Journal of Machine Learning Research 27 (2026) 1-38 Submitted 5/25; Revised 3/26; Published 4/26

FLAGG: Flexible Autoregressive Graph Generation

Samuel Cognolato SAMUEL.COGNOLATO@PHD.UNIPD.IT
Department of Mathematics, University of Padova, Padova, VE, 35121 Italy;
Fondazione Bruno Kessler (FBK), Trento, TN, 38123 Italy

Alessandro Sperduti ALESSANDRO.SPERDUTI@UNIPD.IT
Department of Mathematics, University of Padova, Padova, VE, 35121 Italy;
Fondazione Bruno Kessler (FBK), Trento, TN, 38123 Italy;
Department of Information Engineering and Computer Science, University of Trento, Trento, TN, 38123 Italy

Luciano Serafini SERAFINI@FBK.EU
Fondazione Bruno Kessler (FBK), Trento, TN, 38123 Italy

Editor: Qiang Liu

Abstract

The Deep Graph Generation's panorama spans two extremes: one-shot and sequential models. The former generates nodes and edges jointly, while the latter samples them autoregressively. Each method performs better in different graph domains depending on size and topology, but neither is applicable to all graph categories. For instance, one-shot methods struggle with generating large graphs, while sequential methods underperform on smaller graphs. A possible way to overcome these limitations is to flexibly combine the two methods in a unique system. In this work, we propose the FLAGG (Flexible Autoregressive Graph Generation) framework, which sequentially generates portions of graphs with one-shot models. FLAGG can apply any one-shot model to make it autoregressive, allowing flexibility in choosing the sequential policy. This policy is specified through a stochastic node removal process, which an Insertion Model learns to reverse. We evaluate FLAGG with the DiGress one-shot model on several data sets of different graph sizes and domains. We show that the approach outperforms both one-shot and autoregressive baselines in terms of sampling quality.

Keywords: graph generation, autoregressive models, one-shot generative models, molecule generation, diffusion models

1. Introduction

Graphs are ubiquitous in many fields of science and technology. Generating new graphs from a reference distribution is a task of paramount importance, for instance, in drug design (Vignac et al., 2023; Huang et al., 2023). The task is usually framed as having an unknown distribution over graphs $p_{\text{data}}(G)$, and the goal is to learn a model $p_\theta(G)$ that approximates it. The model should allow sampling and, when possible, evaluating the likelihood of a given graph. The main challenges encountered for this task include handling the discreteness of graphs, the combinatorial explosion of the sample space, and the need for inductive biases in the model. For instance, a probability distribution over graphs should

be permutation invariant, meaning that equal probability is assigned to any permutation of the nodes.

The field has seen rapid growth in recent years. One motivation is the introduction of new powerful generative frameworks, namely Denoising Diffusion Probabilistic Models (Ho et al., 2020), Score-based Models (Song et al., 2021), and Flow Matching Models (Lipman et al., 2023), which were translated from their native task of image generation to graphs. This wave brought many successful instances (Vignac et al., 2023; Jo et al., 2022; Huang et al., 2023; Chen et al., 2023). This category, also comprising Variational Autoencoders (VAE) (Simonovsky and Komodakis, 2018), Normalizing Flows (NF) (Zang and Wang, 2020) and Energy-Based Models (EBM) (Liu et al., 2021), share a common operative pattern, that is, generating graphs in *one-shot*. This term refers to filling the labels and connections of all nodes jointly in one step. On the other hand, a parallel line of works proposes to build graphs by iteratively adding new nodes and edges in an *autoregressive* manner (Liao et al., 2019; Luo et al., 2021; Kong et al., 2023). This approach is more flexible, allowing for interventions during the generative process (Kong et al., 2023), and larger graph generation thanks to the better memory management (Davies et al., 2023). However, poor parallelizability on hardware and worse sampling performance overshadow these advantages.

Lately, a new trend combining the two distinct families has emerged. The idea is to nest powerful one-shot models inside sequential pipelines, hoping to get the best of both worlds (Liao et al., 2019; Cognolato et al., 2024; Davies et al., 2023). The appeal stands in making models compositional, where the problem of graph generation is broken down into smaller, more manageable chunks. Removing the limitations posed by the particular factorization of probability may lead to a more flexible design.

Following this trend, we present a recipe for building autoregressive models with nested one-shot architectures named Flexible Autoregressive Graph Generation (FLAGG). The two main components are an Insertion Model, which samples how many new nodes to add at each step, and a Filler Model, which fills in the content of the nodes and the connectivity with the intermediate graph being generated. A one-shot model can actually play the Filler Model's role. The two components are trained to reverse a graph noise process, gradually corrupting data graphs until they follow a known probability distribution. The approach we follow for this work is to remove nodes as a noise process, with the empty graph as the absorbing final state. The flexibility stands in the infinite ways to design this node removal process, which can be customized, for example, in the order and the number of node removals at each step.

A key observation from the flexibility is that fully one-shot and sequential models (generating one node at a time) are the two extremes of a continuum of hybrid solutions. This axis of sequentiality allows the model to be adjusted to the required time and memory constraints and can be tuned as a hyperparameter. This was also shown in our preliminary work (Cognolato et al., 2024), which we extend. We still employ DiGress (Vignac et al., 2023) but further explore the customization in multiple directions. First, we refine the IFH model proposed in Cognolato et al. (2024) with architectural modifications, additional features, and training techniques. Second, we design a sparse generation for tackling the problem of generating very large graphs like Cora (Sen et al., 2008a), a citation network of 2810 nodes. For this particular case, we adopt a node selection technique keen

to EDGE (Chen et al., 2023). Finally, we show that the model can be further factorized into smaller modules, each responsible for sampling properties of interest. In this work, we showcase the use of the degree distribution and distance of nodes as properties to be sampled, improving the overall performance in these aspects.

The paper's contributions are summarized as follows:

1. FLAGG extends IFH (Cognolato et al., 2024) in: (i) allowing all nodes' embeddings to change during the filler step; (ii) taking an additional set of auxiliary input features, e.g., spectral and topological features; (iii) using techniques like weights Exponential Moving Average (EMA) (like in Ho et al. (2020)) to stabilize the training process.

2. We build a sparser filler step for generating large graphs, e.g., the sizable citation network Cora.

3. We show that FLAGG can be further factorized into submodules, each responsible for sampling features of interest. This explicit generation is the first step towards a property-aware graph generation.

4. FLAGG improves over the original at intermediate sequentiality levels on benchmark data sets like QM9, Zinc, Community-Small, Ego-Small, Enzymes, and Ego.

2. Related Work

In this section we give a brief overview of the two main families of graph generation models: one-shot and autoregressive. We also discuss the recent trend of generating large graphs.

2.1 One-Shot Graph Generation

One way to generate a graph is to sample its adjacency matrix. One-shot models express $p_\theta(G)$ as the joint probability distribution over adjacency matrix and features, letting all nodes and edges be sampled at once. This procedure assumes that the number of nodes is known in advance. Empirical sampling from the frequencies of graph sizes of a data set is usually employed, but other strategies are possible. Generating the adjacency matrix and features has been tackled through latent variable models like Graph Variational Autoencoders (VAE) (Kipf and Welling, 2016; Simonovsky and Komodakis, 2018), Graph Generative Adversarial Networks (GAN) (De Cao and Kipf, 2018; Martinkus et al., 2022), Normalizing Flows (Zang and Wang, 2020) and Diffusion Models for graphs (Vignac et al., 2023; Chen et al., 2023). The common factor is to define a generative process mapping points from a simple, known distribution to the data space. For example, Diffusion Models gradually denoise a corrupted graph structure starting from a random one. Score-based models for graphs (Jo et al., 2022; Huang et al., 2023) use Stochastic Differential Equations (SDE) to model a continuous flow from the adjacency matrix distribution to the known prior distribution and vice versa.

2.2 Autoregressive Graph Generation

Another way to generate graphs is by sequentially adding new nodes and edges to a growing graph. One then must define a node ordering as a permutation π. This is the approach

taken by autoregressive models. Some works (You et al., 2018; Liao et al., 2019) focus on the architecture of the model in itself, using Recurrent Neural Networks (RNN) or particular Graph Neural Networks (GNN) to compute edge probabilities. Others adopt popular generative frameworks like Normalizing Flows and Diffusion Models in the autoregressive setting (Shi et al., 2020; Luo et al., 2021; Kong et al., 2023). Node ordering has been a hot topic (Chen et al., 2021) and is one of the main criticisms of autoregressive models. Because we usually cannot find a canonical order of nodes, these models must be trained on many permutations, possibly $n!$, to learn permutation invariance. Chen et al. (2021) was the first to introduce the idea of learning the node ordering, showing empirically that results are usually better compared to prefixed orderings such as BFS (Breadth-First Search) or DFS (Depth-First Search). GraphARM (Kong et al., 2023) expanded the idea of learning the ordering through a node absorbing state diffusion model.

2.3 Generating Large Graphs

Recently, there has been a growing interest in modeling larger-scale generation. One of the main challenges gatekeeping the task is the memory and time constraints. One-shot models have to store the whole graph in memory, which scales quadratically with the number of nodes, making their application unfeasible. For this reason, autoregressive models (Liao et al., 2019) have been at the forefront of large graph generation, as they can break down the generation process into smaller steps. Another recent trend that is picking up in pace is that of hierarchical graph generation. The idea is to generate a graph through a series of expansion operations, which gradually increase the resolution of nodes. Recently, the work of Davies et al. (2023) proposed a hierarchical graph generative model based on DiGress (Vignac et al., 2023), which can reconstruct large graphs like Cora (Sen et al., 2008a).

3. Method

In this section, we give a detailed explanation of the methodology, integrated with the novel contributions. FLAGG is an extension of our preliminary work (Cognolato et al., 2024).

3.1 Notation and Definitions

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with nodes $\mathcal{V} = \{v_1, \ldots, v_n\}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The graph's connectivity can also be represented with an adjacency matrix $\mathbf{A} \in \{0,1\}^{n \times n}$, where $\mathbf{A}_{i,j} = 1$ if $(v_i, v_j) \in \mathcal{E}$. If the graph is undirected, $\mathbf{A}$ is symmetric, i.e., $\mathbf{A} = \mathbf{A}^\top$. Labels can be assigned to nodes and edges. In this case we will refer to node labels as $\boldsymbol{X} \in \{0,1\}^{n \times d_x}$, and edge labels as $\mathbf{E} \in \{0,1\}^{n \times n \times d_e}$, represented as tensors of one-hot encoded vectors. Global features of the graph are denoted as $\boldsymbol{y} \in \mathbb{R}^{d_y}$. We start our definitions with the concepts of node removal and induced subgraphs.

Definition 1 (Node removal) *Removing a node v_i from $\mathcal{G}$, denoted as $\mathcal{G} - v_i$, returns a new graph where v_i is removed from the set of vertices $\mathcal{V}$, together with all edges incident to v_i removed from $\mathcal{E}$. Removing multiple nodes at once is denoted as $\mathcal{G} - \mathcal{V}'$, where $\mathcal{V}' \subseteq \mathcal{V}$.*

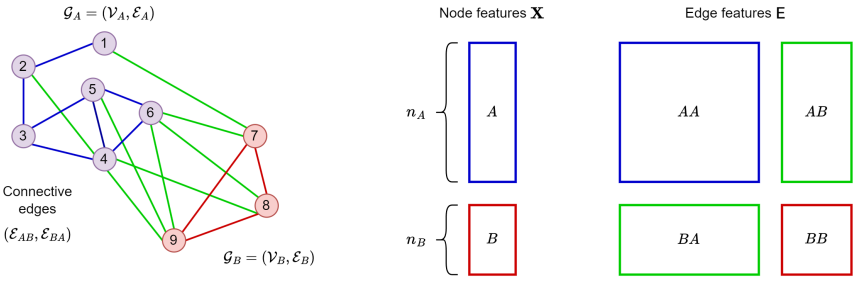


Figure 1: Split operation. The induced subgraphs $\mathcal{G}^A$ and $\mathcal{G}^B$ are blue and red. In green are the intermediate edges $\mathcal{E}^{AB}, \mathcal{E}^{BA}$. On the right is the split adjacency matrix, which has the same coloring.

Definition 2 (Induced subgraph) *The subgraph $\mathcal{G}'$ induced in $\mathcal{G}$ by $\mathcal{V}' \subseteq \mathcal{V}$ is the subgraph with nodes $\mathcal{V}'$ and edges $\mathcal{E}' = \mathcal{E} \cap \mathcal{V}' \times \mathcal{V}'$. It is denoted as $\mathcal{G}[\mathcal{V}']$.*

We can observe that the removal of nodes is equivalent to the induction of subgraphs, as $\mathcal{G} - \mathcal{V}' = \mathcal{G}[\mathcal{V} \setminus \mathcal{V}']$. From the notion of node removal, it follows that if there is a labeling $\boldsymbol{X}, \mathbf{E}$ of nodes and edges, the entries for v_i in $\boldsymbol{X}$ and all its edges in $\mathbf{E}$ are removed. On the opposite side, adding new nodes and edges with labels will concatenate their values to existing nodes and edge labels. To keep the notation simple, we acknowledge this fact now and do not mention it again. Now, we introduce the graph splitting and merging operations, shown in Fig. 1, enabling us to define the forward and reversed node removal sequences for FLAGG. One can observe that the two operations are one the inverse of the other.

Definition 3 (Split operation) *The operation* $\text{split}(\mathcal{G}, \mathcal{V}^A)$ *of* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *through* $\mathcal{V}^A \subseteq \mathcal{V}$ *and* $\mathcal{V}^B = \mathcal{V} \setminus \mathcal{V}^A$ *returns a four-tuple* $(\mathcal{G}^A, \mathcal{G}^B, \mathcal{E}^{AB}, \mathcal{E}^{BA})$ *with:*

$$\mathcal{G}^A = \mathcal{G}[\mathcal{V}^A], \quad \mathcal{G}^B = \mathcal{G}[\mathcal{V}^B],$$
$$\mathcal{E}^{AB} = \mathcal{E} \cap (\mathcal{V}^A \times \mathcal{V}^B), \quad \mathcal{E}^{BA} = \mathcal{E} \cap (\mathcal{V}^B \times \mathcal{V}^A).$$

Definition 4 (Merge operation) *The operation* $\text{merge}(\mathcal{G}^A, \mathcal{G}^B, \mathcal{E}^{AB}, \mathcal{E}^{BA})$ *of two disjoint graphs $\mathcal{G}^A, \mathcal{G}^B$ and their intermediate edges $\mathcal{E}^{AB}, \mathcal{E}^{BA}$ returns a new graph $\mathcal{G}$ with $\mathcal{V} = \mathcal{V}^A \cup \mathcal{V}^B$ and $\mathcal{E} = \mathcal{E}^A \cup \mathcal{E}^B \cup \mathcal{E}^{AB} \cup \mathcal{E}^{BA}$.*

During the explanation of the insertion process of FLAGG, we will also denote the merge operation as $\text{merge}(\mathcal{G}^A, \mathcal{W}^{AB})$, with $\mathcal{W}^{AB} = (\mathcal{G}^B, \mathcal{E}^{AB}, \mathcal{E}^{BA})$, making the notation lighter. $\mathcal{W}^{AB}$ can be considered a new part of the graph being added to $\mathcal{G}^A$. Now, we define the principal object used in FLAGG, namely the forward and reversed removal sequences.

