avg}} \approx 63$				$\lvert V\rvert_{\max} = 399,\ \lvert E\rvert_{\max} = 1071$ $\lvert V\rvert_{\text{avg}} \approx 145,\ \lvert E\rvert_{\text{avg}} \approx 335$			
Method		Deg.↓	Clus.↓	Spec.↓	GIN↓	Deg.↓	Clus.↓	Spec.↓	GIN↓	Deg.↓	Clus.↓	Spec.↓	GIN↓	Deg.↓	Clus.↓	Spec.↓	GIN↓
Metrics on Training Set		0.035	0.067	0.045	0.037	0.025	0.029	0.027	0.016	0.011	0.011	0.011	0.007	0.009	0.009	0.009	0.005
A-R	GraphRNN	0.106	0.115	0.091	0.353	0.155	0.229	0.167	0.472	0.397	0.302	0.260	1.495	0.140	0.755	0.316	1.283
	GRAN	0.125	0.164	0.111	0.196	0.096	0.072	0.095	0.106	0.215	0.147	0.034	0.069	0.594	0.425	1.025	0.244
O-S	VGAE	0.391	0.257	0.095	0.360	0.146	0.046	0.249	0.089	0.811	0.514	0.153	0.716	0.873	1.210	0.935	0.520
	EDP-GNN	0.100	0.140	0.085	0.125	0.026	0.032	0.037	0.031	0.120	0.644	0.070	0.119	0.553	0.605	0.374	0.295
	GDSS	0.102	0.125	0.087	0.137	0.041	0.036	0.041	0.041	0.118	0.071	0.053	0.028	0.314	0.776	0.097	0.156
	CDGS	**0.052**	**0.080**	**0.064**	**0.062**	**0.025**	**0.031**	0.033	**0.025**	**0.048**	0.070	0.033	**0.024**	**0.036**	**0.075**	**0.026**	**0.026**
Ours	seq-1	0.209	0.189	0.082	0.277	0.069	0.084	0.066	0.046	0.049	**0.049**	**0.026**	0.088	0.303	0.643	0.311	0.352
	seq-small	0.177	0.167	0.082	0.203	0.031	0.041	0.040	0.043	0.252	0.237	0.077	0.404	0.435	0.898	0.162	0.403
	seq-big	0.141	0.173	0.089	0.262	0.027	0.042	**0.029**	0.043	0.441	0.470	0.196	0.698	0.276	0.992	0.190	0.479
	oneshot	0.125	0.187	0.081	0.138	0.045	0.065	0.048	0.048	0.264	0.436	0.050	0.180	0.372	0.695	0.458	0.528

Acknowledgements

We acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.

References

[1] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

[2] A. Banino, J. Balaguer, and C. Blundell. Pondernet: Learning to ponder. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021.

[3] X. Chen, X. Han, J. Hu, F. Ruiz, and L. Liu. Order matters: Probabilistic modeling of node sequence for graph generation. In *International Conference on Machine Learning*, pages 1630–1639. PMLR, 2021.

[4] S. Cognolato, A. Sperduti, and L. Serafini. Ifh: a diffusion framework for flexible design of graph generative models, 2024. URL https://arxiv.org/abs/2408.13194. Full version of this paper.

[5] H. Dai, A. Nazi, Y. Li, B. Dai, and D. Schuurmans. Scalable deep generative modeling for sparse graphs. In *International Conference on Machine Learning*, pages 2302–2312. PMLR, 2020.

[6] N. De Cao and T. Kipf. Molgan: An implicit generative model for small molecular graphs. *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

[7] W. Feller. On the Theory of Stochastic Processes, with Particular Reference to Applications. In *First Berkeley Symposium on Mathematical Statistics and Probability*, pages 403–432, Jan. 1949.

[8] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[9] X. Guo and L. Zhao. A systematic survey on deep generative models for graph generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5370–5390, 2022.

[10] X. Han, X. Chen, F. J. Ruiz, and L.-P. Liu. Fitting autoregressive graph generative models through maximum likelihood estimation. *Journal of Machine Learning Research*, 24(97):1–30, 2023.

[11] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[12] H. Huang, L. Sun, B. Du, and W. Lv. Conditional diffusion based on discrete graph structures for molecular graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4302–4311, 2023.

[13] J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad, and R. G. Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

[14] W. Jin, R. Barzilay, and T. Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, pages 2323–2332. PMLR, 2018.

[15] J. Jo, S. Lee, and S. J. Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International Conference on Machine Learning*, pages 10362–10383. PMLR, 2022.

[16] L. Kong, J. Cui, H. Sun, Y. Zhuang, B. A. Prakash, and C. Zhang. Autoregressive diffusion model for graph generation. In *International Conference on Machine Learning*, pages 17391–17408. PMLR, 2023.

[17] G. Landrum, P. Tosco, B. Kelley, Ric, sriniker, gedeck, R. Vianello, NadineSchneider, D. Cosgrove, E. Kawashima, A. Dalke, D. N, G. Jones, B. Cole, M. Swain, S. Turk, AlexanderSavelyev, A. Vaucher, M. Wójcikowski, I. Take, D. Probst, K. Ujihara, V. F. Scalfani, guillaume godin, A. Pahl, F. Berenger, JLVarjo, strets123, JP, and DoliathGavid. RDKit: Open-source cheminformatics. http://www.rdkit.org.

[18] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32, 2019.

[19] M. Liu, K. Yan, B. Oztekin, and S. Ji. Graphebm: Molecular graph generation with energy-based models. *arXiv preprint arXiv:2102.00546*, 2021.

[20] Y. Luo, K. Yan, and S. Ji. Graphdf: A discrete flow model for molecular graph generation. In *International Conference on Machine Learning*, pages 7192–7203. PMLR, 2021.

[21] K. Martinkus, A. Loukas, N. Perraudin, and R. Wattenhofer. Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators. In *International Conference on Machine Learning*, pages 15159–15179. PMLR, 2022.

[22] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

[23] C. Niu, Y. Song, J. Song, S. Zhao, A. Grover, and S. Ermon. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics*, pages 4474–4484. PMLR, 2020.

[24] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[25] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pages 593–607. Springer, 2018.

[26] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.

[27] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3): 93–93, 2008.

[28] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2020.

[29] M. Simonovsky and N. Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*, pages 412–422. Springer, 2018.

[30] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[31] R. Thompson, B. Knyazev, E. Ghalebi, J. Kim, and G. W. Taylor. On evaluation metrics for graph generative models. In *International Conference on Learning Representations*, 2022.

[32] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, and P. Frossard. Digress: Discrete denoising diffusion for graph generation. In *International Conference on Learning Representations*, 2022.

[33] J. W. Wright. The change-making problem. *J. ACM*, 22(1):125–128, jan 1975. ISSN 0004-5411. doi: 10.1145/321864.321874. URL https://doi.org/10.1145/321864.321874.

[34] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5708–5717. PMLR, 2018.

[35] C. Zang and F. Wang. Moflow: an invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 617–626, 2020.