Definition 5 (Forward and reversed removal sequence) *A sequence of graphs $\mathcal{G}_{0:T} = (\mathcal{G}_t)_{t=0}^T$ is a forward removal sequence of $\mathcal{G}$ if $\mathcal{G}_0 = \mathcal{G}$, $\mathcal{G}_T$ is the empty graph $\varnothing$, and $\mathcal{G}_t = \mathcal{G}_{t-1}[\mathcal{V}_t]$ for some $\mathcal{V}_t$. A removal sequence can be reordered in reverse to have $\mathcal{G}_{T:0} = (\mathcal{G}_t)_{t=T}^0$, which is called a reversed removal sequence.*

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.

3.2 Removing Nodes as a Graph Noise Process

Diffusion Models are a staple in modern generative models and have been successfully applied to graphs. Still, not every aspect of graphs has been captured by Diffusion Models. Most prominently, the size of a graph needs to be sampled before running a one-shot model. The FLAGG framework incorporates this missing piece into the mix, defining a diffusion model over the whole graph structure. In this light, a diffusion process will manipulate the graph structure, potentially varying its size along the trajectory. In this work, we define it as a node removal process, which gradually corrupts the graph structure until it collapses into an empty graph.

A node removal process is defined as a Markov Process which, given a graph $\mathcal{G}_{t-1}$ at time $t-1$, samples a subgraph $\mathcal{G}_t$ at time t by removing a set of nodes $\mathcal{V}_t^B \subseteq \mathcal{V}_{t-1}$ from $\mathcal{G}_{t-1}$, i.e., $\mathcal{G}_t = \mathcal{G}_{t-1} - \mathcal{V}_t^B$ (see Definition 1). The transition probability of the process is given by $q(\mathcal{G}_t|\mathcal{G}_{t-1})$. Repeatedly applying the node removal process on a graph $\mathcal{G}$ yields a forward removal sequence $\mathcal{G}_{0:T} = (\mathcal{G}_t)_{t=0}^T$ (see Definition 5). The probability of the forward removal process, given a starting point $\mathcal{G}_0 = \mathcal{G}$, is:

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

The key insight for this autoregressive framework is that the removal transition can be factorized as:

$$\begin{aligned} q(\mathcal{G}_t|\mathcal{G}_{t-1}) &= q(\mathcal{G}_t, n_t|\mathcal{G}_{t-1}) \\ &= q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1}) q(n_t|\mathcal{G}_{t-1}), \end{aligned} \tag{2}$$

where $q(n_t|\mathcal{G}_{t-1})$ is the probability that $\mathcal{G}_t$ will have exactly n_t nodes, and $q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1})$ is the probability of obtaining $\mathcal{G}_t$ by choosing n_t nodes among those in $\mathcal{V}_{t-1}$ to keep. In other words, $q(n_t|\mathcal{G}_{t-1})$ tells *how many* nodes are kept alive, while $q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1})$ determines *which* n_t nodes are selected. This factorization is possible by observing that, among all possible numbers of nodes, the only one having a nonzero probability $q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1})$ is for $n_t = |\mathcal{V}_t|$.

This definition makes no assumptions about the type of graphs or the removal policy. Due to this flexibility, one can design any removal policy, even one tailored to the specific hardware constraints or graph structure. The Absorbing Node State diffusion seen in GraphARM (Kong et al., 2023) shares some similarities with the above definitions, where masking replaces removal. We see it as an instance of the more general framework where the ordering of nodes is learned, and the number of removed nodes is always 1. We touch upon this point again in Section 4.1.

3.3 Parameterizing the Reverse Removal Process

As with Diffusion Models, a neural network has to learn to reverse the removal process defined in the previous section. Reversing a node removal process consists in the introduction of new nodes and edges. This is not trivial, as the specific implementation depends on the nuances of the particular way of removing nodes. In this section, we give the general definition of the reverse process, how to parameterize it, and how to learn its parameters.

The reverse process aims to recover a graph $\mathcal{G}$ corrupted by a removal process, starting from an empty graph. Insertion is done by adding new portions of the graph to the growing

graph across an appropriate amount of time steps. This is formalized by a node insertion process which, given a partial graph $\mathcal{G}_t$, samples a new graph $\mathcal{G}_{t-1}$ with $n_{t-1} \geq n_t$ nodes, and $\mathcal{G}_t$ being its subgraph. This step requires to sample a new subgraph $\mathcal{G}_t^B = (\mathcal{V}_t^B, \mathcal{E}_t^B)$ with $r_t = n_{t-1} - n_t = |\mathcal{V}^B|$ nodes, and edges $\mathcal{E}_t^{AB}, \mathcal{E}_t^{BA}$ to connect it to $\mathcal{G}_t$. Then $\mathcal{G}_{t-1}$ is obtained through a merge operation between $\mathcal{G}_t^A = \mathcal{G}_t$ and the new portion $\mathcal{W}_t^{AB} = (\mathcal{G}_t^B, \mathcal{E}_t^{AB}, \mathcal{E}_t^{BA})$ (see Definition 4). From now on, we use $\mathcal{W}_t$ instead of $\mathcal{W}_t^{AB}$ for brevity. Starting from the empty graph $\varnothing$, we obtain the reversed removal sequence $\mathcal{G}_{T:0} = (\mathcal{G}_t)_{t=T}^0$ (see Definition 5). The insertion transition $p_\Theta(\mathcal{G}_{t-1}|\mathcal{G}_t)$ models one step of it, where Θ are the model's parameters. The reverse process is defined as:

$$p_\Theta(\mathcal{G}_{T:0}) = p_\Theta(\mathcal{G}_T) \prod_{t=1}^{T} p_\Theta(\mathcal{G}_{t-1}|\mathcal{G}_t). \tag{3}$$

Now, this form includes a distribution for $\mathcal{G}_T$, but in the case of the empty graph $\varnothing$, the factor $p_\Theta(\mathcal{G}_T)$ can be removed as all mass is placed on $\varnothing$. As for the removals, also the insertions can be factorized with the same trick:

$$\begin{aligned} p_\Theta(\mathcal{G}_{t-1}|\mathcal{G}_t) &= p_\Theta(\mathcal{G}_{t-1}, r_t|\mathcal{G}_t) \\ &= p_\Theta(\mathcal{W}_t, r_t|\mathcal{G}_t) \\ &= p_\theta(\mathcal{W}_t|\mathcal{G}_t, r_t) p_\phi(r_t|\mathcal{G}_t). \end{aligned} \tag{4}$$

In this context, the Insertion Model $p_\phi(r_t|\mathcal{G}_t)$ samples the number of nodes to add, and the Filler Model $p_\theta(\mathcal{W}_t|\mathcal{G}_t, r_t)$ generates the content of the new nodes and edges, and how to connect them to $\mathcal{G}_t$. The parameters $\Theta = (\theta, \phi)$ are learned by minimizing the variational upper bound, detailed in Cognolato et al. (2024). Briefly, the adopted loss to minimize is:

$$\begin{aligned} L_{\text{vub}} = \mathbb{E}_{\mathcal{G}_0 \sim q(\mathcal{G}_0)} \Bigg[& \sum_{t=2}^{T} D_{\text{KL}}\big(q(r_t|\mathcal{G}_t, \mathcal{G}_0) \| p_\phi(r_t|\mathcal{G}_t)\big) + \\ & + \sum_{t=2}^{T} D_{\text{KL}}\big(q(\mathcal{W}_t|\mathcal{G}_t, r_t) \| p_\theta(\mathcal{W}_t|\mathcal{G}_t, r_t)\big) + \\ & - \mathbb{E}_{\mathcal{G}_1 \sim q(\mathcal{G}_1|\mathcal{G}_0)} \left[\log p_\phi(r_1|\mathcal{G}_1) + \log p_\theta(\mathcal{W}_1|\mathcal{G}_1)\right] \Bigg]. \end{aligned} \tag{5}$$

As with Diffusion Models (Ho et al., 2020) and Score-based Models (Song et al., 2021), the model learns to recover the true data distribution, and not to reconstruct specific data points given their corrupted counterparts. In our case, the generative process starts from $\varnothing$, and expands its support through a series of node additions. As the process evolves, the graphs in the support should resemble subgraphs from the data set.

Another key observation about FLAGG is that $p_\theta(\mathcal{W}_t|\mathcal{G}_t, r_t)$ can be modeled as any one-shot model, as long as it can also generate the interconnections between the new nodes and the partial graph $\mathcal{G}_t$. We detail how this can be done in Section 4.2. This takes FLAGG at the forefront of recent trends of nesting one-shot models in a modular framework, like in HIGGs (Davies et al., 2023) and SaGess (Limnios et al., 2024).

3.4 Controlling Graph Growth through Removal Processes

The framework of FLAGG simplifies the design of the generative process by choosing a formulation of $q(\mathcal{G}_t|\mathcal{G}_{t-1})$. If we decide how nodes are removed, we also get how the model should add them back. The key principles to follow are:

1. The removal process should be reversible. This means that one can find an expression for the reversed process that can be plugged into the model's loss in Equation (5). This lets us train a neural network to approximate the process, thus allowing us to sample from it.

2. The maximum and variance of the number of removed nodes directly impact the memory and time complexities of the algorithms. Control over these quantities is crucial as the filler model needs to materialize the whole adjacency matrix of the new/removed nodes. For example, having high variance and high maximum block size leads to heavy use of padding when batching many adjacency matrices together;

3. Controlling the order in which nodes are removed can positively impact the learning process, as the model can leverage the graph's structure. For example, we can enforce having a single connected component during removals, which translates into a model that tends to connect new subgraphs to the main body.

Thanks to the factorization in Equation (2), we can explore the design of $q(n_t|\mathcal{G}_{t-1})$ as the linkage of nodes is taken care of by the Filler Model. A quantity of notice is the *level of sequentiality*, which we define as the average number of nodes removed at each step, controlling the rate at which the graph grows. We describe two removal processes: the first randomly and independently selects nodes to remove; the second removes blocks of predefined sizes, overcoming the limitations posed by the use of the former naive process. We detail the two methods in the following, and discuss node ordering later. We will refer to r_t as the number of nodes removed at step t, and to $\Delta n_t = n_0 - n_t$ as the number of nodes removed over t steps. All proofs for this section can be found in the supplementary material.

3.4.1 Naive/Binomial

A naive way to approach node removals is to choose, for each node, whether to keep it or remove it with some probability. Formally, we can model the events of removing the nodes of $\mathcal{G}_{t-1}$ as Bernoulli random variables, each with probability q_t. Counting the number of removals, we obtain a Binomial random variable $B(r_t; n_{t-1}, q_t)$. The forward removal transition is then:

$$q(r_t|\mathcal{G}_{t-1}) = B(r_t; n_{t-1}, q_t) \tag{6}$$

Iterating the process many times, starting from $\mathcal{G}_0$, is equivalent to repeating these Bernoulli removals. When marginalized over the intermediate steps, Δn_t is distributed as:

$$q(\Delta n_t|\mathcal{G}_0) = B(\Delta n_t; n_0, 1 - \pi_t) \tag{7}$$
$$\text{with } \pi_t = \prod_{k=1}^{t}(1 - q_k)$$

The target of training, the posterior $q(\mathcal{G}_{t-1}|\mathcal{G}_t, G_0)$, can be obtained using Bayes' rule and is distributed as:

$$q(r_t|\mathcal{G}_t, \mathcal{G}_0) = q(r_t|\Delta n_t) = B(r_t; \Delta n_t, 1 - \bar{q}_t) \quad \text{with } \bar{q}_t = 1 - \frac{1 - \pi_{t-1}}{1 - \pi_t}. \tag{8}$$

Intuitively, this makes sense, as the Binomial models how many nodes r_t to add back among the removed Δn_t nodes. The insertion process can be parameterized as:

$$p_\phi(r_t|G_t) = \sum_{\Delta n_t=0}^{\infty} q(r_t|\Delta n_t) p_\phi(\Delta n_t|\mathcal{G}_t), \tag{9}$$

where $p_\phi(\Delta n_t|G_t)$ predicts the number of missing nodes from $\mathcal{G}_t$. In IFH (Cognolato et al., 2024), it was defined as a neural network predicting the value Δn_t. Then, the KL divergence of Equation (5) is minimum when $p_\phi(\Delta n_t|G_t)$ correctly predicts the true missing nodes, which corresponds to minimizing the Mean Square Error (MSE). This strategy has been evaluated in Cognolato et al. (2024), and presents major drawbacks regarding memory and time complexity. This is motivated by the high maximum and variance of the number of nodes caused by the binomial distribution, leading to a great waste of memory.

3.4.2 Categorical

One way to have a well-behaved distribution over the number of nodes to insert/remove is to restrict the choice of r_t to a small, finite set of c possibilities (block sizes) $D = \{d_1, \ldots, d_c\} \subset \mathbb{N}$. The removal process will then select the number of nodes to remove from D such that their sum results in n_0. A removal transition is therefore modeled as a categorical distribution over these block sizes:

$$q(r_t|\mathcal{G}_{t-1}) = \text{Cat}(r_t; D, q_t), \tag{10}$$

where q_t are the probabilities over the elements of D. Since we want to minimize the number of generation steps and the memory consumption, we have to find the minimal set of blocks that sum up to n_0. The optimal solution can be found in solving the change-making problem (Wright, 1975). The number of nodes is interpreted as the amount to return using as few coins as possible from a given set of denominations D. This problem can be solved in pseudo-polynomial time using dynamic programming. Any permutation of the coins in a solution gives us the shortest possible removal trajectories $\mathcal{G}_{0:T}$ with the possibilities in D. In particular, the number of steps T will always be the number of coins that make up the amount n_0. The process of picking a random coin from the sequence is Markovian, as we only care about the current number of nodes. This is because when we choose a coin r_t and remove it from n_{t-1}, the remaining coins are still the solution of $n_t = n_{t-1} - r_t$. The probabilities q_t are then proportional to the histogram $h(n_{t-1})$ of occurrences of each denomination in the sequence given by n_{t-1}. The categorical transition is defined as:

$$q(r_t = d|G_{t-1}) = \frac{h(n_{t-1})[d]}{T - (t-1)}, \tag{11}$$

where $h(n_{t-1})[d]$ is the histogram value for denomination d. Sampling a denomination can be seen as extracting a colored ball from an urn, where $h(n_{t-1})$ are the number of balls for each color, determining the denominations. Iterating this process from $\mathcal{G}_0$ can be seen as extracting multiple balls from the urn, which starts at $h(n_0)$, and every extraction removes a ball. It can be recognized that the resulting counts of colored balls over t extractions are distributed as a multivariate hypergeometric distribution. The t-step marginal is then:

$$q(\Delta n_t|G_0) = \frac{\prod_{d\in D}\binom{h(n_0)[d]}{h(\Delta n_t)[d]}}{\binom{T}{t}}, \tag{12}$$

where, again, $\Delta n_t = n_0 - n_t$ is the number of missing nodes between $\mathcal{G}_0$ and $\mathcal{G}_t$. Reversing the process yields the posterior:

$$q(r_t = d|G_t, G_0) = q(r_t = d|\Delta n_t) = \frac{h(\Delta n_t)[d]}{t}. \tag{13}$$

Notice that the posterior is precisely the same as the forward transition, considering the amount Δn_t instead of n_t. The task of the insertion model is to predict the distribution of missing nodes $h(\Delta n_t)/t$, and this can be done by minimizing the KL divergence between the posterior and the predicted distribution. We normalize the target into probabilities to avoid learning the scale (i.e., the number of nodes). This process optimizes memory usage by having larger blocks with less padding and reduces time by minimizing the number of steps.

3.4.3 Node Ordering

Other than controlling *how many* nodes are removed, one can also control *which* nodes are removed, as described in Equation (2). Either this is directly implemented in the removal process, like the naive one, which is uniform over all nodes, or it can be predefined before starting the removal process. Particular node orderings can give useful inductive biases to reduce the training examples needed to learn permutation equivariance (Liao et al., 2019; Chen et al., 2021). For example, Breadth First Search (BFS) starts from a random root node and orderly visits neighbors of distance 2, then 3, and so on, which we call layers. Then, the removal process will start from the outermost layer and proceed inwards to the root. BFS ensures that a graph $\mathcal{G}_t$ always has one connected component. In general, given a node ordering π, the transitions will be of the form:

$$\begin{aligned} q(\mathcal{G}_t|\mathcal{G}_{t-1}, \pi) &= q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1}, \pi)q(n_t|\mathcal{G}_{t-1}, \pi) \\ &= q(n_t|\mathcal{G}_{t-1}, \pi). \end{aligned} \tag{14}$$

where $q(\mathcal{G}_t|n_t, \mathcal{G}_{t-1}, \pi)$ deterministically selects the first n_t nodes in the order as $\mathcal{G}_t = \mathcal{G}_{t-1}[v_{\pi(1)}, \ldots, v_{\pi(n_t)}]$. In this case, the design of a removal process can be broken down into choosing a node order and a process on the number of nodes to remove.

3.5 Halting Process

Graphs are inherently arbitrary in size, and removal processes should account for this property to be supported. A fixed number of steps will result in nonuniform block sizes r_t if

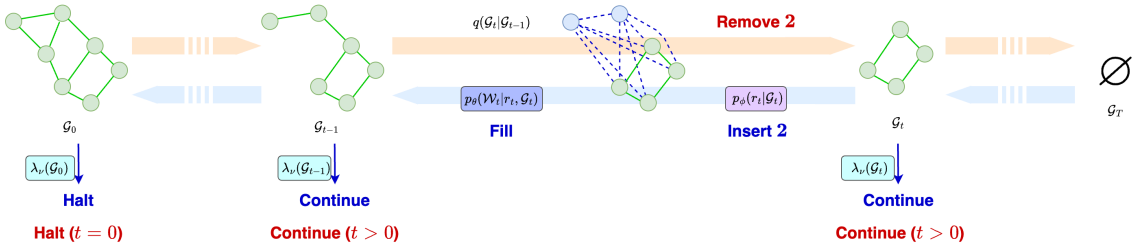


Figure 2: How FLAGG works: during training, a graph is corrupted (left to right) by iteratively removing nodes until the empty graph $\varnothing$ is left. At each step, the insertion (violet), filler (blue), and halt (cyan) models have to predict how many nodes were removed, what content they had, and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

the data set presents a large variability in graph sizes. This can be avoided by adapting the number of steps T to the size of a graph, i.e., taking larger T for larger graphs. The model must infer when to stop the generative process to account for this adaptive generation time. This is already integrated into Insertion Models that predict the number of missing nodes Δn_t. Still, this training objective is hard, as the model tends to learn the average of Δn_t (Cognolato et al., 2024). For the Insertion Models that do not include this mechanism, e.g., the Categorical insertion, we must include a Halting Model, predicting the probability of a partial graph being a terminal graph. This model is trained in a binary classification setup, where the halting ground truth signal is set to 1 for the data set graphs and 0 for all their proper induced subgraphs. This Halting Process has been formalized in Banino et al. (2021), defining a Markov process Λ_t over two states: *continue* (0) and *halt* (1). The process starts in 0, and with probability λ_t jumps to the absorbing state 1. Halting can be integrated into the insertion model as sampling zero as the number of added nodes. By defining $\lambda_{\phi_2}(\mathcal{G}_t) = p_\phi(r_t = 0|\mathcal{G}_t)$, we can write the probability of halting exactly in T steps as:

$$p_\phi\left(\Lambda_0 = 1, \Lambda_{T:1} = \mathbf{0}|\mathcal{G}_{T:0}\right) = \lambda_{\phi_2}(\mathcal{G}_0)\prod_{t=1}^{T}(1 - \lambda_{\phi_2}(\mathcal{G}_t)). \tag{15}$$

In this work, we split the Insertion parameterization ϕ into a Node Insertion Model with parameters ϕ_1 and a Halting Model with parameters ϕ_2.

3.6 The FLAGG Framework

The factorization trick of Equation (2) is central in FLAGG, highlighting the two irreconcilable and highly correlated components of graph generation: selecting the size and building the topology. To our knowledge, a model that merges them end-to-end doesn't exist. For example, one-shot models first sample the size from the empirical distribution and then build the structure for the selected size. On the other hand, FLAGG breaks down generation into a chain of steps, interleaving the insertion of new nodes with the generation

Algorithm 1 Training

repeat
  $\mathcal{G}_0 \sim q(\mathcal{G})$
  $t \leftarrow 1$
  while $\mathcal{G}_{t-1} \neq \varnothing$ **do**
    $\mathcal{G}_t \sim q(\mathcal{G}_t|\mathcal{G}_{t-1})$ ▷ remove nodes
    $r_t \leftarrow n_{t-1} - n_t$ ▷ get true number of nodes
    $\mathcal{W}_t \leftarrow \text{split}(\mathcal{G}_{t-1}, \mathcal{V}_t)$ ▷ get true nodes and edges
    $h_t \leftarrow \mathbf{1}_{t=1}$ ▷ get true halting signal 0/1
    $L_{\text{ins},t}(\phi_1) \leftarrow D_{\text{KL}}\big(q(r_t|\mathcal{G}_t, \mathcal{G}_0) \| p_{\phi_1}(r_t|\mathcal{G}_t)\big)$
    $L_{\text{fill},t}(\theta) \leftarrow D_{\text{KL}}\big(q(\mathcal{W}_t|r_t, \mathcal{G}_t, \mathcal{G}_0) \| p_\theta(\mathcal{W}_t|r_t, \mathcal{G}_t)\big)$
    $L_{\text{halt},t}(\phi_2) \leftarrow L_{\text{halt}}(h_t, \lambda_{\phi_2}(\mathcal{G}_{t-1}))$
  end while
  Perform gradient descent step on

$$\frac{1}{T}\sum_{t=1}^{T}(L_{\text{ins},t}(\phi_1) + L_{\text{fill},t}(\theta) + L_{\text{halt},t}(\phi_2))$$

until converged

of the topology. In particular, FLAGG's Insertion Model $p_\phi(r_t|\mathcal{G}_t)$ and the Filler Model $p_\theta(\mathcal{W}_t|r_t, \mathcal{G}_t)$, are implemented as separate modules, with potentially separate training. For instance, the Filler Model can be implemented by a one-shot model, which also generates the connectivity between graphs. The Insertion Model can be broken down into a halting component and a block size predictive component, as we do in this paper. The whole FLAGG model can be formulated as follows:

$$p_{\theta,\phi}(\mathcal{G}) = \sum_{T=1}^{\infty} \sum_{\mathcal{G}_{T:0} \in \mathcal{R}(\mathcal{G},T)} \underbrace{p_\phi(r_0 = 0|\mathcal{G}_0)}_{\text{halt}}\, p_\theta(\mathcal{G}_T) \prod_{t=1}^{T} \underbrace{p_\theta(\mathcal{W}_t|r_t, \mathcal{G}_t)}_{\text{fill}} \underbrace{p_\phi(r_t|\mathcal{G}_t)}_{\text{insert}}. \tag{16}$$

The modularity of FLAGG allows for different formulations of each part to work in various settings. These inductive biases are first injected in the removal process, which is designed to act on graphs of different topologies. Fig. 2 shows an overview of FLAGG.

The model in Equation (16) is trained following Algorithm 1. Given an example graph $\mathcal{G}$, we first sample an entire removal sequence $\mathcal{G}_{0:T}$. The models parameters ϕ, θ can be optimized by minimizing the variational upper bound $L_{\text{vub}}(\phi_1, \theta)$, defined in Equation (5). We add a binary cross-entropy term L_{halt} to learn the halting signal, with parameters ϕ_2. Sampling from the FLAGG model (16) is expressed in Algorithm 2 and is a loop of node insertions, connecting the new nodes and choosing to halt the loop.

4. Instances of FLAGG

In this section, we outline different ways to implement FLAGG. First, we start with existing models in the literature, showing that they also fit into the FLAGG formulation. Then, we describe the adaptation technique for one-shot models to be used as Filler Models, and

Algorithm 2 Sampling

$\mathcal{G} \leftarrow \varnothing$ ▷ start from the empty graph
repeat
  $r \sim p_{\phi_1}(r|\mathcal{G})$ ▷ sample how many nodes to add
  $\mathcal{W} \sim p_\theta(\mathcal{W}|r, \mathcal{G})$ ▷ sample new nodes and edges
  $\mathcal{G} \leftarrow \text{merge}(\mathcal{G}, \mathcal{W})$
  $h \sim \lambda_{\phi_2}(\mathcal{G})$ ▷ sample halting signal
until $h = 1$
return $\mathcal{G}$

how the different instance of FLAGG affects memory usage and sampling time. Finally, we introduce two additional variants of FLAGG: one for generating large graphs, and one for sampling graph properties of interest.

4.1 Specializing FLAGG to Models in the Literature

One of the aims of FLAGG is to bridge the gap between one-shot and autoregressive models. In this section, we describe how to implement these models with the three blocks of *insertion*, *filling*, and *halting*.

4.1.1 One-Shot Models

For one-shot models (Zang and Wang, 2020; Liu et al., 2021; Martinkus et al., 2022; Vignac et al., 2023; Jo et al., 2022; Huang et al., 2023), all nodes are inserted in one step:

$$p_{\theta,\phi}(\mathcal{G}) = p_\theta(\mathcal{G}|n)p_\phi(n). \tag{17}$$

Insertion: The total number of nodes is sampled from a categorical distribution given by the empirical histogram of graph sizes computed from the training data set.

Filling: The filler model is the one-shot model itself, given the number of nodes to generate.

Halting: The model always halts in one step.

4.1.2 1-Node Sequential Models

These autoregressive models generate one node at a time (Shi et al., 2020; Luo et al., 2021; Kong et al., 2023), and exactly take n steps to generate a graph:

$$p_{\theta,\phi}(\mathcal{G}) = \sum_{\mathcal{G}_{n:0} \in \mathcal{R}(\mathcal{G},n)} \lambda_\phi(\mathcal{G}_0)p_\theta(\mathcal{G}_n)\prod_{t=1}^{n}(1-\lambda_\phi(\mathcal{G}_t))p_\theta(\mathcal{W}_t|r_t,\mathcal{G}_t). \tag{18}$$

Insertion: One node is deterministically inserted at each step.

Filling: Typically, the model samples the node label and an adjacency vector connecting the new node to the current graph. Sometimes, also edges are generated autoregressively (Shi et al., 2020; Luo et al., 2021).

Halting: This model can be implemented in several ways. For example, in You et al. (2018), an End-Of-Sequence (EOS) token is sampled to end generation; in Shi et al. (2020) the number of nodes is fixed at the start, sampled from the empirical distribution; in Luo et al. (2021) generation stops when a threshold is reached, or if the model does not link the new node to the previous subgraph; Han et al. (2023) trains a neural network to predict the halting signal from the adjacency matrix.

4.1.3 Hybrid Models

Other models have tried to fill the gap in autoregressive generation. One of the most well-known is GRAN (Liao et al., 2019), which samples the total number of nodes from the empirical distribution, and generates blocks of a fixed size until the target amount is hit. Then, the overflown nodes are removed, leaving the final graph. The block size is fixed as a hyperparameter, allowing the change in the level of sequentiality. Still, this is a restrictive formulation of block generation, as the Insertion Model of GRAN deterministically chooses one block size. FLAGG generalizes these models by allowing different insertion policies, with different block sizes. Incidentally, FLAGG solves the problem of GRAN eliminating nodes at the end, as we may consider only node-adding operations of different sizes.

4.2 Adapting One-Shot Models for FLAGG

As anticipated in Section 3.3, the Filler Model can be implemented through a one-shot model. We recall that the aim is to generate a new block $\mathcal{W}_t = (\mathcal{G}_t^B, \mathcal{E}_t^{AB}, \mathcal{E}_t^{BA})$ with the new subgraph $\mathcal{G}_t^B$ of r_t nodes and the intermediate edges $\mathcal{E}_t^{AB}, \mathcal{E}_t^{BA}$, and merge it to the previous graph $\mathcal{G}_t^A$ with n_t nodes (Definition 4). When the graphs are undirected, we can just consider $\mathcal{E}_t^{AB}$. Still, a one-shot model only generates a single graph $\mathcal{G}$, so it must be adapted to also generate the links $\mathcal{E}_t^{AB}, \mathcal{E}_t^{BA}$. This can be done by first encoding the topological structure of $\mathcal{G}_t^A$ into its nodes using a Graph Neural Network such as GraphConv (Morris et al., 2019), or RGCN (Schlichtkrull et al., 2018) for labeled data. These encodings are given as input to the architecture of the one-shot model to propagate information from the intermediate graph to the new block. The output of the one-shot model includes the new node labels (if any), the $n_t \times r_t$ rectangular adjacency matrix for intermediate edges, and the $r_t \times r_t$ square adjacency matrix for the new block. After generation, the two partitions are merged and transformed into a sparsely represented partial graph for the next step. This is particularly useful when generating large graphs, as the adjacency matrix can be very sparse. Differently from Cognolato et al. (2024), we let the architecture update the node representations of the partial graph in its neural network layers, having a back-and-forth of information. This is crucial to account for the evolution of new edges during the generative process, particularly in diffusion-based one-shot models. An important implication of this adaptation technique is that any one-shot model can be easily extended to be autoregressive.

4.3 Complexity Considerations

Many factors determine the sampling complexity of an instance of FLAGG. Looking at Algorithm 2, most time is spent generating the new nodes and edges with the Filler Model. For example, in the case of Diffusion Models, which are well known to be time-consuming, having many calls can lead to long sampling times. Parallel hardware like GPUs can accel-

erate a single sampling call, but repeated calls cannot be parallelized. On the other hand, memory consumption scales quadratically with the number of nodes, which hinders using very large blocks, although the sparse representation helps. The flexibility of FLAGG allows trading off memory for compute time and vice versa. On the training side (Algorithm 1), *batching* multiple examples together can create memory bottlenecks, so a removal process should be carefully designed to avoid this, as is done with the Categorical removal process. As explained in Section 3.4, having a high variability and maximum block size can lead to high memory consumption, as smaller blocks will tend to have a lot of padding. See Cognolato et al. (2024) for an empirical evaluation of these effects.

4.4 Generating Large Graphs

Thanks to the sparsity of FLAGG, the model can be used to generate very large graphs. Still, memory usage can be further reduced. One way during training is to avoid sampling the entire removal sequences, but sample a subset of time steps, leveraging the closed form of marginal t-steps distributions. Another less trivial modification is implementing the node selection mechanism of EDGE (Chen et al., 2023), a one-shot Graph Diffusion Model. In our case, we sample a mask over nodes to select a subset for generation at each step. In this case, the Filler Model will generate the connectivity only to this subset. We test this variant in the Experiments Section 5.5.

4.5 Sequential Generation with Properties

The factorization trick in Equation 4 showed us that a generative step can be broken down into a node insertion operation and a connectivity filling operation. We experiment with the addition of new properties to sample in the chain. In general, let $P_{1:k} = (P_i)_{i=1}^{k}$ be a sequence of k properties to sample from an intermediate graph. The filler step will be conditioned on the values of these properties. Then, the FLAGG reverse process can be factorized into the chain:

$$\begin{aligned} p_{\phi,\theta}(\mathcal{G}_{t-1}|\mathcal{G}_t) &= p_\theta(\mathcal{G}_{t-1}|P_{1:k}(\mathcal{G}_{t-1}), \mathcal{G}_t) p_\phi(P_{1:k}(\mathcal{G}_{t-1})|\mathcal{G}_t) \\ &= p_\theta(\mathcal{G}_{t-1}|P_{1:k}(\mathcal{G}_{t-1}), \mathcal{G}_t) \prod_{i=1}^{k} p_{\phi_i}(P_i(\mathcal{G}_{t-1})|P_{1:i-1}(\mathcal{G}_{t-1}), \mathcal{G}_t), \end{aligned} \tag{19}$$

where each p_{ϕ_i} is a model predicting the property P_i of the new graph, given the already predicted properties. We already showed the case of the *number of nodes to insert* property, which we name here as P_1. For a simpler implementation of this variant of FLAGG, we fix the number of nodes to insert to 1. Intending to improve the match with data graphs concerning degree and clustering coefficient distributions, we include the following properties. First, we define P_2 as the *current* degree of the newly inserted node. To enforce this constraint on the edges of the node, we add exactly the number of predicted edges in an autoregressive way. For deciding where to place a new edge, the next model predicts P_3, a set of preference weights over the remaining nodes, proportional to their degrees. The Barabasi-Albert (Albert and Barabási, 2002) models inspire this mechanism, as it incorporates two key properties of real-world networks, i.e., growth and preferential attachment. Growth happens by inserting new nodes, with each new node having an initial degree fixed to

m. Preferential attachment is ensured by making the probability of connecting to a node proportional to its degree: the larger, the more likely it is to connect. These two principles can capture the power-law degree distribution of networks like the World Wide Web, social networks, and biological networks. Another ingredient is to carefully choose the order of removals of nodes, e.g., an exponential distribution proportional to the degree of nodes. This means we first remove nodes with low degrees and so on to the hubs. From the generation perspective, the model first generates the hubs and then the outer nodes. Training a model with P_2 and P_3, and using this order, allows this variant of FLAGG to generate networks following the Barabasi-Albert model. Finally, to capture local structures we also sample preference weights on the distance from the previously linked nodes, which we call P_4. This property is useful when neighbors of a connected node should also be connected, resulting in a better match of node clustering coefficients with the data set. We test this approach in the Experiments Section 5.4.

5. Experiments

In this section, we present the experimental setup and results of FLAGG. We first describe the data sets used for evaluation, the metrics, the baselines considered, and our proposed models. Finally, we present the results and discuss their implications.

5.1 Data Sets

We evaluate FLAGG against the preliminary work Cognolato et al. (2024) and the state of the art, following the same methodology of Cognolato et al. (2024). Due to its challenges, assessing the quality of generated graphs has been a subject of studies in the literature (Thompson et al., 2022). In the graph generative task, one would like samples close in probability to those seen in the data set, but this distance is difficult to define unambiguously. The current metrics are based on the Maximum Mean Discrepancy (MMD) between two sets of graphs, computing pairwise kernel functions $k(\cdot, \cdot)$ on features extracted from graphs using a feature extractor f. MMD is defined as:

$$\mathrm{MMD}^2(p, q) = \mathbb{E}_{\mathcal{G},\mathcal{G}'\sim p}[k_f(\mathcal{G}, \mathcal{G}')] + \mathbb{E}_{\mathcal{G},\mathcal{G}'\sim q}[k_f(\mathcal{G}, \mathcal{G}')] - 2\mathbb{E}_{\mathcal{G}\sim p,\mathcal{G}'\sim q}[k_f(\mathcal{G}, \mathcal{G}')], \tag{20}$$

which can be empirically computed by replacing the expectations with sample averages. The kernel function k that is most often applied is the Radial Basis Function (RBF) kernel, defined as $k(x, x') = \exp(-\gamma\|x - x'\|^2)$. The feature extractor f can be a Neural Network or a function of the graph's adjacency matrix, like the Laplacian Spectrum or the Degree distribution. For this reason, works in this field usually report several metrics, each carrying a different aspect.

5.1.1 Molecular Data Sets

Molecule generation has been a long-standing problem in the field of graph generation. Molecules present a rich structure, with nodes representing atoms and edges representing bonds. In this manuscript we focus on two popular molecular data sets: QM9 (Ramakrishnan et al., 2014), a data set of 133K high-quality small molecules (from 1 to 9 heavy atoms per molecule, with 4 heavy atom types), and ZINC250k (Irwin et al., 2012) with

250K larger and less filtered molecules (from 6 to 38 heavy atoms per molecule, with 9 heavy atom types). As is done in other works (Cognolato et al., 2024; Zang and Wang, 2020; Vignac et al., 2023; Shi et al., 2020), we kekulize the molecules, replacing aromatic bonds with single and double bonds, and remove the hydrogen atoms, using the chemistry library RDKit (Landrum et al.). Hydrogen atoms can be easily added back on unpaired electrons of heavy atoms. For both data sets, we generate 10K molecules, and evaluate the following metrics against the respective test sets, and a 10% of the training set for validation. To measure the quality of sampled molecules we first compute the fraction of (i) valid molecules, which can be reconstructed by RDKit; (ii) unique molecules, that is, the fraction of molecules that are not duplicates; (iii) novel molecules, which are not present in the training set. To compute uniqueness and novelty, we use the canonical SMILES representation, a unique string representation for molecules. We compute the Fréchet ChemNet Distance (FCD) and Neighborhood Subgraph Pairwise Distance Kernel (NSPDK) to evaluate whether the model learned the data set distribution. The two metrics are based on MMD, with FCD getting features from the penultimate layer of ChemNet, a drug activity predictive neural network, and NSPDK accounting for topological features in neighborhood subgraphs. These metrics are computed against the test set.

5.1.2 Generic Graphs Data Sets

We evaluate on graph distributions presenting different topological properties: Community-small (You et al., 2018), with 100 graphs, each with two equally sized communities, linked by a small number of edges; Ego-small and Ego (Sen et al., 2008b), with 200 and 757 graphs, each representing a social network with highly connected hubs; Enzymes (Schomburg et al., 2004), with 563 protein graphs, each representing a protein structure. Preprocessing entails making all graphs undirected and stripped from any pre-existing labeling. In this case, the objective is to learn the topological structure of graphs. We split the train/validation/test sets with the 60/20/20 proportion. Following Huang et al. (2023); Cognolato et al. (2024), we compute the MMD with RBF on the distribution of Degree, Clustering coefficient, Laplacian Spectrum coefficient (Spec.), and random GIN embeddings (Thompson et al., 2022), where the latter can be thought of as 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 their respective test sets.

5.1.3 Large Graphs Generation

To push FLAGG to very large graph generation, we experiment on the Cora citation network (Sen et al., 2008a), with 2810 nodes in the largest connected component. We train FLAGG to generate Cora in an autoregressive way and test whether the statistics match those of the actual network. The metrics used are the same as those in the generic graph data sets, with the addition of Eccentricity, i.e., the average shortest path length. Following Davies et al. (2023), we use the total variation kernel (TV) to compute the MMD of the properties.

5.2 Evaluating FLAGG

We implemented FLAGG using PyTorch (Paszke et al., 2019), PyTorch Lightning (Falcon and The PyTorch Lightning team, 2019) and PyTorch Geometric (Fey and Lenssen, 2019). The configurations for our experiments can be found as Hydra (Yadan, 2019) files. We run each experiment on an A40 GPU, with three seeds from 0 to 2, ensuring reproducibility. The source code is available at `https://github.com/CognacS/flagg-graphgen`.

5.2.1 Vanilla FLAGG

We evaluate the model over the same levels of sequentiality of Cognolato et al. (2024) for comparison. These can be found in the rows of Tables 2a and 2b for molecular data sets, and in Table 1 for generic graphs data sets. We chose not to consider the 1-node sequential and one-shot variants as we found these do not benefit much from the improvements we proposed. We pick DiGress (Vignac et al., 2023) with the optimal prior as the Filler Model, differently from IFH which used DiGress with the uniform prior. We fix the node ordering as the Breadth First Search (BFS) order to enforce the presence of a single connected component, and use the Categorical Removal of Section 3.4, as it is the most efficient and performant (Cognolato et al., 2024). Finally, we implement the new formulation of the Filler Model proposed in Section 4.2, and add new features as input to all neural architectures. These include topological features like the number of cycles of different lengths, and spectral features like eigenvectors and eigenvalues of the Laplacian matrix. The Filler Model is implemented through a Graph Transformer (Dwivedi and Bresson, 2020), the Insertion Model as an RGCN (Schlichtkrull et al., 2018), and the Halting Model as an MLP, only taking global features as input. To overcome the instabilities found in IFH, we integrate techniques such as Exponential Moving Average (EMA) and gradient clipping. To stop training at the right time, we use early stopping. As the stopping criteria, we use the best FCD for molecules and the best GIN MMD for generic graphs, as they offer a comprehensive view of the graph's similarities in the respective domains.

5.2.2 Sparse FLAGG

We keep the same components of the vanilla variant, but reduce the number of sampled steps during training to 5, and fix the block sizes to $\{1, 4, 16, 32\}$. Before calling the Filler Model to generate edges, our model samples a mask over nodes, reducing the number of nodes considered by the Filler Model. Additionally, we replace the BFS ordering for the exponential degree distribution ordering introduced in Section 4.5. From the Barabasi-Albert Model (Albert and Barabási, 2002) point of view, such an ordering emulates that which generated large networks like Cora.

5.2.3 Properties FLAGG

As anticipated in Section 4.5, we factorize FLAGG into submodules, each sampling a specific property or preference weight. Each of these submodules is implemented through an RGCN. During linkage, edges are added autoregressively, and the probability is proportional to the weights predicted by the model. In particular, the weights depend on the nodes' degree and the distance from the already-linked nodes. Finally, the weights are summed, and the edge

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

Table 1: Sequentiality levels on generic graphs, i.e., block sizes used.

is sampled from a categorical distribution with the Softmax of the weights as probabilities. We found that summing the weights instead of computing the chain rule of probabilities leads to better results. Again, we use the exponential degree ordering for this variant. We evaluate Properties-FLAGG (P-FLAGG) on the generic graphs' data sets, also comparing with vanilla FLAGG.

5.3 Results on Molecular Data Sets

5.3.1 Baselines

On molecular data sets, we compare against most of the state-of-the-art models reported in Huang et al. (2023). Specifically, we consider the autoregressive models GraphAF (Shi et al., 2020), GraphDF (Luo et al., 2021), GraphARM (Kong et al., 2023), and the one-shot models MoFlow (Zang and Wang, 2020), DiGress (Vignac et al., 2023), CDGS (Huang et al., 2023). We also compare against IFH (Cognolato et al., 2024), the work we are extending.

5.3.2 Analysis

Looking at Tables 2a and 2b, we observe a substantial improvement of intermediate levels of sequentiality going from IFH to FLAGG, which are now on par with the 1-node sequential variant of IFH. FCD results are the highlights of FLAGG, surpassing all other models in both QM9 and ZINC using large block sizes. This induces us to think that the model is better at incorporating the molecular structure than CDGS, and indeed better than IFH. A reason for the improvement over IFH can be found in the architectural modification introduced in Section 4.2, which allows better information flow and makes the task easier to learn. Regarding the remaining metrics, FLAGG is competitive with the other models, with a validity slightly lower in ZINC.

5.4 Results on Generic Graphs Data Sets

5.4.1 Baselines

On generic graphs data sets, we compare vanilla FLAGG and P-FLAGG with the autoregressive models GraphRNN (You et al., 2018), GRAN (Liao et al., 2019), and the one-shot models VGAE (Simonovsky and Komodakis, 2018), EDP-GNN (Niu et al., 2020), GDSS (Jo et al., 2022), CDGS (Huang et al., 2023). Again, we also compare against IFH (Cognolato et al., 2024), as it underperformed by a great margin for all levels of sequentiality.

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
	DiGress	99.00	5.00e-4	0.360	96.66	33.40
	CDGS	99.68	3.08e-4	0.200	96.83	69.62
IFH	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
FLAGG	{1, 2}	$99.49_{0.07}$	$4.92\text{e-}4_{1.35\text{e-}4}$	$0.212_{0.040}$	$97.02_{0.11}$	$76.92_{0.21}$
	{1, 4}	$99.69_{0.17}$	$3.05\text{e-}4_{0.36\text{e-}4}$	$\mathbf{0.144}_{0.010}$	$96.96_{0.37}$	$76.04_{0.91}$

(a) Performance results on the QM9 data set

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
	DiGress	91.02	0.082	23.06	81.23	100.00
	CDGS	98.13	**7.03e-4**	2.069	99.99	99.99
IFH	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
FLAGG	{1, 3}	$97.51_{1.13}$	$0.002_{4\text{e-}4}$	$2.096_{0.202}$	$99.97_{0.04}$	$99.98_{6\text{e-}5}$
	{1, 4, 8}	$97.81_{0.09}$	$14.85\text{e-}4_{3\text{e-}5}$	$\mathbf{1.810}_{0.030}$	$100.00_{6\text{e-}5}$	$99.98_{6\text{e-}5}$

(b) Performance results on the ZINC250K data set

Method	Deg.↓	Clus.↓	Eccent.↓	Spec.↓
HiGGs	0.254	**0.958**	**0.273**	–
Sparse FLAGG	$\mathbf{0.020}_{0.013}$	$2.0_{0.0}$	$0.751_{0.061}$	$\mathbf{0.025}_{0.008}$

(c) Performance results on the Cora data set

Method		Community $\lvert V\rvert_{\max}=20,\ \lvert E\rvert_{\max}=62$ $\lvert V\rvert_{\text{avg}}\approx 15,\ \lvert E\rvert_{\text{avg}}\approx 36$				Ego-small $\lvert V\rvert_{\max}=17,\ \lvert E\rvert_{\max}=66$ $\lvert V\rvert_{\text{avg}}\approx 6,\ \lvert E\rvert_{\text{avg}}\approx 9$				Enzymes $\lvert V\rvert_{\max}=125,\ \lvert E\rvert_{\max}=149$ $\lvert V\rvert_{\text{avg}}\approx 33,\ \lvert E\rvert_{\text{avg}}\approx 63$				Ego $\lvert V\rvert_{\max}=399,\ \lvert E\rvert_{\max}=1071$ $\lvert V\rvert_{\text{avg}}\approx 145,\ \lvert E\rvert_{\text{avg}}\approx 335$			
		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
	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**
IFH	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
FLAGG	seq-small	$0.091_{0.004}$	$0.108_{0.009}$	$0.051_{0.002}$	$0.126_{0.006}$	$\mathbf{0.023}_{0.002}$	$\mathbf{0.015}_{0.001}$	$\mathbf{0.023}_{0.002}$	$0.031_{0.004}$	$0.048_{0.012}$	$\mathbf{0.035}_{0.007}$	$\mathbf{0.026}_{0.004}$	$\mathbf{0.022}_{0.004}$	$0.060_{0.018}$	$0.061_{0.011}$	$0.099_{0.052}$	$0.075_{0.009}$
	seq-big	$0.092_{0.010}$	$0.085_{0.018}$	$0.056_{0.001}$	$0.113_{0.016}$	$0.027_{0.002}$	$\mathbf{0.015}_{0.002}$	$0.028_{0.002}$	$0.033_{0.001}$	$0.071_{0.014}$	$0.061_{0.022}$	$0.042_{0.011}$	$0.054_{0.016}$	$0.124_{0.027}$	$0.177_{0.045}$	$0.117_{0.047}$	$0.051_{0.008}$
FLAGG with properties		$0.073_{0.002}$	$\mathbf{0.069}_{0.006}$	$\mathbf{0.049}_{0.003}$	$0.094_{0.010}$	$0.028_{0.003}$	$0.020_{0.002}$	$0.029_{0.003}$	$0.034_{0.004}$	$\mathbf{0.030}_{0.008}$	$0.067_{0.028}$	$0.030_{0.004}$	$\mathbf{0.022}_{0.003}$	$\mathbf{0.033}_{0.001}$	$\mathbf{0.057}_{0.015}$	$0.029_{0.017}$	$0.037_{0.004}$

(d) Performance results on generic graphs data sets

Table 2: Results on the molecule generation task on QM9 (a), ZINC250k (b), Cora (c) and generic graphs (d) averaged over 3 runs. Best results are in bold, and the second best are underlined. Values are reported as mean_{std}.

5.4.2 Analysis

Comparing our vanilla FLAGG variants with those of IFH in Table 2d, we see a massive improvement in all metrics. Not only we managed to improve all intermediate models, but we also achieved better results than CDGS in the Ego-small and Enzymes data sets. Again, we argue that the improvements introduced in this work are the reason for the better results. Still, we observe that for Community-small and Ego, our vanilla FLAGG lags behind CDGS. In particular, the Degree and Clustering Coefficient metrics are lower, meaning that the model generates a slightly different number of edges and potentially does not fully capture the community structure measured by the clustering coefficients. Then, we extend the comparison with P-FLAGG, which explicitly learns these features. We want to be clear that the model is not *informed* about which property value is the right one, but rather, this decision is made explicit in the factorization and is learned during training. Looking at the two above-mentioned metrics, we observe an improvement over all data sets but Ego-small, where results are already satisfying. This, combined with its low sampling time, makes P-FLAGG a promising direction for future research.

5.5 Results on Huge Graphs Generation

5.5.1 Baselines

We compare large-scale generation of Cora against HiGGs (Davies et al., 2023). For this purpose, we defined a sparse version of FLAGG, which is more memory efficient and can generate graphs of this size.

5.5.2 Analysis

Results on this challenging benchmark, shown in Table 2c, suggest that FLAGG captures very well the degree distribution of Cora and its spectral information but lags in terms of Clustering Coefficients and Eccentricity. This could be explained by HiGGs's algorithm, which first generates the skeleton of the network, and finally expands the substructures. This might help in capturing the local structure of the graph. On the other hand, first generating the communities, and then linking them in parallel might overshoot the number of edges generated, motivating the worse Degree metric of HiGGs.

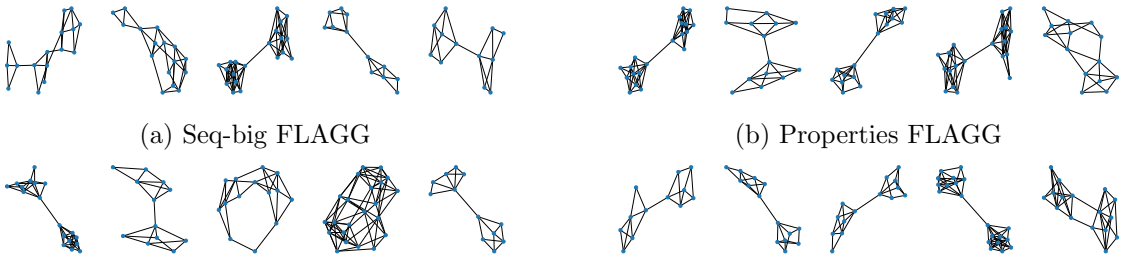
5.6 Qualitative Results

After discussing the quantitative results in the previous section, we briefly evaluate the quality of generated samples from a qualitative perspective. Figures 3, 4, 5, 6, 7, and 8 show random batches generated by each of our models, together with random samples from the training datasets. Generally, we observe that geometric properties are respected, and molecular structures are valid.

5.7 Sampling Efficiency

5.7.1 Sampling time

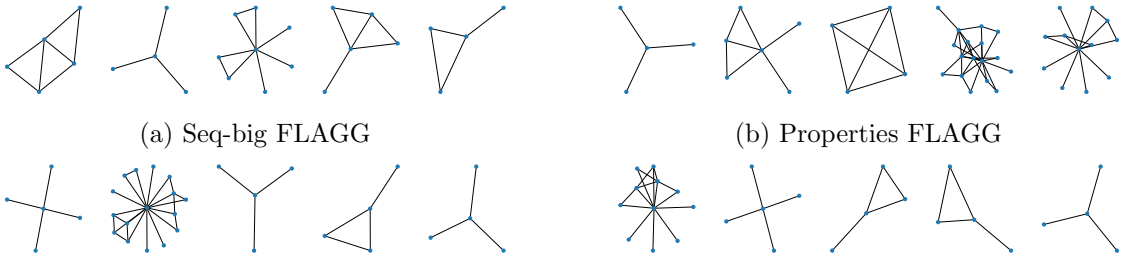
In Table 3 we report the sampling time of all FLAGG variants on all data sets. A clear trend can be observed: sampling time decreases as the block size increases. This reflects



(a) Seq-big FLAGG (b) Properties FLAGG

(c) Seq-small FLAGG (d) Data

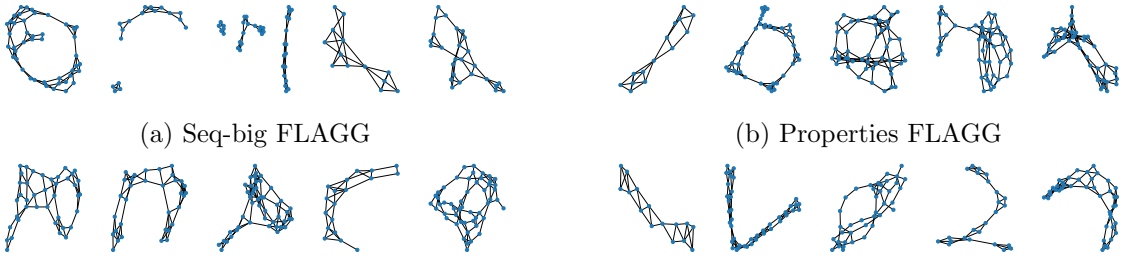
Figure 3: Community samples.



(a) Seq-big FLAGG (b) Properties FLAGG

(c) Seq-small FLAGG (d) Data

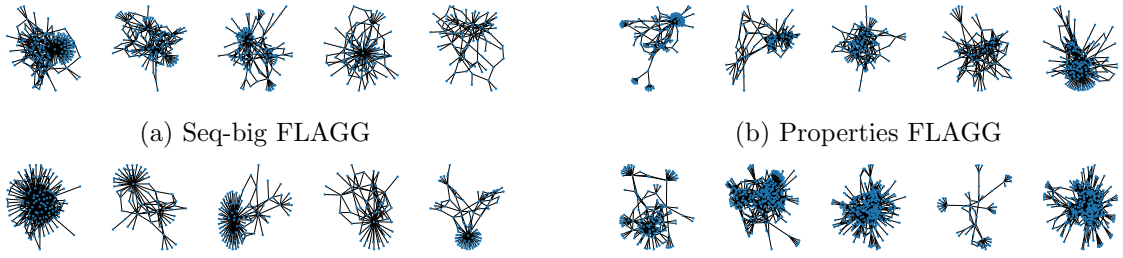
Figure 4: Ego-Small samples.



(a) Seq-big FLAGG (b) Properties FLAGG

(c) Seq-small FLAGG (d) Data

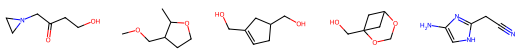
Figure 5: Enzymes samples.



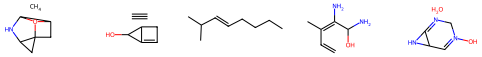
(a) Seq-big FLAGG (b) Properties FLAGG

(c) Seq-small FLAGG (d) Data

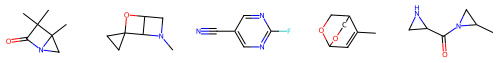
Figure 6: Ego samples.



(a) Seq-big FLAGG

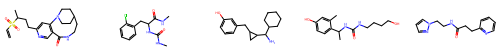


(b) Seq-small FLAGG

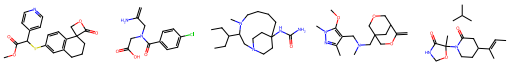


(c) Data

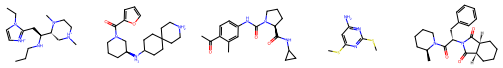
Figure 7: QM9 samples.



(a) Seq-big FLAGG



(b) Seq-small FLAGG



(c) Data

Figure 8: Zinc samples.

the considerations made in Section 4.3, as using larger blocks drastically reduces the number of sequential steps, and increases parallelization on GPU. A notable exception is the Properties-FLAGG, which is orders of magnitude faster than other methods, although being fully sequential, i.e., one node and one edge at a time. For instance, in the Ego dataset `properties` FLAGG's time is roughly 225 times smaller than `seq-small` FLAGG. This is due to the algorithm being fully autoregressive, avoiding the use of expensive Diffusion Models. We investigate whether we can obtain speed-ups in Diffusion-based models in the next section.

5.7.2 Compute Time vs. Performance

We investigate whether a speed-up in Diffusion-based FLAGG variants can be obtained. In particular, we explore the use of a lighter Filler Model, evaluating its sampling time and how it affects the sampling quality. For this purpose, we pick the best performing FLAGG variant on the QM9 molecular data set, i.e., the $\{1, 4\}$-blocks model, and we vary the number of denoising steps performed by DiGress. As explained in Nichol and Dhariwal (2021), it is possible to reduce the number of denoising steps from T to $K = T/J$ by defining a jump J such that 1 step in the reduced process corresponds to J steps in the original process. We define a set of reduced numbers of steps $K = 1, 2, 5, 10, 20, 50, 100, 250, 500$, where $T = 500$. Results in Figure 9 show that reducing the number of steps by up to a factor of $J = 10$ keeps the validity over 99% and the FCD at a very low value, while reducing the sampling time by a factor of ~ 10. After this threshold, FCD and validity start to rapidly deteriorate until, at $K = 1$ step, validity sits at 55.4% and FCD at 6.788. This behavior is consistent

Method	QM9	ZINC250k
seq-small	$122.616_{21.507}$	$295.300_{18.903}$
seq-big	$96.017_{16.266}$	$151.949_{6.764}$

(a) Sampling time (in minutes) on the molecular datasets.

Method	Cora
sparse	$45.865_{1.473}$

(b) Sampling time (in minutes) on the Cora dataset.

Method	Community	Ego-small	Enzymes	Ego
seq-small	$70.710_{2.659}$	$54.071_{6.470}$	$38.831_{9.527}$	$283.899_{19.988}$
seq-big	$22.013_{2.058}$	$30.268_{2.003}$	$18.840_{5.091}$	$67.213_{7.579}$
properties	$0.170_{0.024}$	$0.103_{0.005}$	$0.118_{0.003}$	$1.259_{0.130}$

(c) Sampling time (in minutes) on the generic graph datasets.

Table 3: Sampling time of the test batch of molecular graphs (a), Cora (b) and generic graphs (c) averaged over 3 runs. Values are reported as mean_{std}.

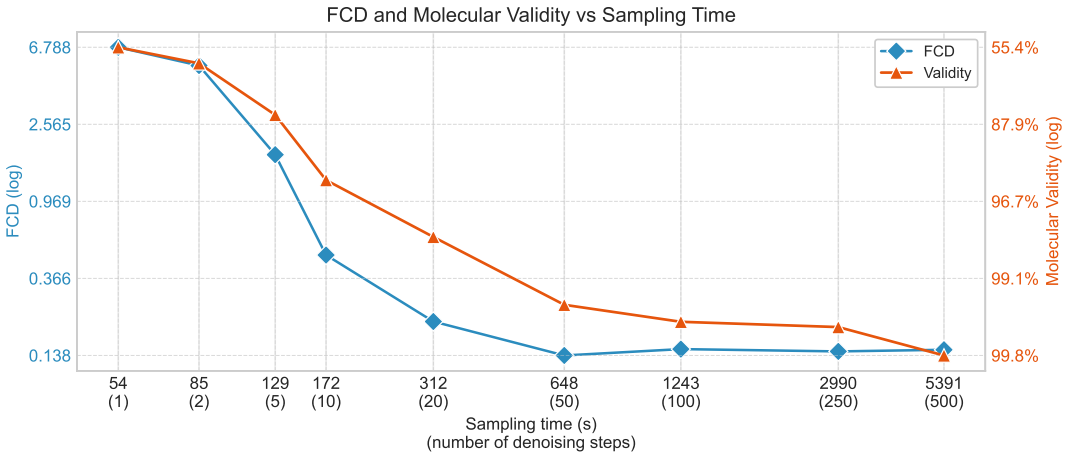


Figure 9: Performance vs. sampling time of Vanilla FLAGG with different numbers of denoising steps on the QM9 data set, using block sizes $\{1, 4\}$. Performance is evaluated in terms of FCD (blue line) and Molecular Validity (orange line). Vertical lines represent runs with different numbers of denoising steps, shown at the center of the line. Each point is the average across 3 different models trained with different seeds. All scales are logarithmic.

with what was found in Nichol and Dhariwal (2021) for image generation with diffusion models. Regarding the relation between K and sampling time, it can be seen that they are almost always directly proportional, with J being the proportionality constant. This is not

Method	Valid (%)↑	NSPDK↓	FCD↓	Unique (%)↑	Novel (%)↑
BFS	$\underline{99.69}_{0.17}$	$\mathbf{3.05\text{e-}4}_{0.36\text{e-}4}$	$\mathbf{0.144}_{0.010}$	$96.96_{0.37}$	$76.05_{0.91}$
DFS	$99.65_{0.15}$	$\underline{3.24\text{e-}4}_{0.34\text{e-}4}$	$\underline{0.153}_{0.018}$	$97.13_{0.13}$	$77.52_{1.20}$
Random	$\mathbf{99.72}_{0.14}$	$3.42\text{e-}4_{0.38\text{e-}4}$	$0.253_{0.043}$	$96.95_{0.21}$	$77.80_{1.40}$

Table 4: Ablation study results for node orderings on the molecule generation task on QM9, averaged over 3 runs. Best results are in bold, and the second best are underlined. Values are reported as mean_{std}.

Blocks	Insertion	Deg.↓	Clus.↓	Spec.↓	GIN↓
{1, 3}	Learned	$\mathbf{0.048}_{0.009}$	$\mathbf{0.043}_{0.02}$	$0.029_{0.003}$	$\mathbf{0.019}_{0.001}$
	Empirical	$0.050_{0.007}$	$0.044_{0.03}$	$\mathbf{0.025}_{0.006}$	$\mathbf{0.019}_{0.002}$
{1, 2, 8}	Learned	$\mathbf{0.062}_{0.012}$	$\mathbf{0.070}_{0.034}$	$0.041_{0.010}$	$0.048_{0.025}$
	Empirical	$0.067_{0.017}$	$0.072_{0.042}$	$\mathbf{0.031}_{0.007}$	$\mathbf{0.037}_{0.015}$

Table 5: Comparison of FLAGG with an "Empirical" baseline, sampling the number of nodes from the empirical distribution, averaged over 3 runs on the Enzymes dataset. Our method, sampling block sizes and the halting signal through learned neural networks, is identified as "Learned". Best results are in bold, and the second best are underlined. Values are reported as mean_{std}.

a perfect relation as the filler model is only a submodule of FLAGG, and fixed costs for insertion and halting are always present.

5.8 Ablation Studies

5.8.1 On the Node Ordering

Recalling from Section 3.4.3, the node ordering is fixed by the experimenter as a hyperparameter of the removal process, in our case the BFS ordering. Then, FLAGG learns to reverse the particular ordering used. Thus, we investigate the impact of this particular ordering. For this purpose, we train two additional variants of the Vanilla FLAGG model on the QM9 dataset with the best performing set of blocks (i.e., $\{1, 4\}$). The first variant uses the Depth First Search (DFS) ordering, which has also been investigated in Liao et al. (2019), and the second variant uses a simple random ordering. Results comparing BFS, DFS and random are shown in Table 4. With respect to molecular validity, all orderings reach $> 99\%$, so the true comparison must be made on the remaining metrics. BFS achieves the best value of FCD and NSPDK, suggesting that it was better suited for learning the molecular structure of QM9. Still, DFS comes close to BFS, while the random ordering is worse, especially in terms of FCD, where it is higher by 75%.

5.8.2 On the Halting Model

The halting model is a critical part of the FLAGG framework. Thus, we evaluate the same framework using a static insertion process, sampling the total number of nodes from the empirical distribution of the training set. Consequently, the block sizes to be inserted are sampled from the removal process, as defined in Sec. 3.4.2. Results are shown in Table 5, using our trained checkpoints on the Enzymes dataset, and replacing the learned components with empirical ones as described above. An immediate observation is that the learned model dominates on the local metrics like Degree and Clustering Coefficients, while the empirical one performs better on global metrics, i.e., the spectrum and the GIN embeddings. This suggests that trained halting and insertion models may allow a better replication of local structures from the dataset, but, if not perfectly mirroring the empirical distribution, they might lead to a shift in the graph distribution, resulting in different global properties. We check this shift in graph sizes in the next section.

5.9 Analysis of the Learned Insertion Policy

We investigate the insertion policy of FLAGG by studying the learned behaviors of its three modules: (i) the Halting model's predicted distribution over the number of nodes, and how it compares to the empirical distribution of the training set; (ii) the preferred block sizes of the Insertion model, depending on time; (iii) the Filler model's learned node ordering. We ground the analysis on the Enzymes dataset, as it presents medium-to-high complexity relative to the presented data sets. It has a non-trivial empirical distribution of the number of nodes, and graphs have both long chains and cycles. We generate a batch of N graphs $\mathcal{B} = \{\mathcal{G}^1, \dots, \mathcal{G}^N\}$, keeping track of intermediate graphs $(\mathcal{G}_t^i)_{t=0}^T$, where we reverse the ordering of time for clarity (i.e., $\mathcal{G}_0^i = \emptyset$ and $\mathcal{G}_T^i = \mathcal{G}^i$), and we refer to it in the following sections.

5.9.1 On Halting and Graph Sizes

First, we check whether the halting model, together with the insertion model, can reconstruct the empirical distribution of the number of nodes. Relevant plots can be found in Figure 10. We compare two different distributions computed from $\mathcal{B}$: the *Generated* node-count frequencies $h_{\text{gen}}(n) = |\{i : |\mathcal{V}^i| = n\}|/N$, and the distribution computed from the halting *Prior* distribution:

$$p^i_{\text{prior}}(n) = \sum_{t:\ |\mathcal{V}_t^i|=n} \lambda_{\phi_2}(\mathcal{G}_t^i) \prod_{s=1}^{t-1} (1 - \lambda_{\phi_2}(\mathcal{G}_s^i)), \tag{21}$$

$$h_{\text{prior}}(n) = \frac{1}{N} \sum_{i=1}^{N} p^i_{\text{prior}}(n), \tag{22}$$

where $\lambda_{\phi_2}(\mathcal{G})$ are the halting probabilities produced by the model to create the batch $\mathcal{B}$. From Figure 10, we can see that the support of both the Prior and Generated histograms matches that of the Dataset, although the probability mass leans towards lower values, meaning the model is sometimes undershooting. An interesting fact that can be observed is how the use of blocks shapes the histograms: in the $\{1, 3\}$ case, both the Prior and

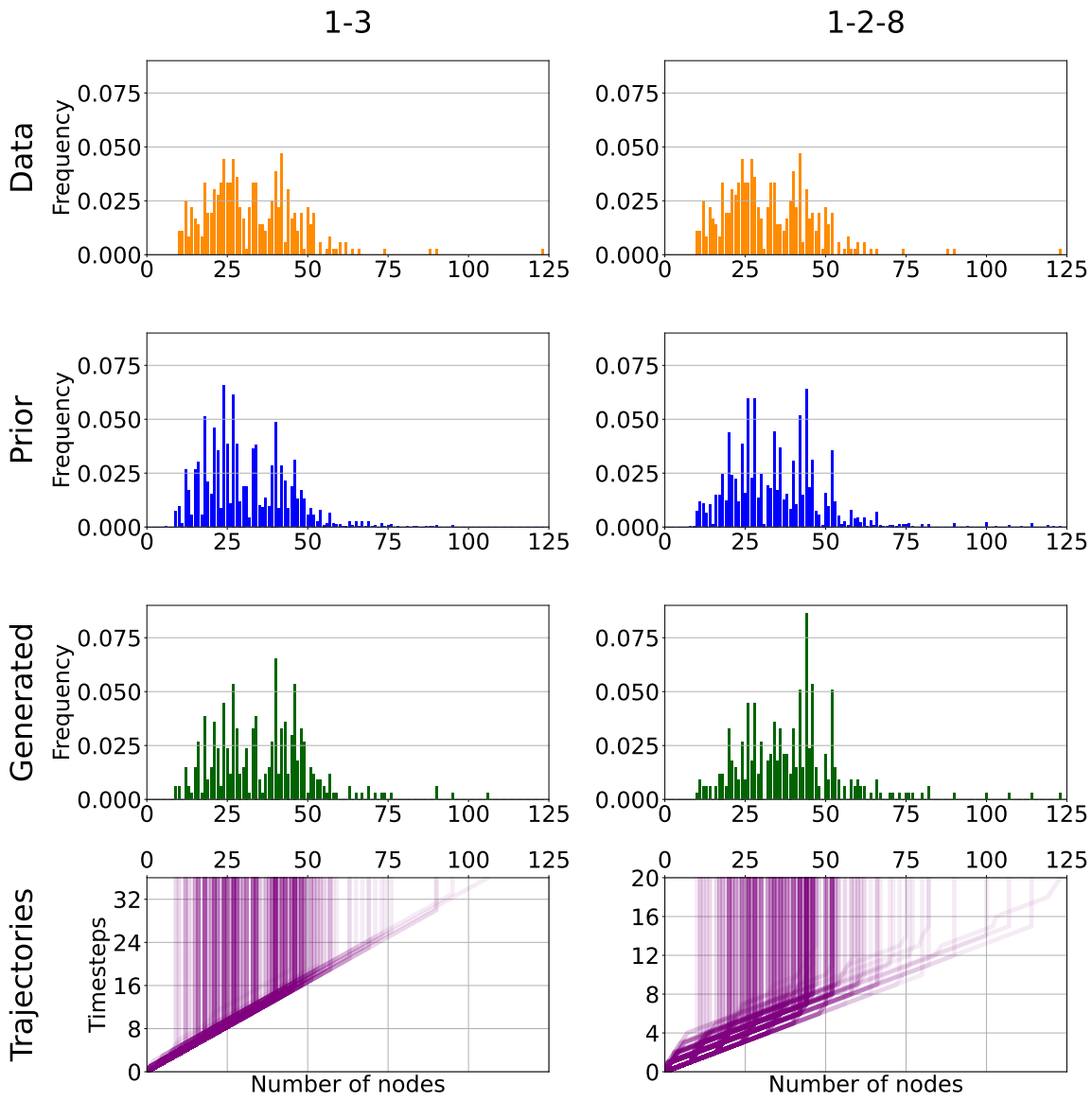


Figure 10: Analysis of the learned insertion policy on the Enzymes dataset. The two columns represent our two variants of FLAGG, trained with different sets of block sizes. The "Data" row shows the empirical distribution of the number of nodes in the training set of Enzymes. The "Prior" row shows the average distribution on nodes, given by the prior probability of halting at each timestep. The "Generated" row shows the number of nodes actually sampled. The "Trajectories" row shows the paths taken by each graph during its generative trajectory, where the y-axis represents time flowing from bottom to top. The intensity of the color represents the number of overlapping trajectories. Thus, the latter two rows are aligned, with the "Generated" histogram being an alternative representation of the x-axis of the "Trajectories" at the final timestep.

Generated histograms present peaks with an interval of 3, which is the most frequent block size judging from the trajectories. The same happens in the $\{1, 2, 8\}$ case, with an interval of 2. This suggests that the insertion policy is overconfident on block predictions, preferring more frequent block sizes. Still, we don't see signs of overshooting, i.e., graphs larger than the dataset's limit. On the bottom row, we show the trajectories created by the graphs in batch $\mathcal{B}$ with respect to the growth in size over time. For the case of $\{1, 3\}$ we can in fact see that graph sizes rapidly evolve by picking the biggest block size, with the halting model contributing to the shape of the histogram by stopping at convenient times. In the $\{1, 2, 8\}$ regime, block sizes are more heterogeneous, and we even see that getting close to the upper bound of the number of nodes, the model starts to prefer smaller blocks sizes, and finally halting generation. We continue the analysis of block sizes and node ordering in the next sections.

5.9.2 On Block Sizes and Node Ordering

We now turn to the policy on block sizes and node ordering. We refer the reader to Figure 11. The plots in the top row confirm the hypotheses of the previous section, showing that the $\{1, 3\}$ model prefers the block size of 3, while the $\{1, 2, 8\}$ model varies among the three. From this plot, we can also observe the behavior of picking only the 2 block at the end of generation by the $\{1, 2, 8\}$ model. The graphs in Figure 11 show the order in which nodes and edges are generated at each step. We can see that in most graphs, the model mimics the BFS ordering, for example in long chains, where either they are generated from one end to the other, or they are starting from the middle and growing in both directions. Still, we observe that in some cases, particularly when having cycles, blocks forward in time are connected with older blocks. We argue that this is caused by the model not having memory on the order in which nodes were generated.

5.10 Discussion

From the experimental results, we can deduce that FLAGG with nested DiGress is a strong model, managing to generate high-quality graphs with an autoregressive pattern. This can be useful for learning the distribution over nodes in a conditional generation setup, where we can't use the empirical distribution. FLAGG may also be applied to expand already existing graphs. The improvement achieved by FLAGG, relative to IFH (Cognolato et al., 2024), can be attributed to the architectural modifications, the new topological and spectral features, and the use of techniques like EMA and gradient clipping, as clearly shown in Table 2. Results also suggest that not having built-in permutation invariance in the model did not hinder its sample quality. Furthermore, we notice that changing the block sizes does not significantly impact the model's performance. This suggests that, differently from the points raised in Cognolato et al. (2024), and in the setup of the present work, FLAGG is not particularly sensitive to the choice of the sequentiality level. We advise more research to pinpoint whether this holds for other removal processes, orderings, and filler models.

Another strength of FLAGG is the flexibility of its design. In Cognolato et al. (2024) we showed that increasing the number of steps leads to lower memory consumption but higher time complexity. In this work we showed that the implementation can also be tailored around a particular task, or to boost the similarity with the data set for some property. For

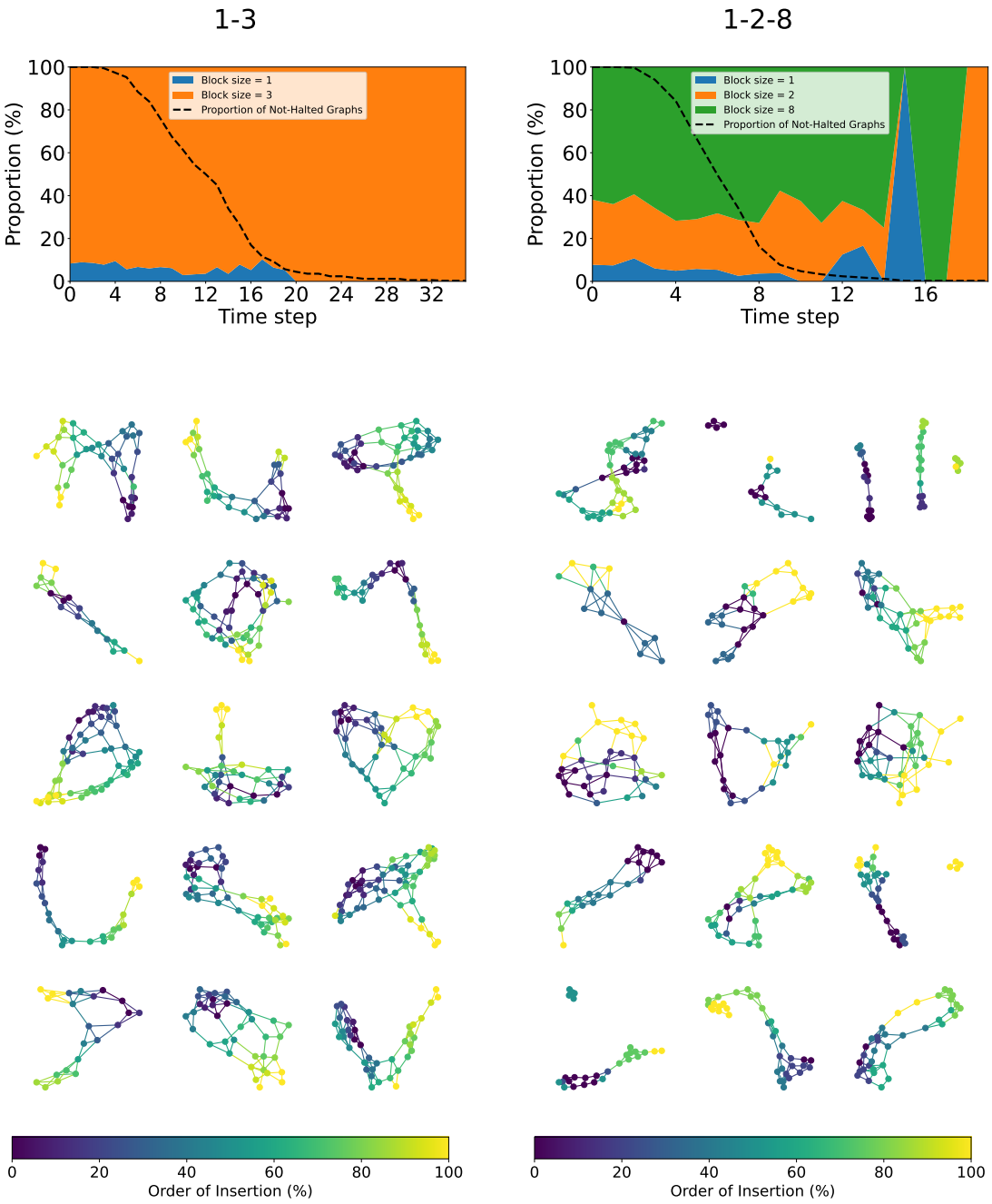


Figure 11: Top row: proportion (y-axis) of block sizes used during generation at each time step (x-axis) on the running batch of graphs, where the black dashed line indicates its size. Bottom row: example of generated graphs with node and edge colors indicating the % time step at which they were generated.

example, P-FLAGG improved its adherence to the degree and clustering distribution of the data graphs, thanks to explicitly learning them. Additionally, P-FLAGG proved to be a light-weight model, reducing sampling time by 2 orders of magnitude. Another advantage of FLAGG is the capability to generate very large graphs, as shown with the Cora data set. On this front, we still see the ground for improvement, as the Clustering metric was worse than HiGGs, which can better capture local information.

For future research, we would like to explore new removal processes. For instance, knowing that the data set consists of communities can be exploited in the generative process. This could be built into the removal process by interleaving the ordering of removals with the block sizes. Another direction is to increase the number of topological properties that FLAGG can incorporate, with the hope of having a better interaction between the symbolic and neural aspects of the model. A current limitation of FLAGG is that the removal process parameters are fixed by the experimenter. One way to learn them is through hyperparameter search, which may prove expensive. Thus, a direct extension of the insertion process would have the model learn an optimal ordering of nodes and block sizes directly from the data, avoiding the need for manual tuning. This extension requires major efforts and additional theoretical background on the specifics, so we leave it as future work.

6. Conclusion

In this paper we propose the Flexible Autoregressive Graph Generation (FLAGG) framework, allowing the harnessing of the generative power of one-shot models in a sequential setting. We improved over our preliminary model IFH, showing that generating with blocks is feasible and leads to high-quality graph generation. We also showed that thanks to the flexibility of FLAGG, we can define several variants, each with its advantages. For example, we can make the model consider subsets of nodes instead of the whole graph to generate large graphs. For boosting the learning of specific properties, we can make those explicit through the factorization trick of FLAGG. Finally, we observed that, in our setup, the model is not very sensible to the choice of the block sizes, but we require further research in this direction. We believe that FLAGG, with its freedom of design, is a promising framework for spurring new interest in autoregressive graph generation.

Acknowledgments

We acknowledge the financial support of (i) the PNRR project FAIR — Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU; (ii) the Italian Ministry of University and Research (MUR) under the PRIN program – Progetti di Rilevante Interesse Nazionale – PRIN 2022 (Secretary-General's Decree No. 1401 of 18/09/2024) – CUP C53C24000770006, project title "DEEP-GRAPH: Design and Theory of Deep Graph Learning".

References

Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. In *ICML 2021 Workshop on Automated Machine Learning (AutoML)*, 2021.

Xiaohui Chen, Xu Han, Jiajing Hu, Francisco Ruiz, and Liping Liu. Order matters: Probabilistic modeling of node sequence for graph generation. In *International Conference on Machine Learning*, pages 1630–1639. PMLR, 2021.

Xiaohui Chen, Jiaxing He, Xu Han, and Liping Liu. Efficient and degree-guided graph generation via discrete diffusion modeling. In *International Conference on Machine Learning*, pages 4585–4610. PMLR, 2023.

Samuel Cognolato, Alessandro Sperduti, and Luciano Serafini. Ifh: a diffusion framework for flexible design of graph generative models. In *European Conference on Artificial Intelligence*, pages 3039–3046. IOS Press, 2024.

Alex Owen Davies, Nirav Ajmeri, and Telmo M Silva Filho. Size matters: Large graph generation with hiGGs. In *NeurIPS 2023 Workshop on Synthetic Data Generation with Generative AI*, 2023.

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

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL `https://github.com/Lightning-AI/lightning`.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Xu Han, Xiaohui Chen, Francisco JR Ruiz, and Li-Ping Liu. Fitting autoregressive graph generative models through maximum likelihood estimation. *Journal of Machine Learning Research*, 24(97):1–30, 2023.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Han Huang, Leilei Sun, Bowen Du, and Weifeng 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.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

Jaehyeong Jo, Seul Lee, and Sung Ju 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.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Lingkai Kong, Jiaming Cui, Haotian Sun, Yuchen Zhuang, B Aditya Prakash, and Chao Zhang. Autoregressive diffusion model for graph generation. In *International Conference on Machine Learning*, pages 17391–17408. PMLR, 2023.

Greg Landrum, Paolo Tosco, Brian Kelley, Ric, sriniker, gedeck, Riccardo Vianello, NadineSchneider, David Cosgrove, Eisuke Kawashima, Andrew Dalke, Dan N, Gareth Jones, Brian Cole, Matt Swain, Samo Turk, AlexanderSavelyev, Alain Vaucher, Maciej Wójcikowski, Ichiru Take, Daniel Probst, Kazuya Ujihara, Vincent F. Scalfani, guillaume godin, Axel Pahl, Francois Berenger, JLVarjo, strets123, JP, and DoliathGavid. RDKit: Open-source cheminformatics. `http://www.rdkit.org`.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Stratis Limnios, Praveen Selvaraj, Mihai Cucuringu, Carsten Maple, Gesine Reinert, and Andrew Elliott. Sagess: A sampling graph denoising diffusion model for scalable graph generation. In *European Conference on Artificial Intelligence*, pages 2950–2957. IOS Press, 2024.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Meng Liu, Keqiang Yan, Bora Oztekin, and Shuiwang Ji. GraphEBM: Molecular graph generation with energy-based models. In *Energy Based Models Workshop - ICLR 2021*, 2021.

Youzhi Luo, Keqiang Yan, and Shuiwang Ji. Graphdf: A discrete flow model for molecular graph generation. In *International Conference on Machine Learning*, pages 7192–7203. PMLR, 2021.

Karolis Martinkus, Andreas Loukas, Nathanaël Perraudin, and Roger 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.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin 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.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics*, pages 4474–4484. PMLR, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1): 1–7, 2014.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max 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.

Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008a. doi: https://doi.org/10.1609/aimag.v29i3.2157.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008b.

Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2020.

Martin Simonovsky and Nikos 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.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

Rylee Thompson, Boris Knyazev, Elahe Ghalebi, Jungtaek Kim, and Graham W Taylor. On evaluation metrics for graph generative models. In *International Conference on Learning Representations*, 2022.

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. In *International Conference on Learning Representations*, 2023.

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.

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5708–5717. PMLR, 2018.

Chengxi Zang and Fei 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.

Appendix A. Proofs

A.1 Proof of the Variational Lower Bound 5

Proof Recall the notation in Section 3.1 of the main paper. To simplify the notation we consider $\mathcal{F}(G)$ as the set of any forward removal sequence of G. To prove loss (5), we start from the prior distribution of the model:

$$
\begin{aligned}
p_{\theta,\phi}(\mathcal{G}_0) &= \sum_{\mathcal{G}_{1:T}\in\mathcal{F}(\mathcal{G}_0)} p_{\theta,\phi}(\mathcal{G}_{0:T}) \\
&= \sum_{\mathcal{G}_{1:T}\in\mathcal{F}(\mathcal{G}_0)} p_{\theta,\phi}(\mathcal{G}_{0:T})\frac{q(\mathcal{G}_{1:T}|\mathcal{G}_0)}{q(\mathcal{G}_{1:T}|\mathcal{G}_0)} \qquad (23)\\
&= \sum_{\mathcal{G}_{1:T}\in\mathcal{F}(\mathcal{G}_0)} q(\mathcal{G}_{1:T}|\mathcal{G}_0)p_\theta(\mathcal{G}_T)\frac{p_{\theta,\phi}(\mathcal{G}_{0:T-1}|\mathcal{G}_T)}{q(\mathcal{G}_{1:T}|\mathcal{G}_0)} \\
&= \sum_{\mathcal{G}_{1:T}\in\mathcal{F}(\mathcal{G}_0)} q(\mathcal{G}_{1:T}|\mathcal{G}_0)p_\theta(\mathcal{G}_T)\prod_{t=1}^{T}\frac{p_{\theta,\phi}(\mathcal{G}_{t-1}|\mathcal{G}_t)}{q(\mathcal{G}_t|\mathcal{G}_{t-1})} \qquad (24)\\
&= \sum_{\mathcal{G}_{1:T}\in\mathcal{F}(\mathcal{G}_0)} q(\mathcal{G}_{1:T}|\mathcal{G}_0)\frac{p_\theta(\mathcal{G}_T)}{q(\mathcal{G}_T|\mathcal{G}_0)}p_{\theta,\phi}(\mathcal{G}_0|\mathcal{G}_1)\prod_{t=2}^{T}\frac{p_{\theta,\phi}(\mathcal{G}_{t-1}|\mathcal{G}_t)}{q(\mathcal{G}_{t-1}|\mathcal{G}_t,\mathcal{G}_0)} \\
&= \sum_{\mathcal{G}_{1:T}\in\mathcal{F}(\mathcal{G}_0)} q(\mathcal{G}_{1:T}|\mathcal{G}_0)\frac{p_\theta(\mathcal{G}_T)}{q(\mathcal{G}_T|\mathcal{G}_0)}p_\phi(n_0|\mathcal{G}_1)p_\theta(\mathcal{G}_0|n_0,\mathcal{G}_1) \\
&\qquad \cdot\prod_{t=2}^{T}\frac{p_\phi(n_{t-1}|\mathcal{G}_t)}{q(n_{t-1}|\mathcal{G}_t,\mathcal{G}_0)}\frac{p_\theta(\mathcal{G}_{t-1}|n_{t-1},\mathcal{G}_t)}{q(\mathcal{G}_{t-1}|n_{t-1},\mathcal{G}_t,\mathcal{G}_0)} \\
&= \sum_{\mathcal{G}_{1:T}\in\mathcal{F}(\mathcal{G}_0)} q(\mathcal{G}_{1:T}|\mathcal{G}_0)\frac{p_\theta(\mathcal{G}_T)}{q(\mathcal{G}_T|\mathcal{G}_0)}p_\phi(r_1|\mathcal{G}_1)p_\theta(\mathcal{W}_1|r_1,\mathcal{G}_1)\prod_{t=2}^{T}\frac{p_\phi(r_t|\mathcal{G}_t)}{q(r_t|\mathcal{G}_t,\mathcal{G}_0)}\frac{p_\theta(\mathcal{W}_t|r_t,\mathcal{G}_t)}{q(\mathcal{W}_t|r_t,\mathcal{G}_t,\mathcal{G}_0)} \\
&= \sum_{\mathcal{G}_{1:T}\in\mathcal{F}(\mathcal{G}_0)} q(\mathcal{G}_{1:T}|\mathcal{G}_0)p_\phi(r_1|\mathcal{G}_1)p_\theta(\mathcal{W}_1|r_1,\mathcal{G}_1)\prod_{t=2}^{T}\frac{p_\phi(r_t|\mathcal{G}_t)}{q(r_t|\mathcal{G}_t,\mathcal{G}_0)}\frac{p_\theta(\mathcal{W}_t|r_t,\mathcal{G}_t)}{q(\mathcal{W}_t|r_t,\mathcal{G}_t,\mathcal{G}_0)}.
\end{aligned}
$$

In step 23 we used importance sampling to change the variable over which the expectation is computed, and step 24 where we factorized the probabilities over sequences with the definition of removal and insertion processes (respectively Equations (1) and (3)). The Variational Upper Bound (VUB) is found computing the negative log-likelihood and applying the Jensen Inequality:

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

■

A.2 Proof of Equation 7

Proof Let's prove this by induction. Consider the simple case for n_1:

$$q(n_1|n_0) = B(n_1; n_0, \pi_1)$$

with $\pi_1 = 1 - q_1$. This is true by the definition of binomial transitions in Equation (6).

Now, assume the property is true for $t-1$, that is, $n_{t-1}|n_0$ is a Binomial random variable $B(n_{t-1}; n_0, \pi_{t-1})$. We know that $n_t|n_{t-1}$ is also a Binomial, and has the same distribution as $n_t|n_{t-1}, n_0$ due to the Markov property. Let's recall what their distribution and parameters are:

$$n_t|n_{t-1}, n_0 \sim B(n_t; n_{t-1}|n_0, 1-q_t)$$
$$n_{t-1}|n_0 \sim B(n_{t-1}; n_0, \pi_{t-1})$$
$$\text{with} \pi_{t-1} = \prod_{k=1}^{t-1}(1-q_k)$$

It can be proven that a Binomial r.v. conditioned on a Binomial r.v. is still a Binomial r.v. with success probability the product of the two success probabilities, and number of experiments the same as the conditioning binomial. From this fact $n_t|n_0$ is a Binomial r.v.:

$$n_t|n_0 \sim B(n_t; n_0, \pi_t)$$
$$\pi_t = (1-q_t)\pi_{t-1} = \prod_{k=1}^{t}(1-q_k)$$

■

A.3 Proof of Equation 8

Proof Let's compute the posterior probability of the binomial removals:

$$
\begin{aligned}
q(n_{t-1}|n_t, n_0) &= \\
&= q(n_t|n_{t-1})\frac{q(n_{t-1}|n_0)}{q(n_t|n_0)} \\
&= \frac{n_{t-1}!}{n_t!(n_{t-1}-n_t)!}(1-q_t)^{n_t} q_t^{n_{t-1}-n_t} \frac{\frac{n_0!}{n_{t-1}!(n_0-n_{t-1})!}\pi_{t-1}^{n_{t-1}}(1-\pi_{t-1})^{n_0-n_{t-1}}}{\frac{n_0!}{n_t!(n_0-n_t)!}\pi_t^{n_t}(1-\pi_t)^{n_0-n_t}} \\
&= \frac{(n_0-n_t)!}{(n_{t-1}-n_t)!(n_0-n_{t-1})!}\pi_{t-1}^{n_{t-1}-n_t} q_t^{n_{t-1}-n_t}\frac{(1-\pi_{t-1})^{n_0-n_{t-1}}}{(1-\pi_t)^{n_0-n_t}} \\
&= \frac{(n_0-n_t)!}{(n_{t-1}-n_t)!(n_0-n_t-(n_{t-1}-n_t))!}\pi_{t-1}^{n_{t-1}-n_t}(1-\pi_{t-1})^{n_0-n_{t-1}}\frac{q_t^{n_{t-1}-n_t}}{(1-\pi_t)^{n_0-n_t}} \\
&= \binom{n_0-n_t}{n_{t-1}-n_t}\left(q_t\frac{\pi_{t-1}}{1-\pi_t}\right)^{n_{t-1}-n_t}\left(q_t\frac{1-\pi_{t-1}}{1-\pi_t}\right)^{n_0-n_{t-1}} \\
&= \binom{n_0-n_t}{n_0-n_{t-1}}\left(\frac{1-\pi_{t-1}}{1-\pi_t}\right)^{n_0-n_{t-1}}\left(1-\frac{1-\pi_{t-1}}{1-\pi_t}\right)^{n_{t-1}-n_t}
\end{aligned}
$$

Finally, by substituting the number of failures at step t: $r_t = n_{t-1} + n_t$ we get:

$$
q(r_t|n_t, n_0) = \binom{n_0-n_t}{r_t}\left(q_t\frac{\pi_{t-1}}{1-\pi_t}\right)^{r_t}\left(\frac{1-\pi_{t-1}}{1-\pi_t}\right)^{n_0-n_t-r_t}
$$

■

A.4 Proof of Equation 12

Proof Let's prove this by induction. Consider the simple case for n_1:

$$
q(n_1|n_0) = q(r_1|n_0) = \frac{\prod_{d\in D}\binom{h(n_0)[d]}{h(r_1)[d]}}{\binom{T}{1}} = \frac{h(n_0)[r_1]}{T}
$$

where the product over denominations only one non-unit factor with $d = r_1$, because $h(r_1)[r_1] = 1$ and $h(r_1)[d] = 0$ for all other denominations, as r_1 is one of the possible choices in D. Now, assume the property is true for $t-1$, that is, $n_{t-1}|n_0$ is a Multivariate hypergeometric, that is:

$$
q(n_{t-1}|n_0) = \frac{\prod_{d\in D}\binom{h(n_0)[d]}{h(\Delta n_{t-1})[d]}}{\binom{T}{t-1}}
$$

Now, using the law of total probability:

$$
\begin{aligned}
q(n_t|n_0) &= \sum_{n_{t-1}=n_t}^{n_0} q(n_t|n_{t-1})q(n_{t-1}|n_0) \\
&= \sum_{d\in D} q(n_t|n_t+d)q(n_t+d|n_0) \\
&= \sum_{d\in D} \frac{h(n_t+d)[d]}{T-t+1} \frac{\prod_{d'\in D}\binom{h(n_0)[d']}{h(n_0-n_t-d)[d']}}{\binom{T}{t-1}} \\
&= \frac{1}{(T-t+1)\frac{T!}{(T-t+1)!(t-1)!}} \sum_{d\in D} h(n_t+d)[d] \prod_{d'\in D}\binom{h(n_0)[d']}{h(n_0-n_t-d)[d']} \\
&= \frac{1}{t}\frac{1}{\frac{T!}{(T-t)!t!}} \sum_{d\in D}(h(n_t)[d]+1)\frac{h(n_0)[d]!}{(h(n_0)[d]-h(n_0-n_t-d)[d])!h(n_0-n_t-d)[d]!} \\
&\qquad \cdot \prod_{d'\in D\setminus\{d\}}\binom{h(n_0)[d']}{h(n_0-n_t-d)[d']} \\
&= \frac{1}{t}\frac{1}{\binom{T}{t}} \sum_{d\in D}(h(n_t)[d]+1)\frac{h(n_0)[d]!}{(h(n_0)[d]-h(n_0-n_t)[d]+1)!(h(n_0-n_t)[d]-1)!} \\
&\qquad \cdot \prod_{d'\in D\setminus\{d\}}\binom{h(n_0)[d']}{h(n_0-n_t)[d']} \\
&= \frac{1}{t}\frac{1}{\binom{T}{t}} \sum_{d\in D}(h(n_t)[d]+1)\frac{h(n_0)[d]!}{(h(n_t)[d]+1)!(h(n_0)[d]-h(n_t)[d]-1)!} \\
&\qquad \cdot \prod_{d'\in D\setminus\{d\}}\binom{h(n_0)[d']}{h(n_0-n_t)[d']} \\
&= \frac{1}{t}\frac{1}{\binom{T}{t}} \sum_{d\in D} h(n_0-n_t)[d]\frac{h(n_0)[d]!}{h(n_t)[d]!(h(n_0)[d]-h(n_t)[d])!} \prod_{d'\in D\setminus\{d\}}\binom{h(n_0)[d']}{h(n_0-n_t)[d']} \\
&= \frac{1}{t}\frac{1}{\binom{T}{t}} \sum_{d\in D} h(n_0-n_t)[d]\binom{h(n_0)[d]}{h(n_0-n_t)[d]} \prod_{d'\in D\setminus\{d\}}\binom{h(n_0)[d']}{h(n_0-n_t)[d']} \\
&= \frac{1}{t}\frac{1}{\binom{T}{t}} \sum_{d\in D} h(n_0-n_t)[d] \prod_{d'\in D}\binom{h(n_0)[d']}{h(n_0-n_t)[d']} \\
&= \frac{1}{t}\frac{\prod_{d'\in D}\binom{h(n_0)[d']}{h(n_0-n_t)[d']}}{\binom{T}{t}} \sum_{d\in D} h(n_0-n_t)[d] \\
&= \frac{1}{t}\frac{\prod_{d'\in D}\binom{h(n_0)[d']}{h(n_0-n_t)[d']}}{\binom{T}{t}} t \\
&= \frac{\prod_{d'\in D}\binom{h(n_0)[d']}{h(\Delta n_t)[d']}}{\binom{T}{t}}
\end{aligned}
$$

To reach the final statement we used the following facts:

- $h(n+d)[d'] = \begin{cases} h(n)[d]+1 & \text{for } d'=d \\ h(n)[d'] & \text{otherwise} \end{cases}$
- $h(n_0) - h(n_t) = h(n_0 - n_t)$
- by definition $\sum_{d\in D} h(n_0 - n_t)[d] = t$

■

A.5 Proof of Equation 13

Proof Let's compute the posterior:

$$
\begin{aligned}
q(n_{t-1}|n_0,n_t) &= \frac{q(n_t|n_{t-1})qp(n_{t-1}|n_0)}{q(n_t|n_0)} \\
&= \frac{h(n_{t-1})[r_t]}{T-t+1} \frac{\prod_{d\in D} \frac{h(n_0)[d]!}{h(\Delta n_{t-1})[d]!h(n_{t-1})[d]!}}{\frac{T!}{(t-1)!(T-t+1)!}} \left(\frac{\prod_{d\in D} \frac{h(n_0)[d]!}{h(\Delta n_t)[d]!h(n_t)[d]!}}{\frac{T!}{t!(T-t)!}} \right)^{-1} \\
&= \frac{h(n_{t-1})[r_t](t-1)!(T-t+1)!}{t!(T-t)!(T-t+1)} \prod_{d\in D} \frac{h(\Delta n_t)[d]!h(n_t)[d]!}{h(\Delta n_{t-1})[d]!h(n_{t-1})[d]!}
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{h(n_{t-1})[r_t]}{t} \prod_{d\in D} \frac{h(\Delta n_t)[d]!h(n_t)[d]!}{h(\Delta n_{t-1})[d]!h(n_{t-1})[d]!} \\
&= \frac{h(n_{t-1})[r_t]}{t} \prod_{d\in D} \frac{h(\Delta n_{t-1}+r_t)[d]!h(n_t)[d]!}{h(\Delta n_{t-1})[d]!h(n_t+r_t)[d]!} \\
&= \frac{h(n_{t-1})[r_t]}{t} \frac{h(\Delta n_{t-1}+r_t)[r_t]!h(n_t)[r_t]!}{h(\Delta n_{t-1})[r_t]!h(n_t+r_t)[r_t]!} \prod_{d\in D\setminus\{r_t\}} \frac{h(\Delta n_{t-1})[d]!h(n_t)[d]!}{h(\Delta n_{t-1})[d]!h(n_t)[d]!} \\
&= \frac{h(n_{t-1})[r_t]}{t} \frac{(h(\Delta n_{t-1})[r_t]+1)!h(n_t)[r_t]!}{h(\Delta n_{t-1})[r_t]!(h(n_t)[r_t]+1)!} \\
&= \frac{h(n_t)[r_t]+1}{t} \frac{h(\Delta n_{t-1})[r_t]+1}{h(n_t)[r_t]+1} \\
&= \frac{h(\Delta n_t)[r_t]}{t}
\end{aligned}
$$

Because $q(n_{t-1}|n_0,n_t) = q(r_t|n_0,n_t)$:

$$q(r_t|n_0,n_t) = \frac{h(\Delta n_t)[r_t]}{t}$$

■